Final Report: Deep Learning for Comparative Control of Quadrotor Unmanned Aerial Vehicles under Newton-Raphson Flow Regime

Evanns G. Morales-Cuadrado
Georgia Insitute of Technology
egm@gatech.edu

Mahya Qorbani
Georgia Insitute of Technology
mqorbani3@gatech.edu

Abstract

Many powerful tracking control architectures depend upon a state predictive element to consider future actuation needs. In particular, the recently demonstrated Newton-Raphson controller is a very powerful and computationally light-weight controller that requires a lookahead predictor of the vehicle's future state. In previous research, the predictive task has been carried out via model-based predictors. The novelty in this work is that we seek to accomplish this task with deep neural networks to compare and contrast the results with the aforementioned predictors from traditional control systems theory. Thus, we execute a comparative analysis of various deep learning prediction methods. We do this by comparing the tracking performance of a Quadrotor UAV on various trajectories while using the outlined deep learning-based predictors through their respective root-mean-squared errors between their paths and the reference trajectories the paths intend to track.

1. Introduction

Unmanned Aerial Vehicles (UAVs) have become increasingly popular in the control research literature in recent years due to the availability of powerful new quadrotor flight controllers like the Pixhawk Series [3], enabling fast and off-the-shelf development for researchers and hobbyists alike [1, 5, 7]. The Newton-Raphson flow controller [9] is an integrator-type controller for aggressive quadrotor trajectory tracking. In this method, a continuous-time variant of the Newton-Raphson method updates the control input via a tunable integration update speed with the goal of driving to zero the error between the predicted future state of the vehicle and the desired reference trajectory. The result is a computationally lightweight feedback control scheme that achieves remarkable asymptotic tracking performance.

However, the Newton-Raphson controller requires a state predictor that approximates the state of the system at some future horizon T given the system's current state and control input. This predictor should also be differentiable in that we can find the Jacobian matrix of derivatives from each of its predicted states to each of the control inputs. Because of the controller's robustness to some flaws in its predictors, our goal is to achieve fast, numerically lightweight, and accurate-enough prediction of quadrotor state in spite of 3D quadrotors' complex, underactuated, and highly coupled dynamics. We seek to achieve this predictive task with various neural network architectures as will be seen in the subsequent sections and compare results among these architectuers as well as against traditional predictors.

Previous works have accomplished this task using model-based predictors. In particular, predictors that take the complex nonlinear dynamics of quadrotors and integrate them through the time horizon T via forward Euler integration have been used successfully. In addition, the Newton-Raphson method has also been shown to be robust to significantly weaker predictors like those based on linearized models under many simplifying assumptions and simply plugging them into the state-transition equation. Moreover, both of these methods utilized a zero-order-hold of the control input which is yet another simplifying assumption on the predictor which the method has been shown to be robust to. Our controller's output is low-level control of body rates and thrust of the quadrotor which must be published at around 100Hz to maintain stability. Moreover, these calculations will have to be performed on a Raspberry Pi onboard the quadrotor, not on a traditional strong computer. Therefore, speed and computational efficiency are of the utmost importance for our predictors.

Our goal in this work is to utilize novel predictors using Feedforward, Recurrent, and LSTM neural networks to achieve this prediction task and achieve com-

petitive if not superior tracking performance to some traditional methods. We also aim to measure the computation time of these methods and measure the relative benefits of giving up accuracy for speed or vice-versa with the traditional and neural network-based methods. Ideally, we would find small neural-networks with accurate parameters that can perform their matrix computations faster than the nonlinear predictors forward-Euler integration. Moreover, the success of neural network prediction would complement greatly the simplicity of the Newton-Raphson controller because it would allow it to work without knowledge of the system dynamics and without the numerical intensity of the standard methods, making this a nontrivial novelty and achievement.

This performance will be qualitatively and visually explored and compared with graphs plotting the paths taken against the trajectories they were meant to follow, and quantitatively measured via the root-mean-squared error between the paths the quadrotors take and the reference trajectories given to them under the various prediction methods. These flights will take place in the realistic Gazebo simulator that utilizes the Dart physics engine developed at Georgia Tech, where in our previous work [2] we have shown that if a control method works in this simulator, it can be translated to work on hardware with relative ease.

