Model Predictive Control

ME499 Final Project Report

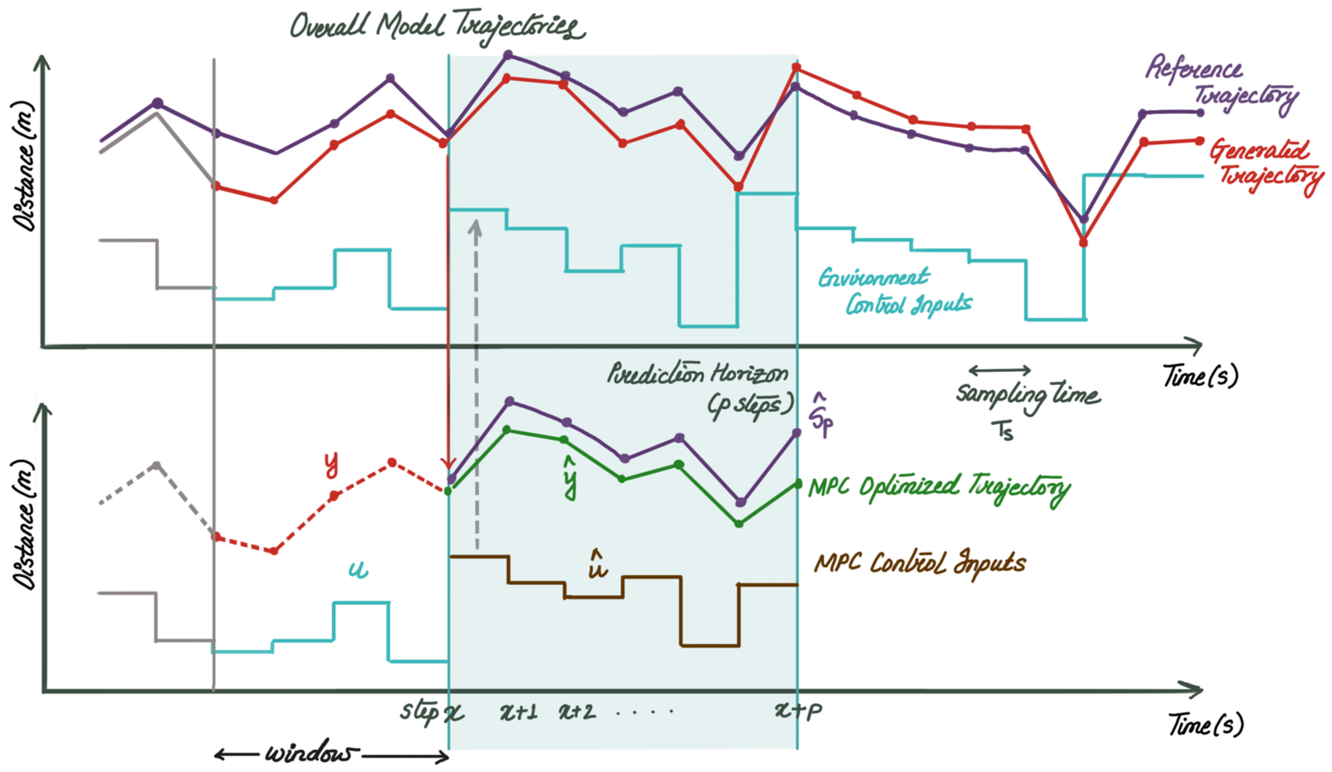
Prarthana Chakrabarti
Northwestern University

May 30 2024

I. Introduction to Model Predictive Control:

Model Predictive Control is one of the only advanced control philosophies that has been successfully implemented in practice. It is in essence a Feed-Forward Controller which attempts to calculate the optimal control actions to minimize the deviation of a system from an ideal trajectory by predicting future changes in the dependent variable and accounting for system disturbances. MPC systems account for constraints applied on both the manipulated and calculated variables and are an excellent choice for multivariate systems. Solving the optimization over a finite horizon also increases the sampling time of the controller, which improves its ability to make predictions on the future states of the system.

Some practical implementations of MPC include autonomous vehicular control, temperature control and chemical plants.



The components of a Model Predictive Control system can be defined as follows:

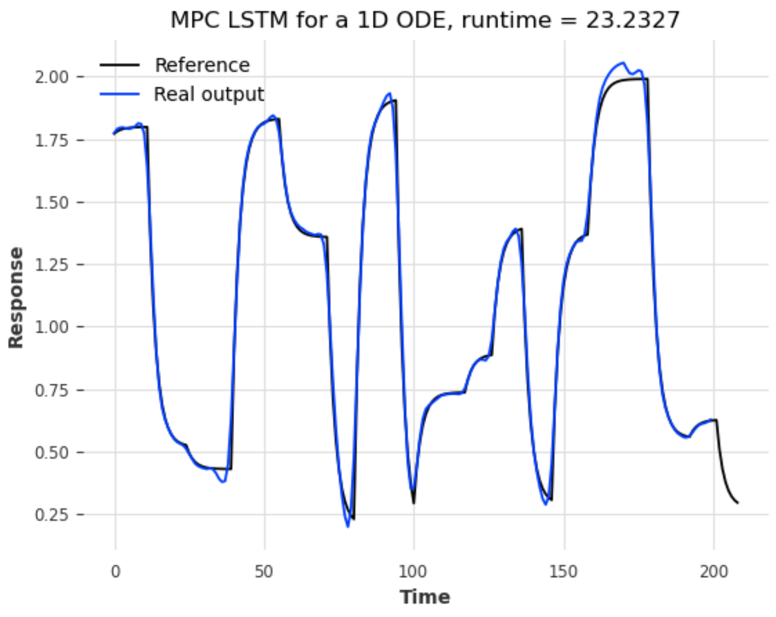
- P: Prediction Horizon
- M: Control Horizon
- $S^{\wedge}_{p}$: Reference trajectory over the Prediction Horizon
- y: initial value/past output
- $u^{\wedge}$: control input
- $y^{\wedge}$: predicted output

- u: initial value/past input
- y: initial value/past output
- window: length of past inputs

