Journal of Research High School

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

Gradient-Based Neural Model Prediction Control for Continuous-Time Control of Quadrotors

Joonyong Choi[1] *

[1]Korea International School Jeju Campus, Republic of Korea
*Corresponding Author: jychoi24@kis.ac

Advisor: Jae Myong Ryu, jmryu@kis.ac

Received June 9, 2023; Revised September 24, 2023; Accepted, October 5, 2023

Abstract

This research proposes a novel deep learning-based control model for quadrotors utilizing gradient-based neural model prediction. The primary objective of this research is to enhance the accuracy and efficiency of quadrotor motion prediction through optimization techniques related to deep neural networks. The proposed model in this research outperforms conventional approaches for quadrotor motion control regarding the accuracy of quadrotor trajectory prediction, which would eventually contribute to precise autonomous control of quadrotors prospected to be widely implemented in drone-related businesses and industries. The model especially becomes more accurate as it adjusts the system to reduce the deviations calculated from the cost function, which occurs during the process of quadrotor motion prediction. Using these techniques, the quadrotor's trajectory and routes were accurately measured through multiple trials, driving the quadrotor to the predetermined destinations accurately with high precisions as well. Therefore, the developed model successfully fulfilled this research's primary objective of developing a deep neural network-based deep learning model to accurately predict quadrotor trajectory for autonomous control. The results gained in this research are highly impactful in both academia and industries as they provide direct and significant benefits on the quadrotor motion development endeavors, aiding control system development and broadening the range for applications of quadrotors across various sectors of industries.

Keywords: Cost function, Deep neural networks, Multi-agent reinforcement learning, Neural model prediction, Proportional integral derivative control

1. Introduction

The emerging advancements in the drone industry holds promising potential as a cornerstone of modern technological and economic landscape. Within the modern landscape, quadrotors are considered one of the most prominent drone models due to the symmetrical characteristics represented by their arranged rotors. The primary objective of this research is to develop a continuous-time neural model prediction control for quadrotors by implementing deep neural networks within the model training process. There are several methods used to build model prediction control models. The first method is the proportional integral derivative control, which contains factors of quick response time, removing offsets, and anticipating the environment. The second method is multi-agent reinforcement learning, which trains multiple agents separately by providing respective rewards and punishments for each agent based on their performances. Finally, the third method, which is the proposed novel approach in this research, is to utilize deep neural networks. By proposing this novel approach for quadrotor motion prediction and control, this research aims to significantly reduce errors in quadrotor motion prediction, thus allowing the reduction of control errors and elevating the accuracy and safety standards for autonomous quadrotor operations. These advancements would be pivotal in both academia and a myriad of applications within industries such as delivery, agriculture, videography, and many more.

A distinct gap is observed in current quadrotor control models, in which a lack of precise trajectory prediction diminishes the reliability and possibility for broader application of quadrotors. Addressing this gap, the proposed model in this research endeavors to significantly enhance quadrotor trajectory prediction accuracy by leveraging deep learning algorithms in the training process. It is hypothesized that a notable reduction in costs, or deviations, in the quadrotor's motion prediction can be achieved by optimizing the cost function within this model.

Anticipated findings of this research could provide a model that notably improves quadrotor control accuracy, which is indispensable for industries aiming to integrate drones in the most pivotal and critical sectors. The validation of the model's efficacy is demonstrated through rigorous sets of repeated trials, with a keen focus on reducing the cost function, which represents the deviations in the quadrotor's motion prediction.

Furthermore, the research also clearly exhibits the methodologies employed to help the foundational understanding for the readers, including the deep neural networks and the optimization techniques. In the subsequent sections, a detailed exploration of the model's development, experimental setup, and findings are laid out. Eventually, this research demonstrates that the proposed model could potentially propel the drone industry forward and provides anticipations for further research and development in this domain.

2. Related Works

2.1 Model Predictive Control (MPC)

Model Predictive Control (MPC), which is a set of advanced control methods that predicts future behavior of the system, is deeply examined and tested in several research. This review summarizes and provides and comprehensive overview on the latest developments, applications, and strategies to handle computational burden of the Model Predictive Control technique. For the automation of systems, feedback controllers can be divided into three categories: classical, bang-bang, or state controllers. All these controllers use a system model to predict future behaviors and anticipate the deviations of that prediction as well. The PID controller is the best-known controller with significance even in its industrial applications. What makes the MPC method valuable is that it anticipates the behavior of the system and considers hard constraints. Furthermore, MPC relies on models, which allows it to make use of a long range of knowledge and leaves the trivial formulation of an explicit control law. Today, the development of MPC theory is pulled forward by its variety of applications in areas such as the manufacturing technology (Schwenzer, et a l., 2021). Nevertheless, the classical MPC methodologies for devising machine learning or deep learning models tend to be computationally intensive, generating a high possibility of lack of ability to manage non-linear dynamics effectively. The gradient-based deep neural network-based model predictive control proposed in this research aspires to address this deficiency.

