# UGP End Term Report
# CS396A, Semester 2017-18-II

# Neural Network based Modelling and Control of Quadrotor

Mrinaal Dogra
Roll No.: 150425
E-mail : mrinaald@cse.iitk.ac.in
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur.

Mentor : Pratyush Varshney
Roll No.: 16211402
E-mail : pratyushvarshney@cse.iitk.ac.in
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur.

Supervisor : Dr. Indranil Saha,
E-mail: isaha@cse.iitk.ac.in
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur.

April 26, 2018

**Abstract**

The physics-based models of the quadrotors are being used for a long time to synthesize their controllers. All such controllers work with the linearized model of quadrotor, thus becoming prone to errors in adverse situations because, in reality, the quadrotor models are non-linear.

Recent advancements in deep learning techniques have shown that these techniques are very efficient in learning non-linear functions. This motivates us to use them for learning the model of a quadrotor. This project aims at using neural network techniques to learn the model of the quadrotor and to use them for synthesizing a controller for it.

# 1 Introduction

## 1.1 Aim

The aim of this project is to first reproduce the results presented in [1], i.e., to learn the quadrotor dynamics using neural networks, and then to extend their work such that the synthesized controller can be used even when there are winds in the environment.

## 1.2 Motivation

In the real world, quadrotor dynamics is a non-linear system. However, the developed physics-based models work under an assumption that the system is linear, thus linearizing the non-linear system around the hover conditions. This assumption leads to certain errors in the developed models of the quadrotor, which also propagates to the controllers derived using them.

# 2 Previous work

In [1], the authors have tried to develop a quadrotor's model using neural networks, which is then used to generate the desired controls and the feasible trajectory for a given reference trajectory. Another approach has been presented in [2], where the authors have used reinforcement learning techniques to train a neural network, which can be used to control a quadrotor. Reinforcement learning techniques are also used to design a controller for flying a helicopter in an inverted fashion, as presented in [3]

# 3 Quadrotor Dynamics

As the derivation of the quadrotor dynamics will itself become a chapter, the major equations used for modelling the quadrotor[4] are described in subsequent sub-sections.

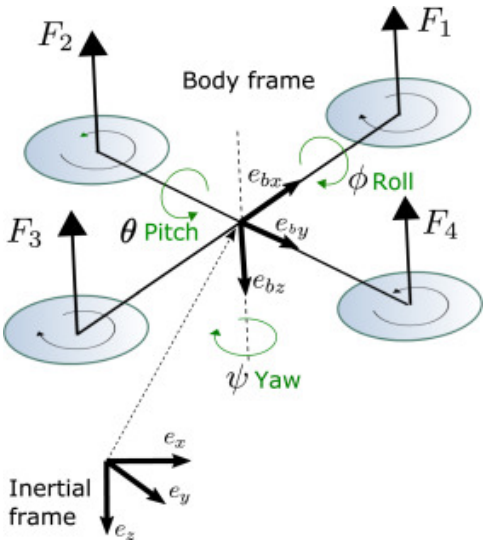


Figure 1: Frame of reference for Quadrotors[1]

[1]Image Source: `https://www.sciencedirect.com/science/article/pii/S0019057817305621`

## 3.1 State Variables

The twelve state variables of the quadrotors are as follows:

$$
\begin{aligned}
p = (x, y, z) &= \text{The inertial North-East-Down position of the quadrotor} \\
v = (\dot{x}, \dot{y}, \dot{z}) &= \text{The linear velocities measured in inertial frames} \\
\zeta = (\phi, \theta, \psi) &= \text{The Euler angles (roll, pitch, yaw) with respect to body fixed frame} \\
\omega = (\omega_x, \omega_y, \omega_z) &= \text{The angular velocities measured with respect to body fixed frame}
\end{aligned}
$$

## 3.2 Quadrotor System Model

The six degree of freedom model for the quadrotor kinematics and dynamics can be described with the following equations:

$$
\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi + s\phi c\psi \\ s\theta & -s\phi c\theta & -c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{1}
$$

$$
\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \omega_z \dot{y} - \omega_y \dot{z} \\ \omega_x \dot{z} - \omega_z \dot{x} \\ \omega_y \dot{x} - \omega_x \dot{y} \end{bmatrix} + \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{2}
$$

$$
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3}
$$

$$
\begin{bmatrix} \dot{\omega_x} \\ \dot{\omega_y} \\ \dot{\omega_z} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} \omega_y \omega_z \\ \frac{J_z - J_x}{J_y} \omega_x \omega_z \\ \frac{J_x - J_y}{J_z} \omega_x \omega_y \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x} \tau_\phi \\ \frac{1}{J_y} \tau_\theta \\ \frac{1}{J_z} \tau_\psi \end{bmatrix} \tag{4}
$$