1.1. Related Works

There exists a body of work developing the Newton-Raphson tracking controller technique in various forms and guaranteeing asymptotic error bounds under certain conditions [8]. The Newton-Raphson flow controller is essentially a variable-gain integral controller with error bounds generally based on the quality of a lookahead output predictor. It is useful for nonlinear systems while not requiring linearization or nonlinear inversions. While not as powerful as some of the aforementioned techniques, the Newton-Raphson flow approach is computationally very lightweight, fast, and carries agreeable convergence bounds. For certain systems it may even be globally convergent, despite the fact that the core Newton-Raphson iterative method is a local one. This work builds off of an author's previous work in [2], which was the first to successfully apply the Newton-Raphson flow controller to a real-world underactuated and open-loop unstable nonlinear plant, namely, the 6-DOF 3D quadrotor. The results of this work will soon be verified via hardware flight experiments in the laboratory alongside other results and submitted to a control systems journal.

2. Method: Newton-Raphson with Neural Network Prediction

The essence of the control systems problem is to drive the output of a dynamical system $y(t)$ to a desired reference trajectory $r(t)$ via a control input $u(t)$. While there are many methods to accomplish this for various systems, we have chosen our system to be the 3D quadrotor and our control method to be the Newton-Raphson controller for dynamical systems. This gives us a computationally-lightweight method that is described by the following simple control law:

$$\dot{u}(t) = \alpha \left(\frac{\partial \rho}{\partial u} \left(x(t), u(t) \right) \right)^{-1} \left(r(t+T) - \rho\big(x(t), u(t)\big) \right). \tag{1}$$

This control method in itself effectively has two inputs: the reference $r(t)$, and a differentiable prediction function $\rho(\cdot)$. Note that $\left(\frac{\partial \rho}{\partial u} \left(x(t), u(t) \right) \right)^{-1}$ is the inverse of the partial Jacobian of the predictor outputs with respect to the control inputs $u(t)$. Also note that $x(t)$ is the full 9-dimensional state vector of the quadrotor and $y(t) = C \cdot x(t)$, in other words the group of states in $x(t)$ whose values are relevant for trajectory tracking.

However, this predictor is not directly specified in the literature. In fact, the Newton-Raphson method has been shown to be so powerful that it can be paired with a predictor based on greatly simplified linearized dynamics of a quadrotor and perform very well nevertheless [2]. Thus, this is where we seek our goal of achieving novelty by attempting quadrotor flight with the Newton-Raphson method by utilizing deep-learning based prediction methods for the first time. As can be seen in Figure 1, we seek to implement the most complicated part of the controller by using deep neural networks for the prediction task, shown in the box in green. Because the control algorithm itself (1) is not too involved, this frees us to focus on the deep learning prediction task for this project without spending much time on the control sytems portion.

2.1. Deep Learning Applications

We seek to experiment with various types of deep learning architectures to compare these architectures with existing methods in the literature. To achieve this, we evaluated three distinct deep neural network architectures: the Feedforward Network, Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM) Network. Our choice of the Feedforward architecture may be obvious as it is the standard neural

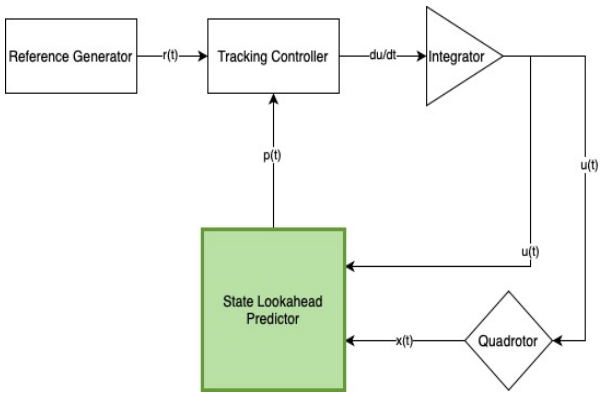

Figure 1. Control Diagram with Deep Neural Network Prediction Element in Green; We use neural networks to achieve prediction $p(t)$ of what the state vector will be in $T = 0.8$ seconds into the future given the current state $x(t)$ and the current input $u(t)$.

network architecture. However, the use of recurrent methods like RNN and LSTM is also natural, as by definition of the problem statement of predicting system states at T seconds into the future based on current information, we are feeding sequential state and input data and expecting the network to find the underlying pattern in these state sequences. This justifies our experimentation with these architectures.