2.2 Quadrotor Control Planning

This research identifies the challenges of planning quadrotor trajectories in indoor environments. The research proposes a way to extend the existing work on polynomial trajectory generation through presenting a method of jointly optimizing polynomial path segments in an unconstrained quadratic program that is stable for high-order polynomials and large numbers of segments. Furthermore, this research also suggests a technique for an automation of selecting the allocated time for each segment, which allows the quadrotor to speed along the path. Applying polynomial trajectories eliminates the need for computationally intensive sampling and simulation when handling motion. The proposed approach creates high-quality trajectories significantly faster than purely sample-based optimal planning methods. The performance of the algorithm of efficiently generated trajectories is also provided (Richter, et al., 2013). However, despite these advancements, producing optimal trajectories for quadrotors based on prediction in dynamically changing environments remains a challenging task, which is an issue that this research aims to address with the novel deep-learning based quadrotor trajectory prediction model for autonomous control.

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

2.3 Learning-Based Motion Control

Learning-based methods have shown favorable performance for accelerating motion planning, but one shortcoming is that they were mostly created for the setting of static environments. For a more challenging planning of dynamic environments, such as multi-arm assembly tasks or human-robot connection, motion planners should consider the trajectories of the dynamic obstacles and reason the temporal-spatial interactions in a very large extent. This research proposes GNN-based approach that uses temporal encoding and imitation learning with data aggregation for learning embeddings and the edge prioritization policies. The experiments performed show that proposed methods significantly accelerate online planning. Also, the models reduce costly collision, and therefore accelerate planning by up to 95% while achieving high success rates on challenging cases at the same time (Zhang, et al., 2022). Nonetheless, conventional learning-based approaches often lack the real-time adaptive capabilities essential for safe and efficient quadrotor operation in complex and dynamic environments. They are often deficient on their ability to make quick real-time predictions as well. As a result, this research endeavors to overcome this deficiency of conventional learning-based motion control through the integration of gradient-based neural model prediction and deep learning algorithms.

3. Theoretical Background

3.1 Proportional Derivative (PD) Control

PD control is a type of controller in which the output is printed proportionally to the error signal together with the derivative of the error signal. This controller provides the functions of both the proportional controller and the derivative controller.

The proportional controller is a feedback controller that minimizes the fluctuation in the process variable, but it often forms deviation from the set point. The advantage of proportional controllers is that it provides a faster response than most controllers. The P-only controller could respond even faster as the system has more complicated algorithms. However, the deviation, known as offset, from the set point increases, which could eventually result in an error.

The derivative controller is a form of forward control. The D-control analyzes the change of error and predicts process conditions. The advantage of a D-controller is to resist change, which is especially important when facing oscillations (Brogan, 1990).

$$\begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} = \begin{bmatrix}-I_{xx}\left(K_d\dot{\phi} + K_p\int_0^T \dot{\phi}\,dt\right)\\ -I_{yy}\left(K_d\dot{\theta} + K_p\int_0^T \dot{\theta}\,dt\right)\\ -I_{zz}\left(K_d\dot{\psi} + K_p\int_0^T \dot{\psi}\,dt\right)\end{bmatrix} \tag{1}$$

$$\tau_B = \begin{bmatrix}Lk(\gamma_1-\gamma_3)\\ Lk(\gamma_2-\gamma_4)\\ b(\gamma_1-\gamma_2+\gamma_3-\gamma_4)\end{bmatrix} = \begin{bmatrix}-I_{xx}\left(K_d\dot{\phi} + K_p\int_0^T \dot{\phi}\,dt\right)\\ -I_{yy}\left(K_d\dot{\theta} + K_p\int_0^T \dot{\theta}\,dt\right)\\ -I_{zz}\left(K_d\dot{\psi} + K_p\int_0^T \dot{\psi}\,dt\right)\end{bmatrix} \tag{2}$$

$$T = mg \tag{3}$$

$$T_{proj} = mg\cos(\theta)\cos(\phi) \tag{4}$$

$$T = \frac{mg}{\cos(\theta)\cos(\phi)} \tag{5}$$

$$T = \frac{mg}{\cos(\theta)\cos(\phi)} = k\sum\gamma_i \Rightarrow \sum\gamma_i = \frac{mg}{k\cos(\theta)\cos(\phi)} \tag{6}$$

$$\gamma_1 = \frac{mg}{4k\cos(\theta)\cos(\phi)} - \frac{2be_\phi I_{xx} + e_\psi I_{zz}kL}{4bkL} \tag{7}$$

$$\gamma_2 = \frac{mg}{4k\cos(\theta)\cos(\phi)} + \frac{e_\psi I_{zz}}{4b} - \frac{e_\theta I_{yy}}{2kL} \tag{8}$$

$$\gamma_3 = \frac{mg}{4k\cos(\theta)\cos(\phi)} - \frac{-2be_\phi I_{xx} + e_\psi I_{zz}kL}{4bkL} \tag{9}$$

$$\gamma_4 = \frac{mg}{4k\cos(\theta)\cos(\phi)} + \frac{e_\psi I_{zz}}{4b} + \frac{e_\theta I_{yy}}{2kL} \tag{10}$$

3.2 Proportional Integral Derivative (PID) Control

PID control is a type of controller that is a combination of all three types of control methods: proportional, integral, and derivative control. So, it contains integral control in addition to the proportional control and derivative control priorly introduced.

The integral control is also a type of feedback controller which is used to remove deviations. The controller makes the system return to its steady state. However, the I-only controllers have much slower response speeds compared to the P-only controllers because they are more dependent on the parameters.

The PID control uses the P-only control for a quick response time and removes the offset created by the P-only control using the I-only control. Furthermore, the D-control increases the controller's response because it anticipates the conditions. The PID controller is one of the best controllers which is highly accurate and stable (Brogan, 1990).