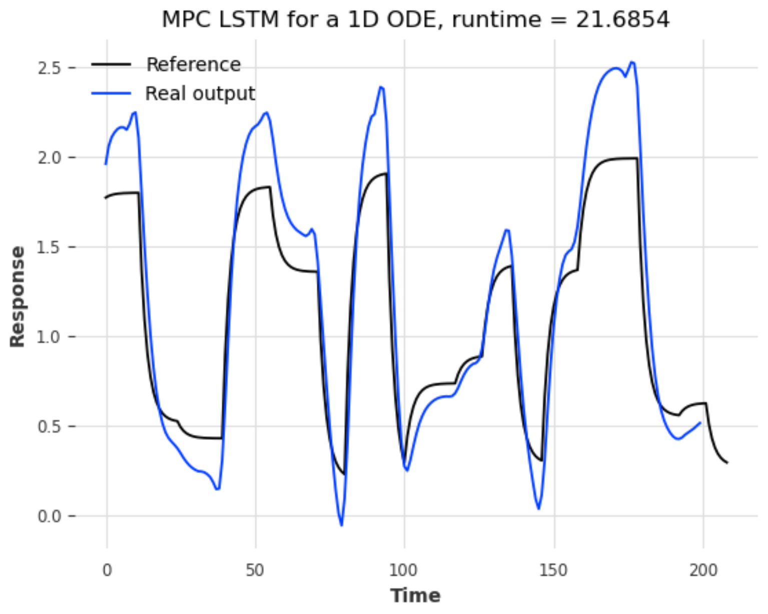
Changing the Window in MPC:

Experimenting with reducing the window in an MPC system from 5 to 1:

- **Case 1: Window = 5**



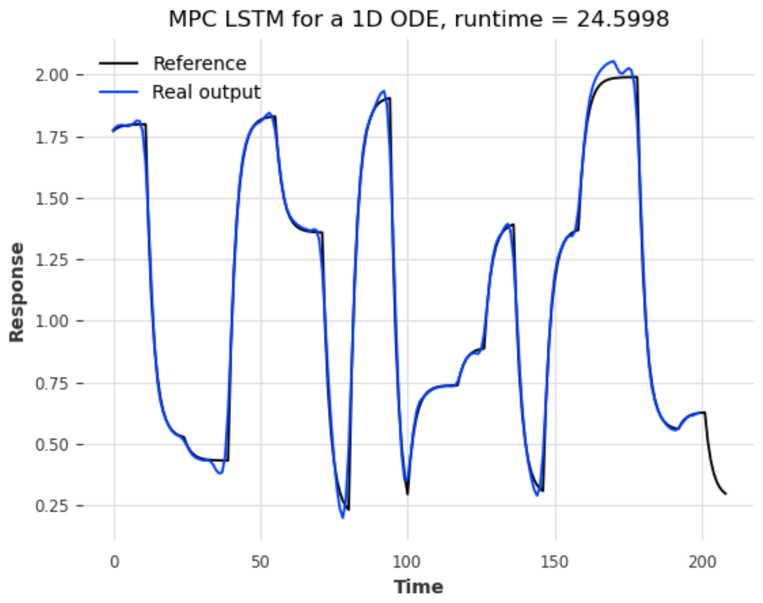
- **Case 2: Window = 1**



This reduces the runtime by 7% but the performance is demonstrably poorer.
This is because reducing the window size inherently reduces the amount of past data that the system processes and consequently the associated computational load. This also

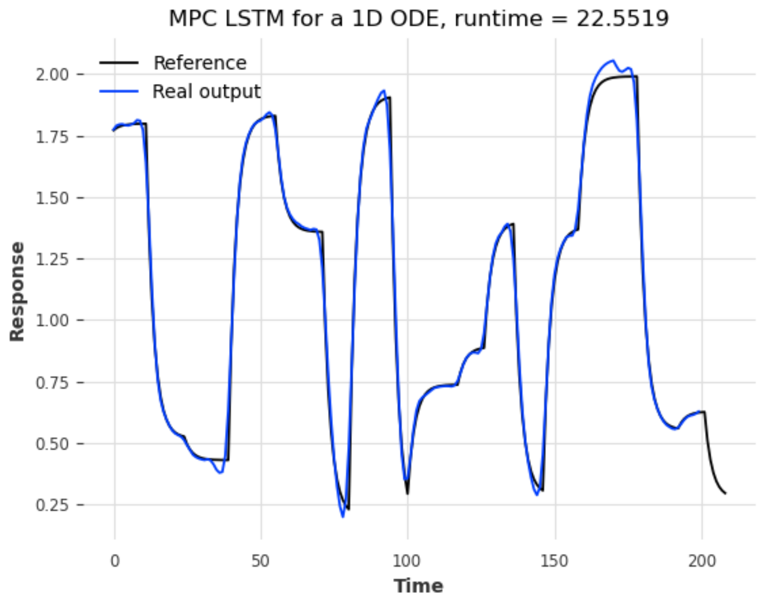
makes systems with smaller windows easy to tune. However, with less data, the measurement noise and disturbances in the system have a higher impact. It also offers less information for the system to utilize, leading to less accurate control policies.

Solving the Model Predictive Control Objective Function:



L-BFGS-B (Limited-Memory Broyden-Fletcher-Goldfarb-Shanno with Bounds) is the optimization algorithm used to minimize the MPC Objective Function in the original code. L-BFGS-B is a good candidate for this kind of optimization problem as it calculates an approximation of the Hessian at each step, making it more computationally efficient which in turn makes it ideal for handling multivariate systems.

- ***Switching to SLSQP with Autograd:***



Sequential Least Squares Programming (SLSQP) is an algorithm that minimizes a multi-variable function while incorporating bounds and equality and inequality constraints. SLSQP is a good candidate for minimizing the MPC algorithm as it performs well for objective functions that are twice continuously differentiable.[4]

To minimize the constrained objective function, the SLSQP algorithm solves a KKT problem with a Lagrangian Equation of the form:

$$\boldsymbol{L(x,\lambda,\mu) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{p} \mu_i\, g_i(x)}$$

Where f is the objective function, h and g represent equality and inequality constraints respectively and λ and μ are the Lagrange multipliers associated with them.