These neural networks are given 13-dimensional input data which we will denote as $\mathcal{V}$. This data comes from two sources. Firstly, it has the 9-dimensional state vector $x(t)$ at time t made up of position, velocity, and Euler angles in all three dimensions. Secondly, it has the 4-dimensional input vector at time t, made up of angular velocities and net thrust. All together, concatenating them yields the 13-dimensional input vector $\mathcal{V}$. On the other hand, the output vector, denoted as $\mathcal{D}$ is represented as a 9-dimensional vector ($\mathcal{D} \in \mathbb{R}^9$). This is simply the state vector prediction at time $t+T$, and it accordingly has the same dimensions as $x(t)$.

For all of our training, we split the dataset in to 80% train dataset, and 20% test dataset. Moreover, we used a batch size of 128, 50 epochs of learning, and mean squared error as our criteria to evaluate the performance of these models. The detailed structure of every model is explained thoroughly in the next section.

2.1.1 Neural Network Models

For our feedforward neural network, we have designed a structure consisting of five layers in total. Each layer is composed of a linear transformation followed by a rectified linear unit (ReLU) activation function. This configuration is achieved by repeating a block of linear and ReLU layers four times followed by a final linear layer. We conducted experiments with various optimizers, including SGD, Adam, and AdamW, and fine-tuned the learning rate to identify the optimal value, which was determined to be 1×10^{-3}. Table 1 presents the performance of this Feedforward neural network model after 50 epochs under various optimizers.

Our RNN model is composed of 8 layers, and the hidden size for each layer is set to 4. In the process of configuring this model, we conducted experiments with multiple optimizers, including SGD, Adam, and AdamW. Through iterative tuning, we identified the optimal learning rate, which was determined to be 0.001. It's worth noting that we also undertook data preprocessing, reshaping the input data to ensure compatibility with the RNN architecture. The performance results of the RNN model trained with different optimizers, following 50 epochs, are summarized in Table 1.

The LSTM model employed in our experiment consists of two layers, each with a hidden size of 64. To optimize its performance, we carefully tuned two critical hyperparameters: the learning rate and the weight decay. After conducting a series of experiments, we determined the optimal values for these parameters to be 0.001 and $1e^{-4}$, respectively. We subjected the LSTM model to the same suite of optimizers we applied to the Feedforward and RNN models. The results of our LSTM model, obtained after training with various optimizers over the course of 50 epochs, are presented in Table 1.

3. Neural Network Development

In this section, we provide an overview of the data-collection strategies and what we learned from them. In addition, we showcase the training results obtained from our three models and conduct a comparative analysis. A comparative survey of our results organized by architecture and optimizer can be seen in Table 1. Moreover, Figure 2 demontrates the accuracy of each method as the epochs progress.

3.1. Data Collection and Training

3.1.1 Dataset #1

Our first method of collecting data was to collect flight data from the Iris quadrotor in the Gazebo simulator, and to use this to train and validate the three different neural network architectures. We achieved this data collection by taking each data sample from a buffer that at each time step of our algorithm saves state and input data under the label of time t, and waits for T seconds to elapse and saves the data for each t and $t+T$

pair together to yield our complete input and output data for training and testing. This means that each data sample has $9 + 4 + 9 = 22$ elements altogether. We collected data from our standard flight trajectories that we test controllers on, namely horizontal and vertical circles, as well as various lemniscate trajectories. This initial dataset 16,244 timesteps over the various different trajectories, resulting in a data matrix with dimensions of $16,244 \times 22$.

As can be seen in figure 2 and table 1 below, the training went very well. Unfortunately, when it came to test predictors based on this dataset in the gazebo simulator, it did not perform well. Not only did it not perform well, but the results were so inaccurate that when our code called a function to translate the required upward thrust force into a throttle vector from $[-1, 1]$ which requires us to take a square root, it gave us a math domain error. This became a major challenge in our project to overcome. Clearly, the dataset was not large enough or varied enough in the flight trajectories that it covered. We also suspected that because we are only predicting the output at $T = 0.8$ seconds into the future, that with such limited data the neural networks would simply give us outputs that are very similar to the inputs without learning the underlying system dynamics and not be penalized very harshly via the loss function.