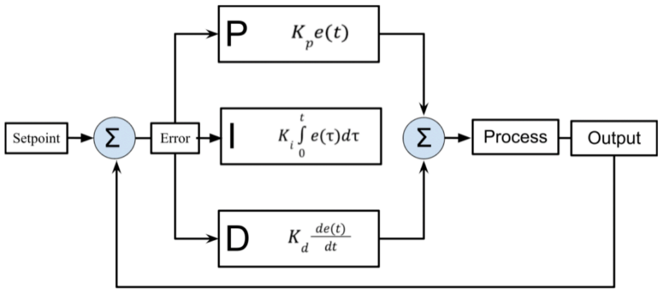

Figure 1. The Diagram of the PID Control (Mehta, Nikunj., et al., 2017)

$$e_\phi = K_d\dot{\phi} + K_p \int_0^T \dot{\phi}\, dt + K_i \int_0^T \dot{\phi}\, dt\, dt \qquad (11)$$

$$e_\theta = K_d\dot{\theta} + K_p \int_0^T \dot{\theta}\, dt + K_i \int_0^T \dot{\theta}\, dt\, dt \qquad (12)$$

$$e_\psi = K_d\dot{\psi} + K_p \int_0^T \dot{\psi}\, dt + K_i \int_0^T \dot{\psi}\, dt\, dt \qquad (13)$$

$$F_i = k_F \cdot P_i^2 \qquad (14)$$

$$T = \sum_{i=0}^{3} (-1)^{i+1}\, k_T \cdot P_i^2 \qquad (15)$$

$$\ddot{x} = \left(R \cdot \left[0, 0, k_F \sum_{i=0}^{3} P_i^2\right] - [0,0,mg]\right) m^{-1} \qquad (16)$$

$$\dot{\psi} = J^{-1}\left(\varkappa \times (L, k_F, k_T, [P_0^2, P_1^2, P_2^2, P_3^2]) - \psi \times (J\psi)\right) \qquad (17)$$

3.3 Reinforcement Learning (RL)

Reinforcement Learning is a technique of training machine learning models in order to make a series of decisions. The agents of the models learn to achieve a goal in a certain environment. In reinforcement learning, the AI attempts several trials and experiences failures until it achieves the goal, which is in most cases solving a problem. To set the AI to do a certain task, the AI receives rewards or penalties depending for its actions and achievements. Its goal is to maximize the total reward received. Reinforcement learning has been widely applied in the robotics fields for decades, and it had confined to higher-level decisions. However, recent research proves that reinforcement learning could also serve as a basis for serving aerial vehicles, such as quadrotors. A model-based reinforcement learning could be used to train a locally weighted linear regression policy, which could lead to a more dynamic motion of the quadrotor (Hwangbo, et al., 2017).

$$E\left[\sum_{t\geq 0} \gamma^t R(s_t, a_t, s_{t+1}) | a_t \sim \pi(\cdot\, | s_t), s_0\right] \qquad (18)$$

$$\varrho_\pi(s,a) = E\left[\sum_{t\geq 0} \gamma^t R(s_t, a_t, s_{t+1}) | a_t \sim \pi(\cdot\, | s_t), a_0 = a\right] \qquad (19)$$

$$V_\pi = E\left[\sum_{t\geq 0} \gamma^t R(s_t, a_t, s_{t+1})\, | a_t \sim \pi(\cdot\, | s_t), s_0 = s\right] \qquad (20)$$

3.4 Multi-Agent Reinforcement Learning (MARL)

Multi-agent reinforcement learning is a type of reinforcement learning, which is a field of training machine learning models to make a series of decisions. For many years, reinforcement learning was enacted through single agents. However, since there are many situations where agents should make independent decisions, it is better to apply multi-agent reinforcement learning in most cases. The advantage of multi-agent reinforcement learning is that it allows us to explore all different alignments appearing when multiple agents share an environment and know how those alignments affect each agent. Each agent is motivated by its own rewards, similar to the single-agent reinforcement learning. However, compared to the single-agent reinforcement learning, multi-agent reinforcement learning is more complex as the technique combines the pursuit of finding ideal algorithms that maximizes awards along with more sociological concepts. Since there are multiple agents, machine learning might not be effective as a whole since each

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

agent works for its own rewards. Therefore, it is also important to evaluate the social metrics of the agents such as cooperation, reciprocity, equity, and social influence (Panerati, et al., 2021; Zhang, et al., 2021).

3.5 Neural Differential Equations

Neural Differential Equations are continuous depth generalizations on standard layer to layer propagation. They use the neural network to parameterize the vector field. The purpose of using neural differential equations is to make the computation graph continuous so that it is actually differentiable. The residue network is formulated as $y_{j+1} = y_j + f_\theta(j, y_j)$, where $f_\theta(\mathrm{j},\cdot)$ is the j-th residual block. The ordinary differential equation was $\frac{dy}{dt}(t) = f_\theta(\mathrm{t}, \mathrm{y}(\mathrm{t}))$.