To solve the objective function, the SLSQP algorithm then approximates the original model to a quadratic problem and linearizes the constraints to create Quadratic Subproblems of the form:

$$\begin{aligned} \min_{d} \quad & f(x_k) + \nabla f(x_k)^T d + \tfrac{1}{2} d^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k, \sigma_k) d \\ \text{s.t.} \quad & h(x_k) + \nabla h(x_k)^T d \geq 0 \\ & g(x_k) + \nabla g(x_k)^T d = 0. \end{aligned}$$

It then solves these Quadratic sub-problems by finding a search direction *d* and conducting a line search of the form: $\mathbf{x_{k+1} = x_k + \alpha d}$ until the convergence criteria is met. To improve the accuracy and performance of the SLSQP algorithm, I integrated it with the use of Autograd to calculate the gradients of the objective function. In future work, I will also be expanding the system to include constraints, and incorporate the Autograd gradients of the same into the algorithm. Autograd calculates gradients using reverse-mode differentiation in a single backward pass, which is significantly faster and more accurate than manually computing gradients.

II. LSTM Networks to Simulate MPC Systems:

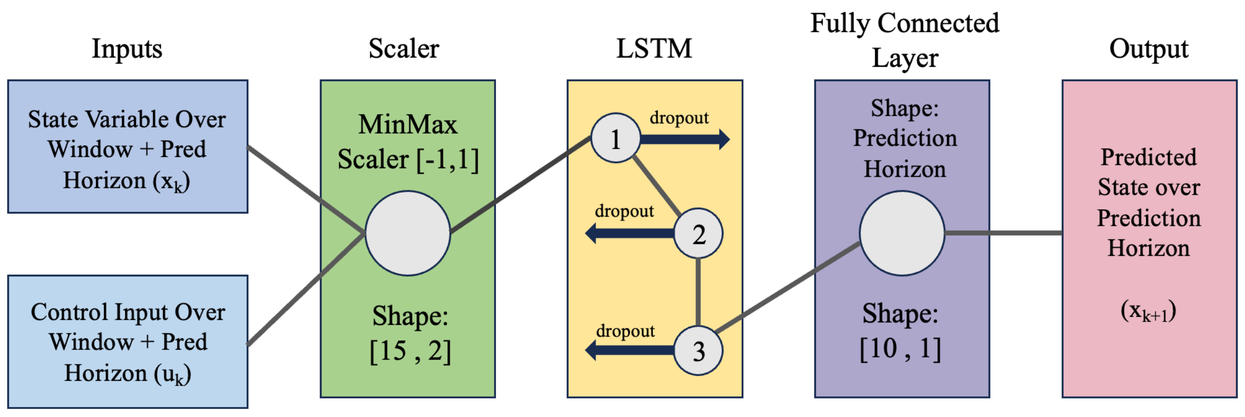
Recurrent neural networks are often used as surrogates to traditional MPC systems. Since LSTMs utilize data driven learning, they are resilient to changes in operating parameters and disturbances, which allows controllers to combine the robustness of neural networks with the control philosophy of MPC.
Long Short-Term Memory networks (LSTMs), a type of Recurring Neural Network that are one of the mechanisms used to create a black box simulation of the MPC system in this project. Traditional RNNs generally run into issues with exploding or vanishing gradients while tracking

long term dependencies. LSTMs are able to address these issues by controlling the flow of information by implementing mechanisms like memory cells and gates. Some popular implementations of LSTM based MPC systems are Energy Management Systems where they can be used to optimize the temperature set points and autonomous vehicular control systems.

The components of a LSTM system are as follows:

- **Forget Gate:** This gate decides how much of the past information should be discarded from the cell state. The formulation takes into account the previous hidden state and current input values.
- **Input Gate:** This gate determines what information gets updated to the cell state from the current input.
- **Memory Cell or Cell State:** This component retains old information and combines it with new inputs to preserve long term dependencies. It is often called the 'core' of an LSTM system.
- **Output Gate:** This gate controls what information from the cell state gets converted to output and what parts are hidden.

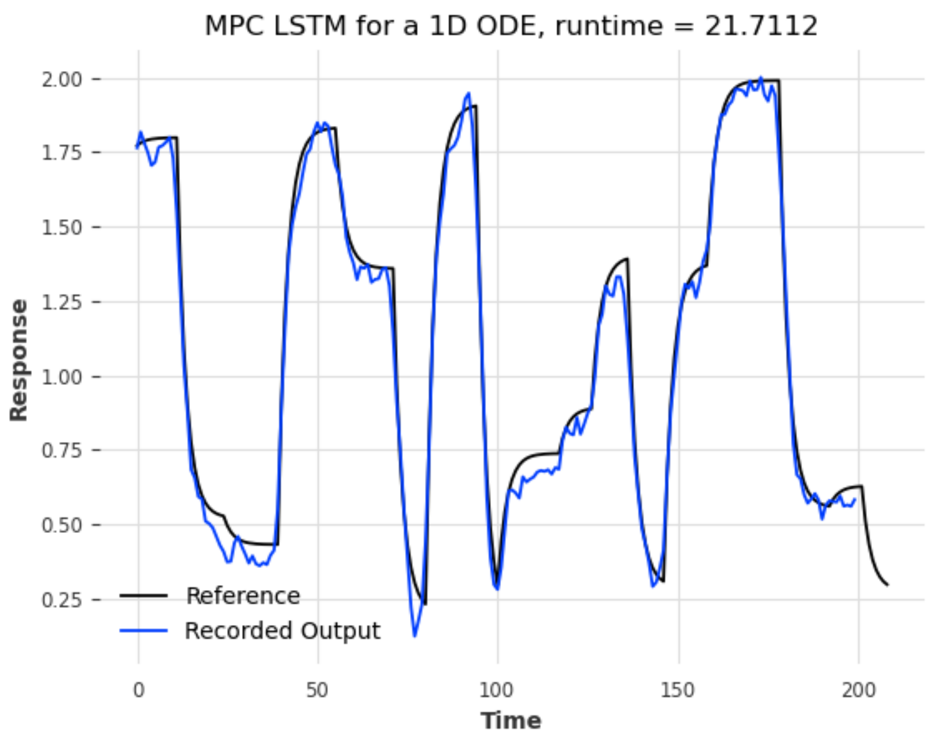
The LSTM surrogate model for the MPC System in our case can be represented as:


Inputs
Scaler
LSTM
Fully Connected Layer
Output
State Variable Over Window + Pred Horizon (x_k)
Control Input Over Window + Pred Horizon (u_k)
MinMax Scaler [-1,1]
Shape: [15 , 2]
1
2
3
dropout
dropout
dropout
Shape: Prediction Horizon
Shape: [10 , 1]
Predicted State over Prediction Horizon
(x_{k+1})

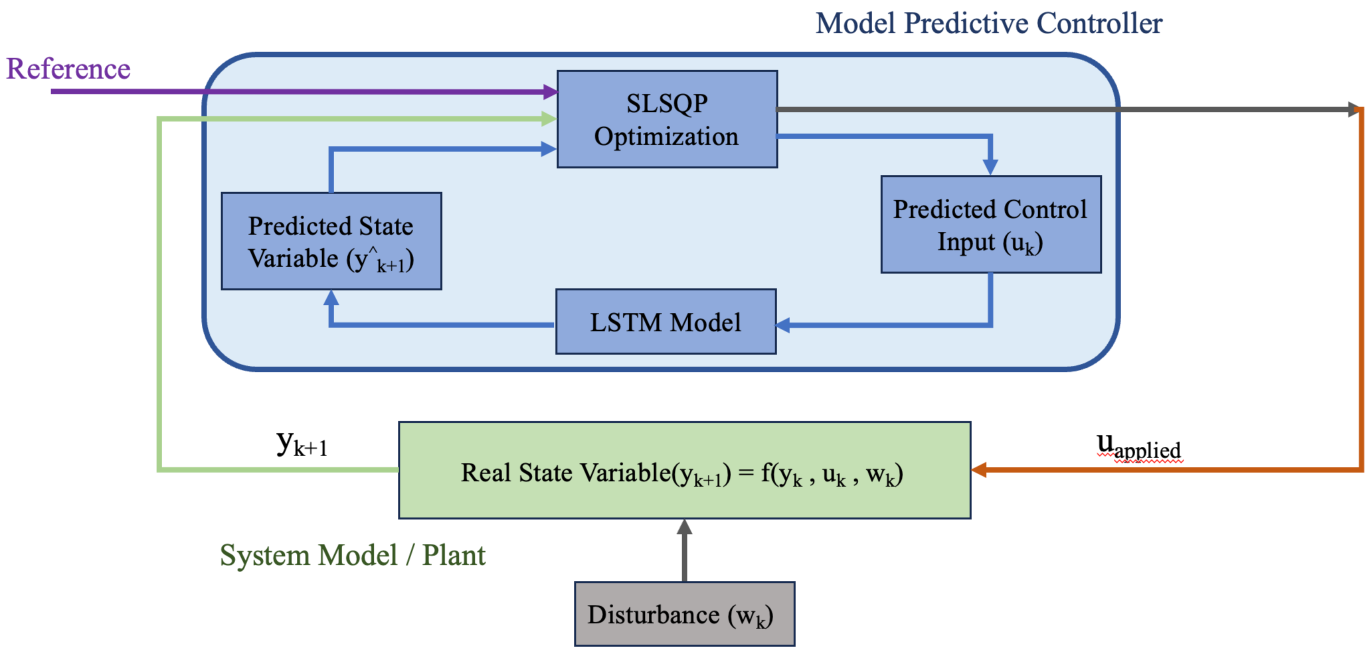
III. Robust MPC:

Robust MPC is a subtype of MPC which aims to explicitly handle constraints and guarantee the performance of the system under the worst possible case. The worst possible case for a system could involve control effort, tracking errors and constraint satisfaction under uncertainties and unpredictable circumstances in the system. Traditional MPC systems rely on accurately modeling the system dynamics to predict future states. This can get more and more unrealistic as non-linearities are introduced, introducing errors in the system. Moreover, we always run the risk of parametric uncertainties in the variables of the equations or external disturbances.

I modelled uncertainties on the system by introducing a random noise term in the real system black-box.



The MPC system in our project can be illustrated as:


Model Predictive Controller
Reference
SLSQP Optimization
Predicted State Variable ($\hat{y}_{k+1}$)
Predicted Control Input (u_k)
LSTM Model
y_{k+1}
Real State Variable(y_{k+1}) = f(y_k , u_k , w_k)
$u_{applied}$
System Model / Plant
Disturbance (w_k)

IV. Solving Robust Model Predictive Control Problems:

Robust MPC Problems are historically tackled in one of two ways [2]:

1. **Min-Max Formulations:**

   Mix-Max formulations are designed by considering the worst possible scenario and aim to then optimize it – resulting in very cautious or conservative control strategies. These algorithms first consider all possible disturbances that might affect the system performance, where each disturbance is bounded but otherwise unpredictable within the bounds.

   These functions then try to minimize the maximum possible loss function, resulting in an objective function structure that looks like:

   $$\min_{u} \max_{d} J(x, u, d)$$

   The inner Maximization function seeks to find the maximum possible disturbance in the set that would lead to the worst-case scenario for the control input. Then the outer minimization problem then finds the best possible control value for this worst-case scenario. Sine the Min-Max formulations deal with every possible scenario, they are inherently infinite-dimensional and generate over-arching control policies that ensure robustness for all possible states of the system. However, the need to solve maximization and minimization problems at every step makes these algorithms extremely computationally inefficient.

2. **Tubular Model Predictive Control:**

   Tubular Model Predictive Control is a strategy that involves calculating an 'ideal' or nominal trajectory using standard MPC and then adding a Feedback Controller to it. This Feedback Controller adjusts for any deviations from the ideal trajectory while ensuring that the system stays within a safe zone or 'tube' around it. This technique is far simpler than Min-Max formulations as it focuses on control sequences (much like standard MPC) unlike the control policies generated by the latter. It also ensures that despite external disturbances and uncertainties in the system, its deviation remains within controlled limits.