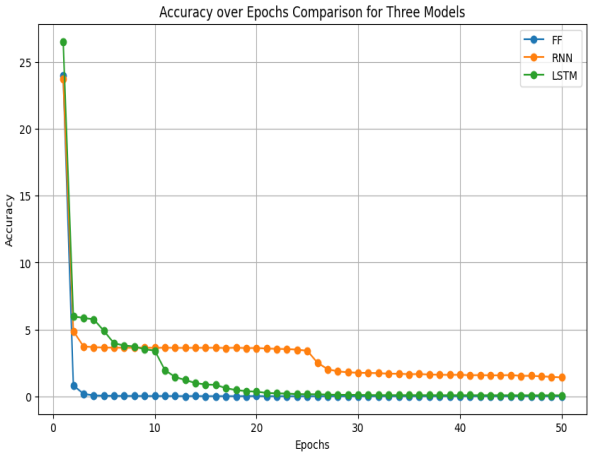

Figure 2. Loss Comparison for Three Models with AdamW Optimizer on Dataset #1

Although the first dataset was not successful in its application, we learned some lessons from the process of collecting data and training the three architectures. Firstly, as as seen in Table 1, the feedforward network model, optimized with AdamW, produced the most promising performance in terms of error.

We believed that the LSTM would easily be the best-performing architecture because of its capacity to deal with sequential data, then the feed-forward, and lastly the RNN due to its natural limitations regarding vanishing and exploding gradients. However, at first glance through this initial dataset and training success, we learned that the feedforward network emerged another leading candidate in our comparative evaluation. As illustrated in Figure 2, it is evident that the feedforward model exhibits faster convergence compared to the other two models, although their final accuracy values were comparable.

Table 1. Preliminary Results of Each Architecture with Various Optimizers for state lookahead predictor

Model	Optimizer	Train Error	Test Error
FF	SGD	0.0638	7.997
FF	Adam	0.0004	0.0462
FF	AdamW	0.0001	0.0161
RNN	SGD	0.0454	5.823
RNN	Adam	0.0320	4.208
RNN	AdamW	0.0411	5.308
LSTM	SGD	0.06	7.931
LSTM	Adam	0.044	0.5704
LSTM	AdamW	0.0005	0.042

3.1.2 Dataset #2

In order to fix the implementation issues from the first dataset and use the lessons we learned from the process of training the networks, we sought to generate a larger dataset that incorporated many more input-output relationships that demonstrated the dynamics of the 3D quadrotor for the neural networks. To do this, we simulated the quadrotor numerically via its nonlinear dynamics and used the random module from numpy to generate random values for its 9-dimensional state and 4-dimensional control input vector in order to generate accurate input-output relationships mathematically and have a wide range of generalizable relationships via the random inputs.

We achieved this numerical simulation by utilizing the quadrotor system equations and embedding them into our 9-dimensional model's states and integrating them forward in time:

$$\dot{V}_x = -(u_\tau/m)\big(\mathrm{s}(\phi)\mathrm{s}(\psi) + \mathrm{c}(\phi)\mathrm{c}(\psi)\mathrm{s}(\phi)\big) \quad (2)$$

$$\dot{V}_y = -(u_\tau/m)\big(\mathrm{c}(\phi)\mathrm{s}(\psi)\mathrm{s}(\theta) - \mathrm{c}(\psi)\mathrm{s}(\phi)\big) \quad (3)$$

$$\dot{V}_z = (u_\tau/m)\big(\mathrm{c}(\phi)\mathrm{c}(\theta)\big) - g. \quad (4)$$

We created a function that took these equations and random values and generated $50,000$ vectors with the 13-dimensional inputs alongside the 4-dimensional prediction output that we care to track as the 'labels'.

Whereas our initial dataset included all 9 predicted states, we simplified our architecture by only saving the four predicted states that are relevant for the Newton-Raphson controller. Although it took around 40 minutes to generate these random inputs and run them through the complex quadrotor equations, ultimately this dataset proved to not only train well, but to generalize well in our simulation experiments shown in section 4.

In training with this dataset, it was observed that the recursive neural network took approximately 80 iterations to achieve low loss values, a notable increase from the initial 50 iterations expected. This early indication of performance challenges is further discussed in the experiments section.