By discretizing this via the Euler method, $\frac{y(t_{j+1})-y(t_j)}{\Delta t} \approx \frac{dy}{dt}(t_j) = f_\theta(t_j, y(t_j))$ is obtained, which shows that neural differential equations are the continuous limit of residual networks. There is also the physics-informed neural network (PINN), which is effective for the AI to recognize physical conditions of the environment similar to that of the human intuition process. However, when working with a model, there is a structural problem called the 'inductive bias.' There is always some gap between the theoretical prediction and the actual values. As a result, the high-capacity function approximation is used as a solution to close the gap between theory and actual conditions.

There are certain ways to train the universal differential equations. One option is to use the parameters to initialize and only train the network parameters. Another option is to regularize the neural network so that it only fits to the residual between the theoretical model and actually observed data values (Kidger, 2021; Sangalli, et al., 2022).

3.6 Model Predictive Control (MPC)

Model Predictive Control is an advanced method of process control used to make prediction about future outputs in a set of constraints. It is a widely used concept in process industries such as petroleum and chemistry industries. The model predictive controllers are dependent on dynamic models. The primary advantage of model predictive control is that it allows the optimization of current timeslots, while considering future timeslots for prediction. As a result, this makes the model predictive control to anticipate future events and automatically control the object's motion. This method is a widespread digital control model used in many fields in modern days.

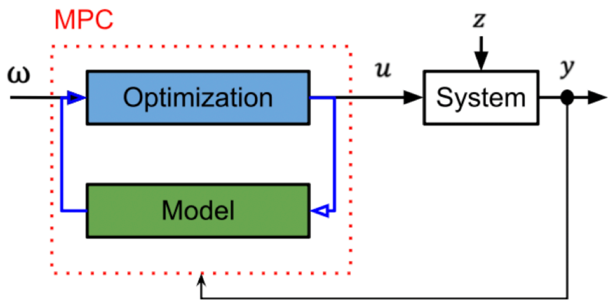

Figure 2. The Diagram of the MPC Process (Schwenzer, M., et al., 2021)

3.7 Gradient Descent

Gradient descent is an optimization algorithm used to train machine learning/deep learning models and neural networks in the manner of minimizing possible errors. Gradient descent method is important because it enables the model to be trained efficiently and effectively to achieve the goal. The gradient descent algorithm acts similarly to a commonly known quadratic function. The gradient descent method is used based on the cost function of a model, which is a function that measures the deviation, or error, between the actual output and the predicted output. From the starting point of the function, the derivative (slope) of the graph is determined to measure the steepness of the graph. Moreover, the slope would inform the parameters, possibly weights and bias, of the changes to be applied. Then, this process is repeated until it finds the lowest point on the cost function curve, the point of convergence, which would have a derivative nearly zero and the lowest cost. For the gradient descent to work, it requires two data points, direction and learning rate, which determine the partial derivative that would contribute to finding the minimum, which would

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

be the point of convergence. Learning rate is the step size taken in the process to find the minimum. High learning rate means the step size is large, which indicates a possibility of more error. On the other hand, low learning rate means the step size is small, which means it would have more precision but less efficiency since it would require more computation. There are primarily three types of gradient-descent. The first one is the batch gradient descent. This method takes the sum of the error of each training set, and it is called a training epoch. Batch gradient descent provides computation efficiency, but in exchange it takes long processing time. Another type of gradient descent is stochastic gradient descent. This method runs a training epoch for each sample and updates parameters once at a time. It has faster processing speed compared to the batch gradient descent, but it can have losses in computation efficiency. Finally, the mini-batch gradient descent method combines the concept of batch and stochastic gradient descent. It divides the dataset into small batches and updates parameters of each of those batches, creating a balance between processing speed and computation efficiency.

As an example of a gradient descent model, the cost function of the training set is as the following equation (21), where L is the loss function per example, $f(x;\theta)$ is the predicted output for input x, and $\hat{p}_{data}$ is the empirical distribution. The optimization of the model is represented as equation (22), where m is the number of training examples.

$$J(\theta) = E_{(x,y)\sim\hat{p}_{data}} L\big((f(x;\theta), y\big) \quad (21)$$

$$E_{x,y\sim\hat{p}\,data(x,y)}[L(f(x;\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L\big(f\big(x^{(i)};\theta\big), y^{(i)}\big) \quad (22)$$

The maximum likelihood estimation problems decompose into a sum over each example.

$$\theta_{ML} = \arg max \sum_{i=1}^{m} log p_{model}\big(x^{(i)}, y^{(i)};\theta\big) \quad (23)$$

Maximizing the sum is equal to maximizing the expectation over empirical distribution defined by the training set, which is the following in equation (24). The most commonly used property is the gradient in equation (25).

$$J(\theta) = E_{x,y\sim\hat{p}_{data}} log p_{model}(x, y;\theta) \quad (24)$$

$$\nabla_\theta J(\theta) = E_{x,y\sim\hat{p}_{data}} \nabla_\theta \log p_{model}\,(x, y;\theta) \quad (25)$$

The stochastic gradient descent, an optimization algorithm for an objective function to find the model parameters that correspond to the best fit between predicted and actual values, minimizes the generalization error faster than other gradient descent algorithms. Also, it is especially suitable for large-scale datasets. The generalization error could be written as the following equation (26). The exact equation for the gradient is represented in equation (27).

$$J^{*}(\theta) = \sum_x \sum_y p_{data}(x, y) L(f(x;\theta), y) \quad (26)$$

$$g = \nabla_\theta J^{*}(\theta) = \sum_x \sum_y p_{data}(x, y)\, \nabla_\theta L(f(x;\theta), y) \quad (27)$$

For the experiment of direct optimal control of inverted pendulum with a torsional spring and the gradient-based control experiment for quadrotors, which would be articulated in section 4, both uses the *Adam* optimizer.