   The first step in formulating a Tubular Model is computing the nominal trajectory by solving an MPC Optimization Problem:

   $$J_{nom} = \int_{0}^{T} ((x_{nom}(t)^{T}\, Q\, x_{nom}(t) + (u_{nom}(t)^{T} R u_{nom}(t))dt$$

   This is essentially the 'ideal' path the system must follow in the absence of external disturbances.

Following that, the tube is defined by the constraints that bound the deviation: $\delta \dot{x} = A(t)\delta x + B(t)\delta u + w$. The 'tube' defined through these constraints ensures robustness against uncertainties and disturbances by ensuring that the real trajectory does not deviate significantly from the nominal trajectory.

Finally, a Feedback control law is applied to the system to correct the deviations: $u(t) = u_{\text{nom}}(t) + K(t)\delta x(t)$. Here, K(t) is a gain matrix to stabilize the system within the tube.
Since this Feedback controller does not need to consider all possible cases, but simply deviations from the 'ideal' trajectory, the complexity of the algorithm and subsequently the computational load are lower than a Min-Max Formulation.

V. Linearization:

As much as Tubular Model Predictive Control is an improvement on the computational efficiency of the Min-Max Formulation, in real time applications of MPC such as autonomous vehicular control or building management systems, the system dynamics can be complex. These dynamics lead to nonconvex optimization problems that are computationally expensive especially for low-tech control devices like PLCs (programmable logic controllers) and may not respond fast enough for vehicular control.
Hence instant linearization can be applied at each operating point of a non-linear machine learning model to make these problems more computationally practical. The operating point is essentially the current state at every control interval (e.g. temperature, current position, etc.). Since this linearization is performed at every control interval, it reflects the most recent operating conditions of the system and allows the algorithm to perform with quite high accuracy.
Once the Instant Linearization has been performed, a Linear Quadratic Regulator Control strategy can be used to minimize the quadratic cost function.

Expanding on the Linear Quadratic Regulator (LQR):

The LQR cost function is a least squares function minimizing the difference between the desired and actual system state and control input over the Prediction Horizon defined. Since this policy is inherently linear with respect to the state of the system, it makes the computational cost significantly lower.

The objective function for the LQR looks like:

$$J = \int_0^\infty (x^T Qx + u^T Ru)dt$$

- x(t) is the state vector at a time t
- u(t) is the control input vector at a time t
- Q is the positive semidefinite matrix of weights for the state vector
- R is the positive definite matrix of weights for the control input vector

Q and R are user-defined with the motivation to penalize deviation from ideal trajectory and large variations in control input respectively.

In a LQR Formulation, the system dynamics are given as: $\dot{x}(t) = Ax(t) + Bu(t)$ where A and B are the system matrices defining the effect of the state and control input respectively. Subsequently, the control law for LQR Formulations is of the form: $u(t) = -Kx(t)$ where K is the Gain Matrix. To solve the gain matrix, one must solve Riccati's equation as discussed next.

Riccati's Equation:

The Algebraic Riccati's Equation is defined as:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

- Q is the positive semidefinite matrix of weights for the state vector
- R is the positive definite matrix of weights for the control input vector
- A is the system matrix defining the effect of state
- B is the system matrix defining the effect of control input
- ***P is the solution of the Riccati's Equation***

The Optimal Gain can then be calculated from this equation as: $K = R^{-1}B^T P$

The aim of this project is to implement instant linearization for a system to efficiently simulate a simplified tubular model predictive control algorithm that is less computationally expensive.

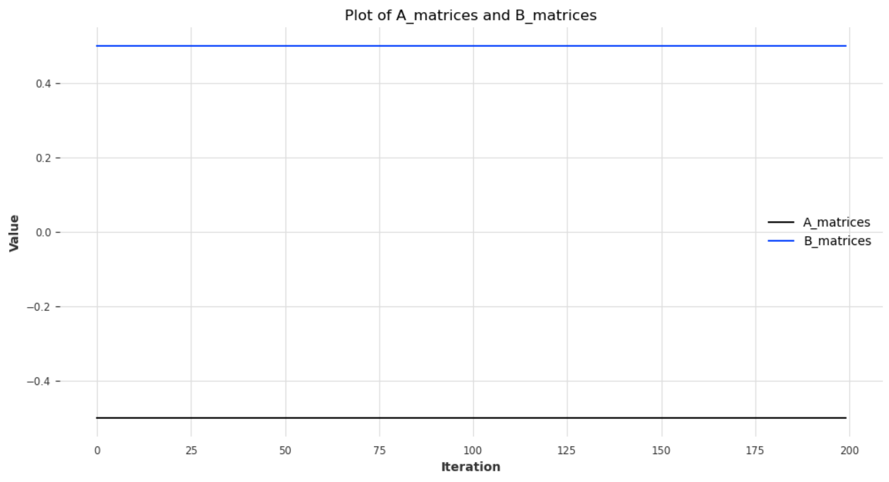
VI. Evaluating State Matrices:

a. Evaluating State Matrices of the Continuous System:

While we are using an LSTM network as a black box for this system, the process dynamics of the dummy system are:

$$dx/dt = (-x + K * u) / \tau \text{ (corresponding to } dx/dt = Ax(t) + Bu(t))$$
$$\text{where } K = 1,\ \tau = 2$$

The continuous state matrices of the system can be evaluated for each Time Steps as: $A = -1 / \tau$ and $B = K / \tau$. This is also verified in code using the autograd function on the process model:

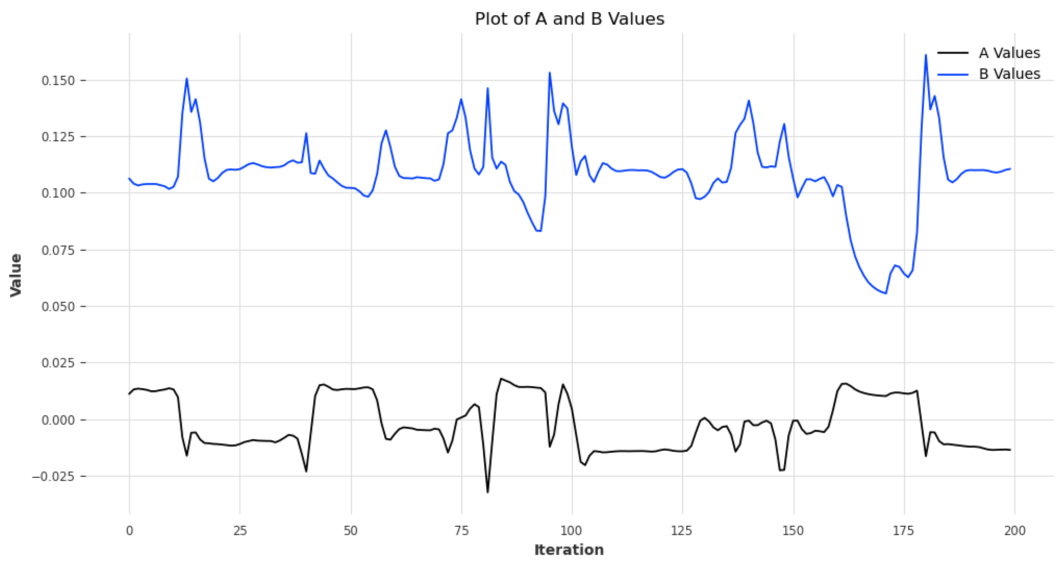


For this system, the system output is considered to be same as the controller output, hence **y = x** and **C = 1, D = 0.**

b. Calculating the Gradients of the Output Tensor - Torch.Autograd

As defined in (Yang, 2022), the state matrices A and B can be defined as the Jacobian of the output state with respect to the past state vector and the control input.
A significant portion of this project was focused on estimating the state matrix Jacobians of the LSTM neural network simulating the Model Predictive Control system.

Following is the plot of A and B values obtained by using torch.autograd to automatically calculate gradients of the system with respect to the output of the LSTM:



c. *Attempting to Verify Results via Discretization:*

For the purposes of demonstrating instant linearization while using LSTM as a black box to simulate MPC, it is assumed that the system dynamics of the original model are unknown. To verify the state matrices calculated by torch.autograd, we attempted to cross verify the results with state matrices calculated by discretizing the State Matrices of the original system at each time step referencing (Ławryńczuk, 2014). [3]

The methodology for discretizing the original continuous system was as follows:

The original system dynamics can be defined as:

$$\dot{x} = F(x, u), \quad y = Z(x, u)$$

The linearized system dynamics can be defined for sampling time T_s as:

$$\dot{x}_L = A_c x_L + B_c u_L + K_c$$
$$y_L = C_c x_L + D_c u_L$$

Where the continuous system state matrices are:

$$A_c = \partial F / \partial x \Big|_{u = u_{OP}}^{x = x_{OP}}$$

$$B_c = \partial F / \partial u \Big|_{u = u_{OP}}^{x = x_{OP}}$$

$$C_c = \partial Z / \partial x \Big|_{u = u_{OP}}^{x = x_{OP}}$$

$$D_c = \partial Z / \partial u \Big|_{u = u_{OP}}^{x = x_{OP}}$$

As defined in (Ławryńczuk, 2014), the discretized system dynamics can be approximated to be:

$$x[k+1] = A_d\, x[k] + B_d\, u[k] + K$$
$$y[k] = C_d\, x[k] + D_d\, u[k]$$

Where the discrete time state matrices can be defined as:

$$A_d = e^{A_c T_s}$$
$$B_d = A_c^{-1}(e^{A_c T_s} - I)\, B_c$$
$$K_d = A_c^{-1}(e^{A_c T_s} - I)\, K_c$$

For the process dynamics of the system we are considering:

$$\mathbf{A_c = -0.5,\ B_c = 0.5\ \text{and}\ T_s = 1s}$$

Subsequently, the values of the discretized state matrices are:

$$A_d = e^{A_c T_s} = e^{-0.5} = 0.61$$

$$B_d = A_c^{-1}(e^{A_c T_s} - I)\, B_c = -0.5^{-1}(e^{-0.5} - I)\ 0.5 = 0.394$$

d. *Understanding the Discrepancy in Results – Input Scaling:*

As demonstrated, there is a discrepancy in the state space matrices calculated via torch.autograd and the expected discrete time state space matrices. We were able to recognize the reason for this discrepancy as the scaling applied to the input values of the LSTM network.
The inputs of the LSTM are first scaled via a [-1, 1] MinMax Scaler.
MinMax Scalars are usually of the form:

$$X_{scaled} = a + ((X - X_{min})\ (b - a) / (X_{max} - X_{min}))\ \text{from}\ [a\ ,\ b]$$

For a [-1 , 1] MinMax Scalar, this equation becomes:

$$X_{scaled} = -1 + (2(X - X_{min}) / (X_{max} - X_{min}))$$

It is not possible to mathematically quantify the effect of this scalar on the gradients calculated via torch.autograd due to the inherent non-linearity of LSTM cells which contain activation functions such as sigmoids and tanh in the gates. It is also impossible to isolate the impact of each input in a multivariate time series system such as the one we are using. Finally, due to the fact that gradients in the LSTM are calculated using backpropagation, the scaling factor also propagates through the system using chain rule creating dependencies that cannot be directly quantified.

We theorize that despite the values of the state matrices being scaled, they can still be used to adequately model a tubular model predictive control system. Testing the feasibility of this hypothesis is one of the next steps of this research.

VII. Ancillary Control:

A lower complexity strategy to improve the robustness of an MPC system is to add a supplementary controller to the system. This ancillary controller runs an additional control philosophy over the sequences generated by MPC to further refine the control input applied to the system. This allows the ancillary controller to compensate for any errors and add a layer of robustness against uncertainties in the system.
Ancillary controllers can be of many types, with the simplest one being PID controllers. Some other variations include State Observer Controllers: where mathematical modelling and measurements are used to estimate the current state of a system, which is then used to predict the

magnitude of compensation applied to the control input and Disturbance Observer Controllers which aim to predict the disturbance in the system to generate control philosophies. Ancillary controllers can be applied in Sequential, Parallel or Hierarchical fashion.