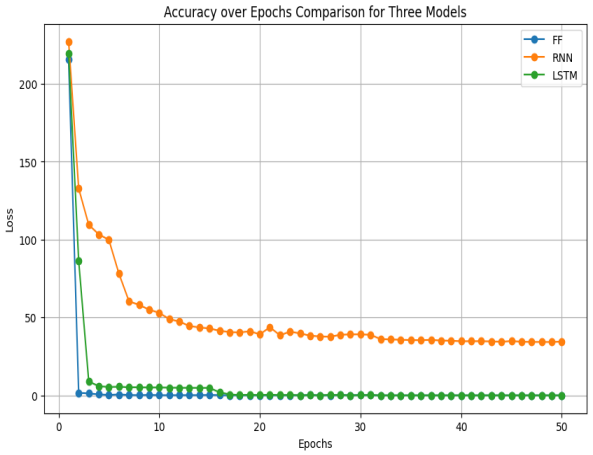

Figure 3. Loss Comparison on Dataset #2

4. Experimental Results

We conduced our experiments on the Gazebo Simulator with the Dart Physics engine [6]. This is a very realistic simulator which is used to test controllers before trying them on hardware. The same code that we use to run these tests in simulation is easily converted to the code necessary for hardware by simply changing some variable values in the thrust-throttle conversions and introducing safety measures and ROS2 topics necessary for hardware experimentation.

In the experiments we implemented the standard model-based predictors based on the linear and nonlinear dynamics. We then implemented the feedforward, and LSTM-based predictors and got these predictors working on the same piece of code with input statements in python to allow the user to select which predictor they would like to use in order to perform a comprehensive comparison for many methods in terms of accuracy as well as computation time. Unfortunately, we discovered very quickly that the well-known issues that recursive neural networks face regarding their vanishing and exploding gradients are another challenge that prevents them from being used in our controller. As was stated in Equation (1), our controller requires calculating the partial Jacobian of our predictor with respect to inputs and due to the RNN architecture's known problems with gradients, the Jacobian matrix calculation was not accurate enough to fly our quadrotors without immediately crashing.

4.1. Performance Comparison of Controllers

In this section we compare the performance of our Newton-Raphson flow controller with our various predictors as well as a different PID-based controller unrelated to the Newton-Raphson controller that is native to the PX4 flight stack [4]. This baseline controller is a cascaded PID architecture that includes an outer PID loop with reference trajectory as input and a desired acceleration as output, a nonlinear conversion from desired acceleration and yaw to desired attitude, and an inner PID loop that tracks the desired attitude by generating commanded body rate and thrust inputs, which are then converted to motor commands with the same mixer used for the proposed Newton-Raphson flow controller.

Therefore, in total we have 5 controllers including the native PX4 controller, and the Newton-Raphson-based controller with the linear, nonlinear, feedforward, and LSTM predictors. We test all of them on five reference trajectories: vertical circle, horizontal circle, horizontal lemniscate, vertical tall lemniscate, and vertical short lemniscate. This yields a total of 25 different trajectories that we ran in our comparative analysis. Note that "lemsniscate" trajectories refer to the "figure-8" or "infinity" shape. All trajectories are periodic with periods of 6.28 seconds. The results are shown in Figure 4. The root-mean-squared error (RMSE) of the tracking trajectory is reported in Table 2. We neglect the initial transient portion of flight for Figure 4 and Table 2 to focus our attention solely on the trajectory tracking portion of flight. In all cases, the Newton-Raphson flow controller outperforms the baseline controller.

4.2. Analysis of Results

As seen in Figure 4, all methods perform relatively well at a visual level. More precise analysis can be gleamed from Table 2 which allows us to compare the root-mean-squared error of the various methods. As can be seen, the nonlinear predictor performed the best generally, followed by the linear predictor, then the feedforward and LSTM model performing about

Table 2. Baseline vs Newton-Raphson with All Predictors RMSE on All Trajectories

Predictor/Control Type	Vert Circle	Horz Circle	Horiz Lemniscate	Vert Short Lemniscate	Vert Tall Lemniscate
Baseline Controller	0.23115	0.17167	0.20169	0.60109	0.26446
NR Linear	0.10784	0.051794	0.15755	0.17316	0.072669
NR Nonlinear	0.089295	0.098793	0.11829	0.1491	0.13284
NR FeedFWd	0.18393	0.097052	0.33244	0.12261	0.13211
NR LSTM	0.16067	0.11359	0.1146	0.20568	0.095147