Adam optimizer is the extended version of stochastic gradient descent which is expected to be applied in various deep learning fields such as computer vision and natural language processing in the future. This algorithm can serve as an alternative option of the stochastic gradient descent. *Adam* uses estimations of the first and second moments of the gradient in order to adapt the learning rate for each weight of the neural network. So, since the *Adam* optimizer has better generalizing performance, it was adopted in both experiments.

The main component of the *Adam* optimizer is the Momentum algorithm. The momentum algorithm considers the exponentially weight average and expedites the gradient descent process. The momentum algorithm is divided into two parts. The first part calculates the displacement, or the position change, while the second part updates the prior position. The change in position is given as the equation (28), where α is the stepsize, or in other words, the hyperparameter which controls the movement in search space, also called the learning rate.

$$update = \alpha \times m_t \quad (28)$$

The new position of the or weights at time t is given by the equation (29). Furthermore, $f'(x)$ is the derivative or aggregate of gradients at time t. The equation (30) below shows the aggregate gradients at time t, represented as m_t and $m_t - 1$. The momentum algorithm dampens down the change of the gradient and the step size with each new point of the search space (Goodfellow, et al., 2016).

$$w_t + 1 = w_t - update \quad (29)$$

$$m_t = \beta_1 m_t + (1-\beta_1)\left(\frac{\delta L}{\delta w_t}\right) \quad (30)$$

4 Materials and Methods

This section shows experimental data, measurements, and observations. No explanations or interpretations are expected in this section and those information needs to be addressed in the discussion section. All tables, figures, and equations should be located in the proper positions and all descriptive explanation needs to be referred from the context in the body.

The experiment described in this section was conducted by an M1 MacBook Pro Max 2021 laptop, a 14-inch version, which has 10 cores of Central Processing Unit (CPU) and 32 cores Graphics Processing Unit (GPU). Regarding the software, this research utilized the Visual Studio Code Integrated Development Environment (IDE) and specifically employed the Python3 programming language. The experiment was only based on simulation settings, and results were measured by how closely the quadrotor reaches to the predetermined destination point in a 3-dimensional coordinate space. After the experiment trials were completed, the range of the predicted position within the predetermined destination and the error bound derived from that range were obtained to quantify the accuracy of the model's prediction.

4.1 Direct Optimal Control of an Inverted Pendulum with a Torsional Spring

Using the theories listed in prior sections, it is also possible to control an inverted pendulum with a torsional spring and stabilize it in the upright position. The equation (31), which is in Hamiltonian form represents the inverted pendulum with a torsional spring.

$$\begin{bmatrix}\dot{q}\\ \dot{p}\end{bmatrix} = \begin{bmatrix}0 & 1/m\\ -k & -\beta/m\end{bmatrix}\begin{bmatrix}q\\ p\end{bmatrix} - \begin{bmatrix}0\\ mgl\sin(q)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}u \tag{31}$$

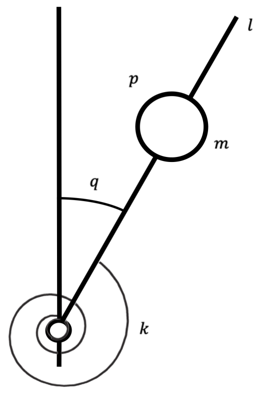

Figure 3. Diagram of Inverted Pendulum with Torsional Spring (Pierre, Coullet., Gilli, J.M., & Rousseaux, Germain., 2009)

First, to set up the experiment for the inverted pendulum, *matplotlib* and *torch* should be imported to the system. Also, the device should be set up according to the configuration. Then, to control the pendulum, the proper integral cost function must be derived so that the loss to be minimized through training could be accurately represented. The cost function is defined as the following equations (32) and (33), where x is the state and u_θ is the controller. x^* and u^* are the desired position, and the matrices P, Q, R represent the weights for altering the performance (Markus, et al., 1976).

$$\min J = \left(x(t_f) - x^*\right)^T \boldsymbol{P}\left(x(t_f) - x^*\right) \tag{32}$$

$$\int_{t_0}^{t_f} [(x(t) - x^*)^T \boldsymbol{Q}(x(t) - x^*) + (u_\theta(t) - u^*)^T \boldsymbol{R}(u_\theta(t) - u^*)]\, dt \tag{33}$$

After setting the proper integral cost function, the initial conditions such as the time span, step size, and initial and final time should be declared as variables and assigned input values. For this inverted pendulum experiment, a box-constrained controller would be used since there should be a given limited control input for the parameters p and q, which respectively represent the angular position and angular speed of the inverted pendulum. Finally, the optimization loop would be run. Specifically, the stochastic gradient descent method would be used along with *Adam* to optimize the parameters.

4.2 Gradient-Based Control for Quadrotors

$$min \sum_{k=0}^{T-1} J\ (x_k, u_k) \tag{34}$$

$$subject\ to\ \dot{x}(t) = f\ \big(t, x(t), u(t)\big),\ \ t \in ℸ \tag{35}$$

$$x(0) = x_0 \tag{36}$$

This research uses Model Predictive Control (MPC) in order to enable the quadrotor to predict the future behavior of its system to predict the trajectory. The formulation for this Model Predictive Control (MPC) could be written as the following equations (34), (35), and (36), where J is the

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

cost function and $T \in$ ℸ is the model predictive control over which predicted future trajectories are optimized.