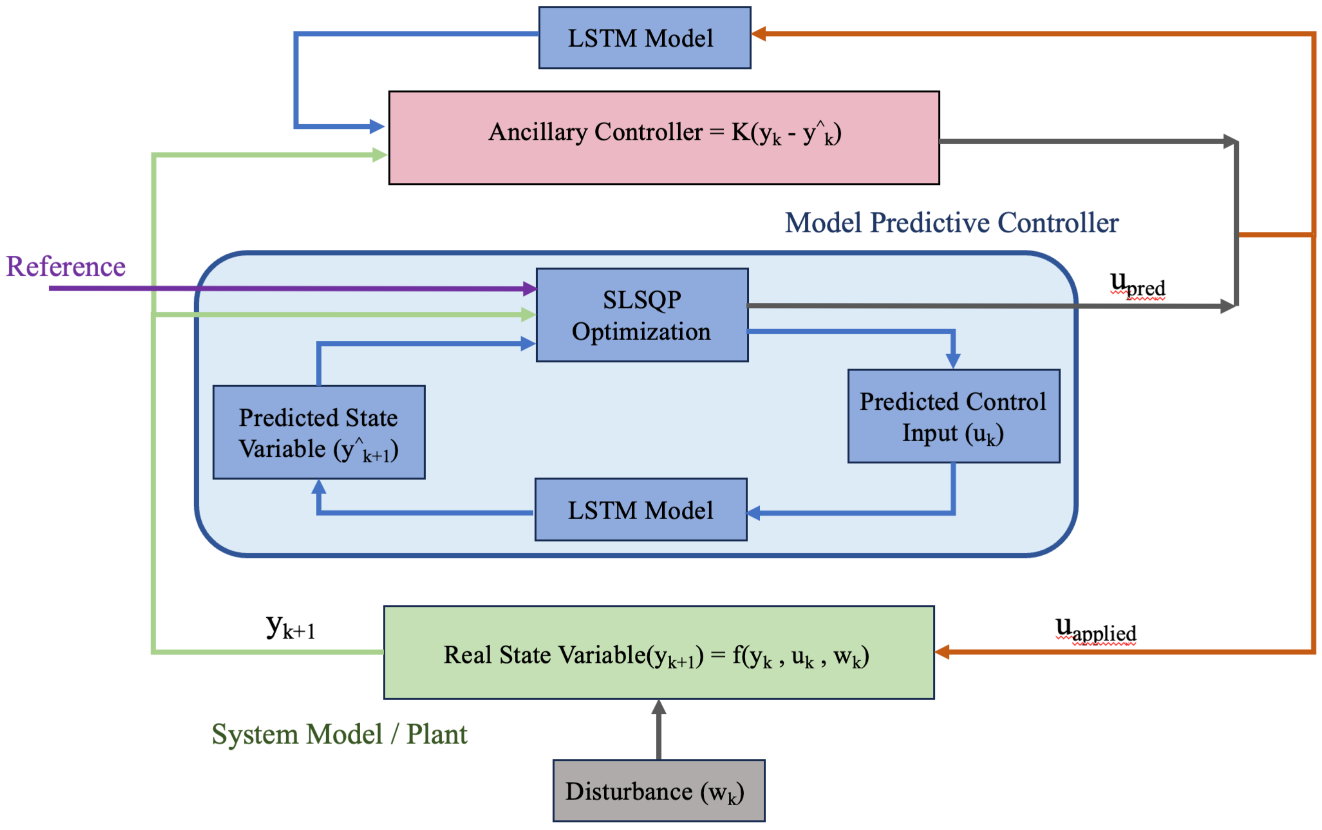
For our system, the control input applied is:

$$\mathbf{U_{(applied)K+1} = U_{(pred)K+1} - (Constant) * (y_{(real)k} - y^{\wedge}_{(pred)k})}$$