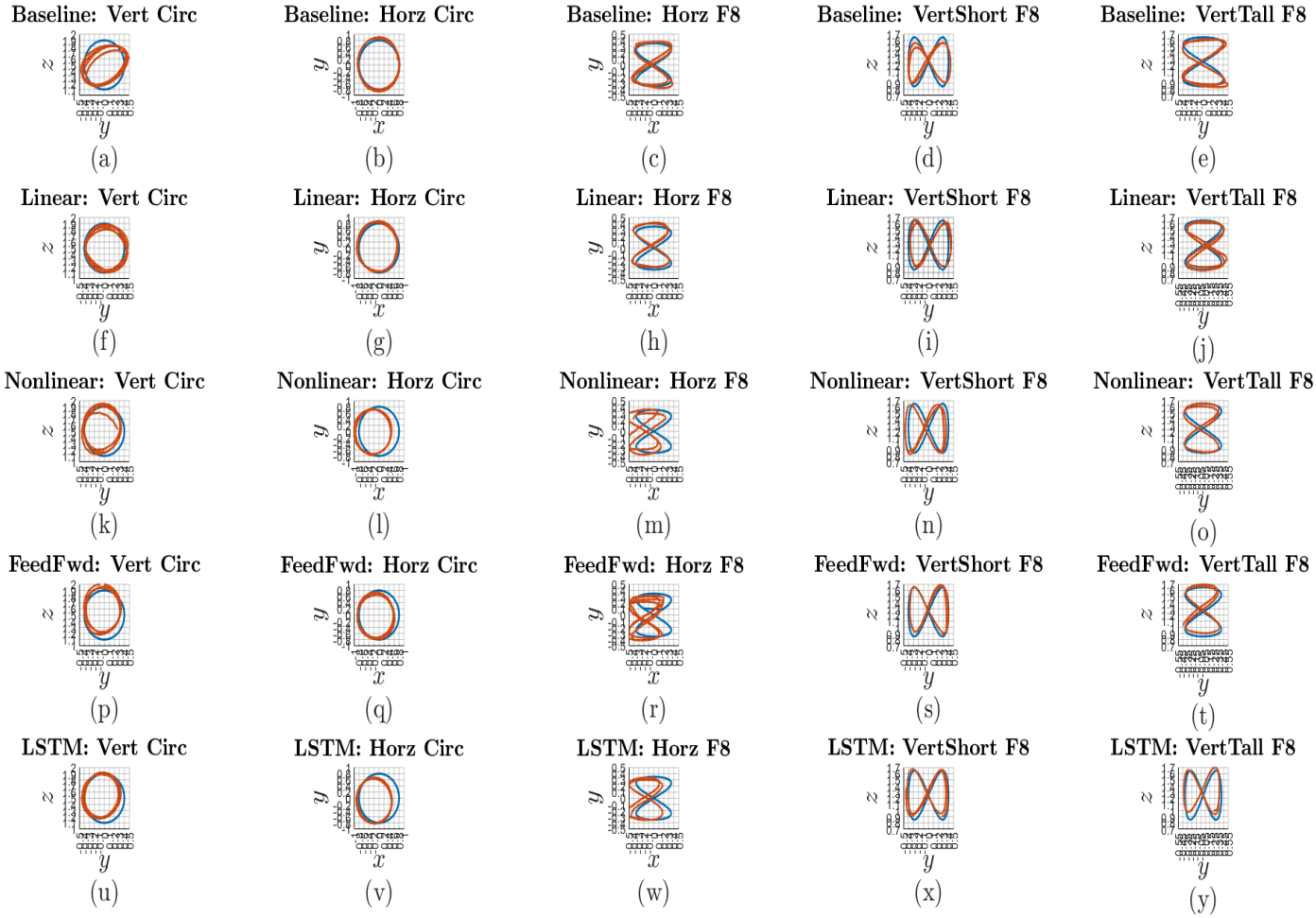

Figure 4. Comparison of five flight trajectories (red) with the reference trajectories (blue). From top to bottom we show the trajectories from the baseline controller, NR with Linear Predictor, NR with Nonlinear Predictor, NR with Feedforward Predictor, and NR with LSTM predictor. Axis units are meters. In all cases, the Newton-Raphson flow controller outperforms the baseline controller, evaluated with the root-mean-square error, as reported in Table 2.

equally well, and lastly the baseline controller. As is clear, the Newton-Raphson controller performed better than the baseline controller regardless of predictor.