The quadrotor model is an appropriately modified version of explicit dynamics for batched training in *PyTorch*. The following equations (37) and (38) are for the accelerations of the quadrotor model, where $\boldsymbol{x} = [x, y, z]$ represents the quadrotor's 2-dimensional position and $\boldsymbol{\psi} = [\phi, \theta, \psi]$ represents the quadrotor's angular position. $\boldsymbol{R}$ and $\boldsymbol{J}$ are the quadrotor's rotational and inertial matrices respectively, and the function τ is a function that calculates the torque created by the motor generated with the speed of w_i, arm length of l, mass m, acceleration due to gravity g, and quadrotor's physical properties K_F, K_T. The initial conditions and some constants such as g are determined before the model is trained.

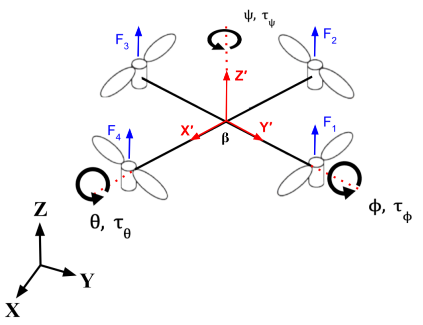

Figure 4. Diagram of the Quadrotor Dynamics

$$\ddot{x} = \left(\boldsymbol{R}\cdot\left[0,0,k_F\sum_{i=0}^{3}\omega_i^2\right] - [0,0,mg]\right)m^{-1} \tag{37}$$

$$\ddot{\psi} = J^{-1}\left(\tau(l, k_F, k_T, [\omega_0^2, \omega_1^2, \omega_2^2, \omega_3^2]) - \dot{\psi}\times(\boldsymbol{J}\dot{\psi})\right) \tag{38}$$

Then, the cost function is programmed. First, I set a goal that the quadrotor should be reached. The goal given to the quadrotor was to reach the position of $[x, y, z] = [3,3,3]$ under the circumstance that the quadrotor is initially at rest and has no rotation and velocities at the end. The cost function is programmed so that it could measure the deviations between the actual quadrotor's behavior and the predictions made by the Model Prediction Control (MPC) and contribute into making a more accurate prediction.

The initial conditions should be programmed after setting the cost function. Since the motion of the quadrotor initially begins at rest, all variables of the quadrotor motion and position should be set to 0. This includes the $[x, y, z]$ coordinates of the quadrotor's 3-dimensional position and the $[\phi, \theta, \psi]$ coordinates of the quadrotor's angular position.

The controller that would be used is a simple controller based on the Neural Network with bounded control inputs of the quadrotor. The input dimension, output dimension, hidden dimension, and the initialization of the last layer of the Neural Network to zero are included in these bounds.

The next step is the MPC initialization. First, the time step, noted as dt, is set to 0.02, and the final time would be 2 seconds. Then, a horizontal line would be programmed. This means that the quadrotor would be tested for 2 seconds with 0.02 seconds of intervals. Also, the learning rate would be set up, which means that the training would be repeated until the error is lower than the value that is set up. After that, the parameters would be optimized by the optimizer known as *Adam*. Finally, the MPC receives the data of every component and sends it to the device. At the end, the MPC finally goes through a simulation which produces the trajectories through repeated multiple trials.

5 Results

5.1 Inverted Pendulum with a Torsional Spring

As the optimization loop is run, the training loop is run, and the losses are plotted as a result with respect to the progress of epochs. In the result, represented in Figure 5, the diminishing trend of losses as the epochs are progressed are shown.

The final plot results of the pendulum trajectories are derived after the losses. The graphs for each of the variables p and q, respectively representing the position and angle of the pendulum, are obtained after the experiment. The final plot results for those variables are as the following.