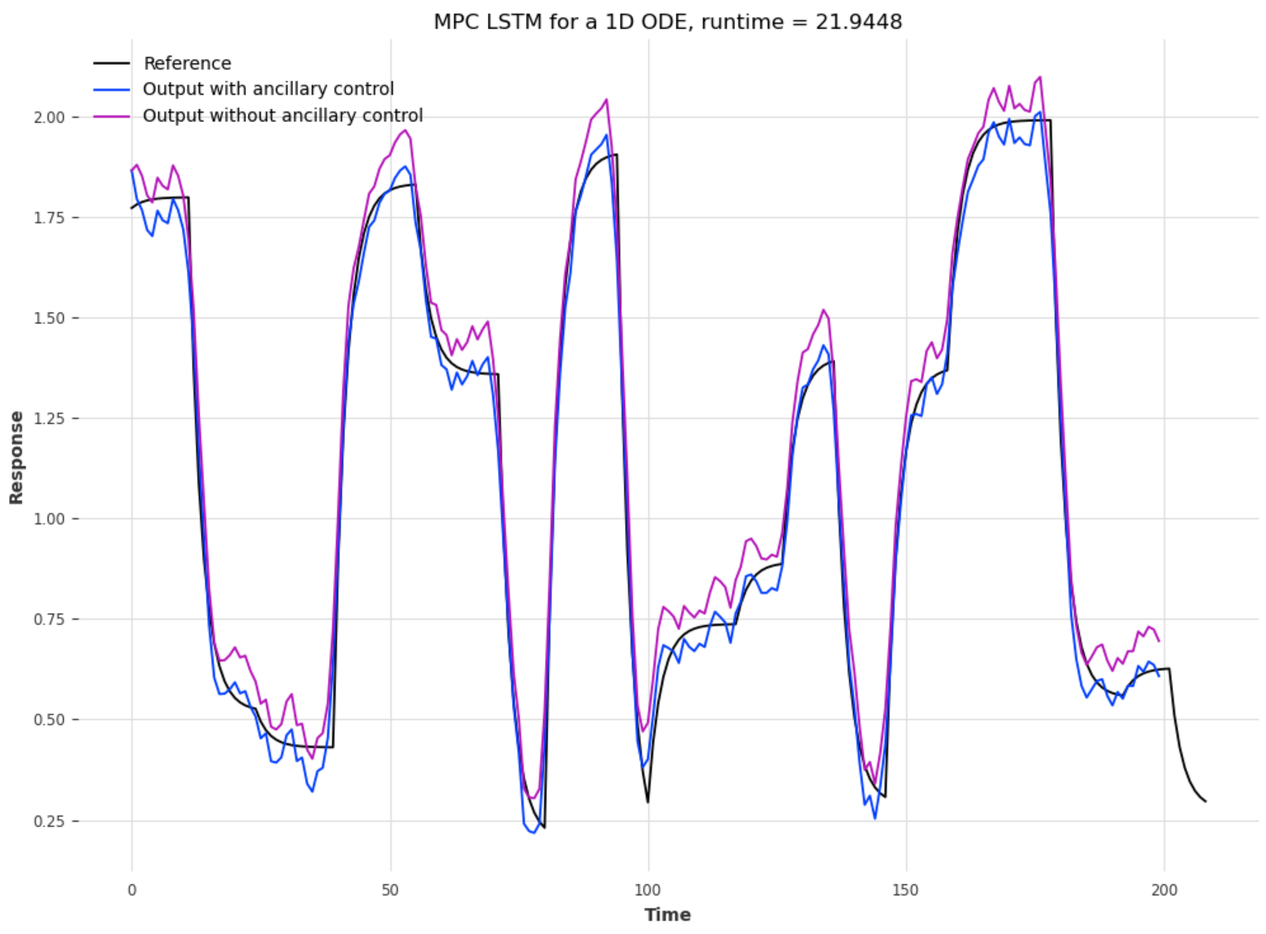
Where:

- $\mathbf{U_{(pred)K+1}}$ is the control input predicted by the LSTM
- $\mathbf{(Constant) * (y_{(real)k} - y^{\wedge}_{(pred)k})}$ is the Ancillary Controller

This controller is a PID controller and it was applied in parallel to the MPC Controller. When applied to the system, the ancillary controller can be demonstrated as:


LSTM Model
Ancillary Controller = K(y_k - y^_k)
Model Predictive Controller
Reference
SLSQP Optimization
u_pred
Predicted State Variable (y^_k+1)
Predicted Control Input (u_k)
LSTM Model
y_k+1
Real State Variable(y_k+1) = f(y_k , u_k , w_k)
u_applied
System Model / Plant
Disturbance (w_k)

Applying the ancillary controller significantly improves the performance of the system in the presence of uncertainties. To demonstrate the improvement in the system, a random noise in the range of [-0.05 to 0.05] was applied to the system and the state of the system was observed with and without an ancillary controller.



VIII. Future Work:

The two main goals I hope to work on in the future are adding constraints to the system and modeling Robust MPC with instant linearization using the scaled state matrices from torch.autograd. Adding constraints to the system would allow me to demonstrate the full scope of Robust MPC in maintaining system performance and allow us to evaluate the performance of the instant linearization algorithm.

IX: Acknowledgements:

I would like to thank Professor Chen for allowing me the opportunity to pursue this project as it has been an incredible learning opportunity for me. My greatest appreciation to Yi-Ping Chen and Ying-Kuan Tsai (Rick) for being such patient teachers to me throughout this process. This process has taught me a lot about the intricacies of advanced control philosophies and I hope to continue this work in the future.

X. References:

[1] Shiyu Yang, Man Pun Wan, Machine-learning-based model predictive control with instantaneous linearization – A case study on an air-conditioning and mechanical ventilation system, Applied Energy, Volume 306, Part B, 2022, https://doi.org/10.1016/j.apenergy.2021.118041.

[2] Singh, S., Pavone, M., & Slotine, J.E. (2016). Tube-Based MPC : a Contraction Theory Approach.

[3] M. Ławryńczuk, "Model predictive control with on-line optimal linearization," 2014 IEEE International Symposium on Intelligent Control (ISIC), Juan Les Pins, France, 2014, pp. 2177-2182, Doi: 10.1109/ISIC.2014.6967645.

[4] *SLSQP#*. SLSQP - Qiskit Algorithms 0.3.0. (2024, April 10). https://qiskit-community.github.io/qiskit-algorithms/stubs/qiskit_algorithms.optimizers.SLSQP.html

[5] Alexander Winkler, Weizhou Wang, Armin Norouzi, David Gordon, C.R. Koch, Jakob Andert, Integrating Recurrent Neural Networks into Model Predictive Control for Thermal Torque Derating of Electric Machines, Volume 56, Issue 2, 2023, https://doi.org/10.1016/j.ifacol.2023.10.1010.

[6] K. Huang, K. Wei, F. Li, C. Yang and W. Gui, "LSTM-MPC: A Deep Learning Based Predictive Control Method for Multimode Process Control," in *IEEE Transactions on Industrial Electronics*, vol. 70, no. 11, pp. 11544-11554, Nov. 2023, doi: 10.1109/TIE.2022.3229323.

[7] *Rufflewind's scratchpad*. Reverse-mode automatic differentiation: a tutorial - Rufflewind's Scratchpad. (2016b, December 30). https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation

[8] Mohammad Alsalti, Victor G. Lopez, Julian Berberich, Frank Allgöwer, Matthias A. Müller, Data-driven nonlinear predictive control for feedback linearizable systems, IFAC-PapersOnLine, Volume 56, Issue 2, 2023, https://doi.org/10.1016/j.ifacol.2023.10.1636.

[9] Walter Fetter Lages, Jorge Augusto Vasconcelos Alves, Real-Time Control Of A Mobile Robot Using Linearized Model Predictive Control, IFAC Proceedings Volumes, Volume 39, Issue 16, 2006, https://doi.org/10.3182/20060912-3-DE-2911.00166.