Within the Newton-Raphson controller class, the more precise nonlinear predictor performed the best, which makes sense. Nevertheless, variability is seen in the results, as the linear predictor performed better than the nonlinear predictor on the horizontal circle and tall vertical lemniscate trajectories. And although the linear predictor generally performed better than the feedforward predictor, the latter did outperform the former on the vertical short lemniscate trajectory.

Table 3. Computation Times for Prediction-Only and Prediction+Jacobian

Predictor	Prediction Only(s)	Pred+Jacobian(s)
Linear	5.4155e-05	5.4155e-05
Nonlinear	1.3744e-05	5.8430e-05
FeedFWd	9.7558e-04	7.3060e-03
LSTM	1.7167e-03	6.0087e-03

The LSTM outperformed the feedforward on the vertical circle, horizontal lemniscate, and vertical tall lemsniscate. It is notable that the LSTM often performed among the top two predictors on a few trajectories and the feedforward predictor's performance was comparable to the nonlinear predictor's performance.

In terms of computation time, we expected the neural network prediction to be faster than the nonlinear predictor because the nonlinear predictor requires integrating the complicated dynamics via forward Euler integration, which requires many for-loops which are time consuming. As seen in Table 3, this was not the case in reality. However, we suspect that if we simply took the learned parameters from pytorch and implemented the neural networks via matrix multiplications on our own and hard-coded the backpropagation equations without the extra overhead that comes with pytorch that we might achieve faster results.

5. Conclusion and Next Steps

The analysis above shows us that the LSTM and feedforward predictors performed about on-par with the nonlinear on most trajectories, while giving up some computation time. During training, the feedforward architecture initially appeared to be the most effective for this task. However, upon experimentation, it became evident that the LSTM model, as initially hypothesized, was indeed marginally superior. In terms of computation time, while it may not seem like an advantage to give up computation time for roughly equal performance between traditional predictors and neural-network predictors, it is still a notable achievement to achieve prediction without requiring knowledge of the system dynamic equations. This is a great luxury in the field of control systems where some systems are difficult and expensive to identify and formulate in the form of dynamical systems equations. Moreover, we believe that we can speed up computation by separating our neural network implementations from the overhead of pytorch in order to achieve model-free prediction that performs about as well as the nonlinear predictor with similar computation time. We hope to make this simplified implementation of neural network prediction and Jacobian computation the focus of future research in order to achieve competitive computation speeds on the neural network predictors in comparison to the model-based predictors.

6. Team Contribution

Table 4. Team Contributions

Tasks:	Mahya	Evanns
Data Collection	✗	✓
NN Coding	✓	✓
Network Optimization	✓	✗
Jacobian Code	✓	✓
Quadrotor Implementation	✗	✓

References

[1] Moses Bangura and Robert Mahony. Real-time model predictive control for quadrotors. In The International Federation of Automatic Control, volume 19, 2014. 1

[2] E.Morales, C.Llanes, Y.Wardi, and S.Coogan. Newton-raphson flow for aggressive quadrotor tracking control. ACC. 2

[3] Pixhawk. pixhawk series. https://pixhawk.org/products/, 2023. [Online; accessed 2023-09-22]. 1

[4] PX4. Controller diagrams. https://docs.px4.io/main/en/flight_stack/controller_diagrams.html, 2023. [Online; accessed 2023-09-21]. 5

[5] Elias Reyes-Valeria, Rogerio Enriquez-Caldera, Sergio Camacho-Lara, and Jose Guichard. Lqr control for a quadrotor using unit quaternions: Modeling and simulation. In CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing, pages 172–178, 2013. 1

[6] Open Robotics. Gazebo physics. https://gazebosim.org/libs/physics, 2023. [Online; accessed 2023-09-28]. 5

[7] Ezra Tal and Sertac Karaman. Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. IEEE Transactions on Control Systems Technology, 29(3):1203–1218, 2021. 1

[8] Y. Wardi, C. Seatzu, J. Cortes, M. Egerstedt, S. Shivam, and I. Buckley. Tracking control by the newton-raphson method with output prediction and controller speedup. International Journal of Robust and Nonlinear Control, http://doi.org/10.1002/rnc.6976, 2023. 2

[9] Y. Wardi, C. Seatzu, M. Egerstedt, and I. Buckley. Performance regulation and tracking via lookahead simulation: Preliminary results and validation. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pages 6462–6468, 2017. 1