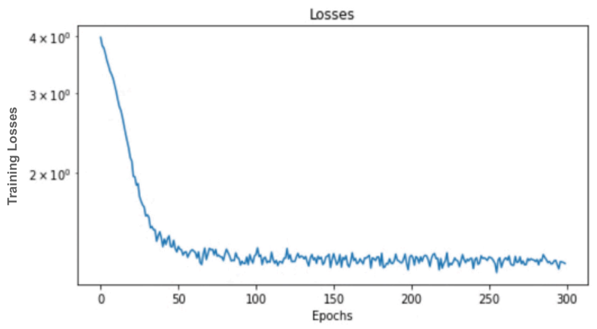

Figure 5. The Subplot of Losses with respect to the Epochs

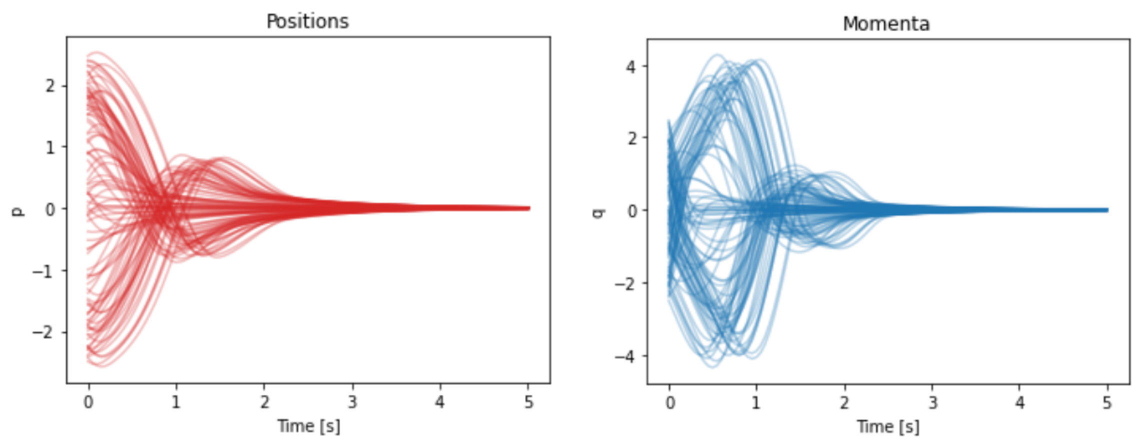

Figure 6. Plot Results of Pendulum Trajectories of each Variables

Ultimately, considering Figure 6, the overall phase space of the pendulum could be solely represented by p and q, as shown in Figure 7.

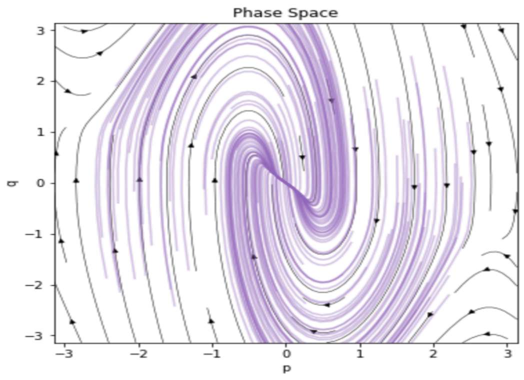

Figure 7. Plot Result of Pendulum for p and q

5.2 Gradient-Based Control for Quadrotors

During the simulation, the four propellers of the quadrotors are controlled. The change of the control of each propeller, in RPM, is shown as the following in Figure 8.

Due to the change of control inputs in the time interval, the trajectories for each of the twelve variables (linear positions x, y, z; angular positions ϕ, θ, ψ; linear velocities $\dot{x}, \dot{y}, \dot{z}$; angular velocities $\dot{\phi}, \dot{\theta}, \dot{\psi}$) were derived as the following in Figure 9.

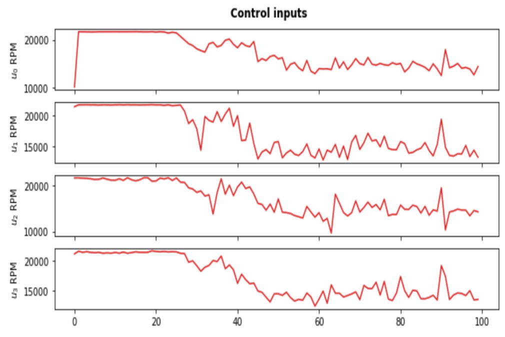

Figure 8. Change in Control Inputs of Each Propeller

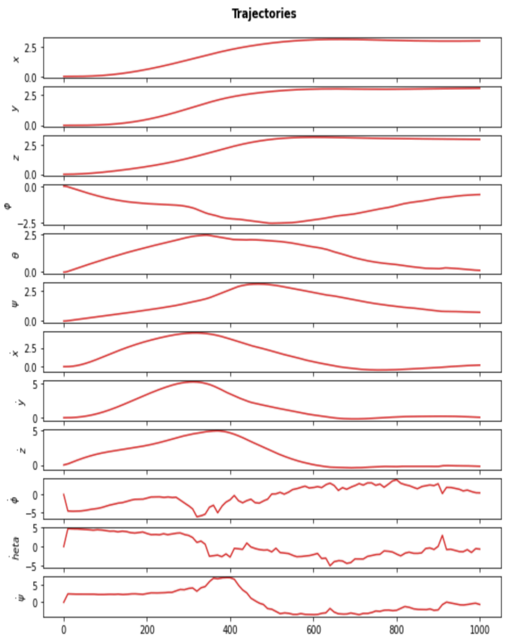

Figure 9. Trajectories of Twelve Variables of Quadrotor Motion after MPC simulation

Finally, as these trajectories for all twelve variables were derived, the complete trajectory of the quadrotor in the 3-dimensional space was obtained considering these changes in the variables in given time. The final trajectory of the quadrotor was derived as the following in Figure 10.

Vol. 2023 (2) xxx – xxx
ISSN 2688-3651 [online]

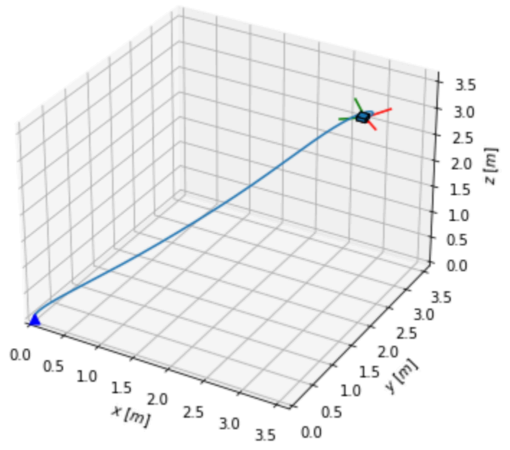

Figure 10. The 3-D Trajectory of the Quadrotor

6. Discussion

The result of the experiment was successfully derived as demonstrated visually in Figure 10. Using methods of MPC, cost function, initial conditions, controller, MPC initialization, and simulation, the quadrotor reached about a final destination of approximately $[x, y, z] = [2.974, 3.015, 3.009]$ compared to the predetermined destination of $[x, y, z] = [3, 3, 3]$, which is both in the units of meters. As a result, the error bounds of the axes are approximately 0.026, 0.015, and 0.009 meters respectively for each x, y, z axis respectively. Calculating the percent error based on these error bounds in meters, the absolute percentage error for each x, y, z axis is approximately 0.872%, 0.498%, and 0.299%, respectively. As a result, the mean absolute percent error across all the dimensions is approximately 0.556%. Therefore, based on these statistics, the devised model provided a significantly accurate final position of the quadrotor utilizing the techniques, control inputs, and trajectories of the twelve variables that represent the quadrotor motion, which are represented in Figure 9.

The experiment was successful, as a gradient-based neural model prediction control for continuous time-control of quadrotors was created and verified through simulation. The supposed benefit of this model is that it improved the accuracy and efficiency of quadrotor motion prediction using optimization techniques related to deep neural networks. Also, the model has reduced deviations calculated from the cost function, promising more accurate quadrotor trajectory prediction essential for autonomous control. Furthermore, as the model implements deep neural networks, it is likely to be more effective in complex environments as well with reduced costs and control errors. Eventually, this would enable broader applications of quadrotors in academia and industries, contributing to the advancement of quadrotor implementation in various areas such as delivery, agriculture, videography, and many more.

The limitation of this research is that this model still experiences significant computational intensity because it uses MPC as one of its methods for training. Also, since the experiment was only done at a simulation-based setting, it might be difficult for immediate and direct implementation in real-world, dynamic environment settings. Although it is possible to use in those kinds of environments, it is likely to demonstrate diminished performance compared to the experimental results shown in this paper. Finally, the research needs some reinforcement in indoor setting environments in the real-world as well.

Some potential solutions to these limitations would be utilizing more advanced deep learning and optimization techniques in the training process to significantly enhance quadrotor trajectory prediction accuracy. Also, combining the MARL technique along with the deep neural network technique implemented in this research could also serve as a feasible solution to address the limitation. Furthermore, extending the existing work through joint optimizing polynomial path segments in an unconstrained quadratic program or utilizing imitation learning along with data aggregation could potentially aid quadrotor motion prediction in dynamic environments and help improve quadrotor control planning.

7. Conclusion

The reason why this development is important is because it creates an accurate automatic quadrotor control model using the techniques that enable the system to anticipate the motions of the quadrotor precisely. Eventually, the suggested quadrotor control model could be applied in many areas of research or industries in the future.

For further development, it would be good to increase the accuracy of the model through constant training of the model. Moreover, as mentioned priorly, the results derived from this research could be applied in many areas, such as the delivery industry and agriculture. As many companies are developing drone deliveries, the model for accurate prediction of quadrotor trajectories and destination would be the most critical technology required for them. Also, in terms of agriculture, people are attempting to use drones, especially quadrotors, to check water levels and farmland conditions as well as any risks in surrounding areas. The neural model prediction control for quadrotors would be significant during this as it would allow the users to know the accurate trajectories and the final location of their quadrotors. As suggested in these examples, the results of this research would be highly influential and contributive to the development of the drone industry and society.

References

Brogan, W. L. (1990). Modern Control Engineering Theory Third Edition.

Conti, R., Markus, L., & Olech, C. (1976). Control Theory and Topics in Functional Analysis.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Deep Learning.

Hwangbo, J., et al. (2017, July 17). *Control of a quadrotor with reinforcement learning*. arXiv.org.

Kidger, P. (2022, February 4). *On neural differential equations*. arXiv.org.

Mehta, N., et al. (2017). Design of HMI Based on PID Control of Temperature. International Journal of Engineering Research and Technology. V6. 10.17577/IJERTV6IS050074.

Panerati, J., et al. (2021, July 25). *Learning to fly -- a gym environment with pybullet physics for reinforcement learning of Multi-agent Quadcopter Control*. arXiv.org.

Pierre, C., Gilli, J. M., & Rousseaux, G. (2009). On the critical equilibrium of the spiral spring pendulum. Proceedings of The Royal Society A Mathematical Physical and Engineering Sciences. 466. 407-421. 10.1098/rspa.2009.0393.

Richter, C., Bry, A., & Roy, N. (2013). Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments.

Sangalli, S., et al. (2022, January 5). *Constrained Optimization to train neural networks on critical and under-represented classes*. arXiv.org.

Schwenzer, M., et al. (2021, August 11). *Review on Model predictive control: An engineering perspective - the International Journal of Advanced Manufacturing Technology*. SpringerLink.

Zhang, K., Yang, Z., & Başar, T. (2021, April 28). *Multi-Agent Reinforcement Learning: A selective overview of theories and algorithms*. arXiv.org.

Zhang, R., et al. (2022, December 6). *Learning-based motion planning in dynamic environments using GNNS and temporal encoding*. Advances in Neural Information Processing Systems.