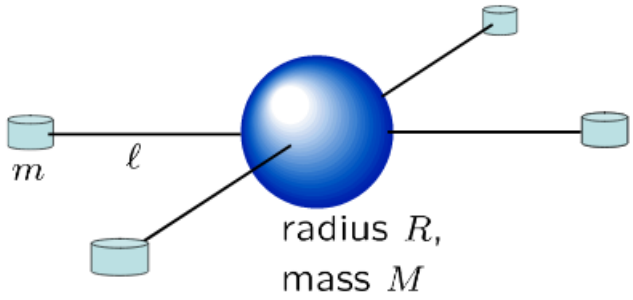


Figure 2: The moments of inertia for the quadrotor are calculated assuming a spherical dense center with mass $M$ and radius $R$, and point masses of mass $m$ located at a distance of $l$ from the center.[2]

where
$c\phi, s\phi, t\phi \rightarrow \cos\phi, \sin\phi$ and $\tan\phi$ respectively. Similarly for other angles
$f_x, f_y, f_z \rightarrow$ Forces along the $e_{b_x}, e_{b_y}, e_{b_z}$ axis respectively. Refer Figure 1
$J_x, J_y, J_z \rightarrow$ Moment of inertia of quadrotor along the three principal axes
$\tau_x, \tau_y, \tau_z \rightarrow$ Moments along the three principal axes

[2]Image and Caption source: [4]

# 4 Quadrotor System Identification

This section describes the system identification problem for a quadrotor system. As we were trying to reproduce the results of [1], most of these equations and explanations are referenced from there.

Let $s$ and $u$ be respectively the state vector and the control inputs of the dynamical system. Then, the goal of system identification is to find a mapping $f$ from state-control to state-derivative, i.e.:

$$\dot{s} = f(s, u; \alpha)$$

where the system model is parameterized by $\alpha$. The quadrotor state vector is a twelve dimensional vector which includes all the state variables described in section 3.1,

$$s = \begin{bmatrix} p \\ v \\ \zeta \\ \omega \end{bmatrix}$$

The Euler angles ($\zeta$) parameterize the coordinate transformation from inertial frame to body fixed frame with the standard *yaw-pitch-roll* convention, i.e., a rotation by $\psi$ about the $z$-axis in the inertial frame, followed by a rotation of $\theta$ about the $y$-axis of the body-fixed frame, and finally another rotation of $\phi$ about the $x$-axis in the new body-fixed frame. This is written compactly as:

$$\prescript{B}{I}{R}(\phi, \theta, \psi) = R_x(\phi) R_y(\theta) R_z(\psi) \tag{5}$$

where I is the inertial NED frame, B is the body-fixed frame, and $R_x, R_y, R_z$ are the basic $3 \times 3$ rotation matrices about their respective axes.

The control inputs $u = [u_1 \;\; u_2 \;\; u_3 \;\; u_4]$ are used to control the system, where $u_1$ is the thrust along the $z$-axis in $B$, and $u_2,\; u_3$ and $u_4$ are rolling, pitching and yawing moments respectively, all in $B$. Thus, the equation which governs the dynamics of our system:

$$\dot{s} = \begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{\zeta} \\ \dot{\omega} \end{bmatrix} = f(s, u; \alpha) = \begin{bmatrix} f_p(s, u; \alpha_1) \\ f_v(s, u; \alpha_2) \\ f_\zeta(s) \\ f_\omega(s, u; \alpha_3) \end{bmatrix} \tag{6}$$

where the system model is parameterized by $\alpha := (\alpha_1, \alpha_2, \alpha_3)$. The functions $f_p, f_v, f_\zeta$ and $f_\omega$, and the parameters $\alpha$ are explained in section 5. The system identification task for the quadrotor is thus to determine $\alpha_1$ (resp. $\alpha_2, \alpha_3$), given observed values of $f_p$ (resp. $f_v, f_\omega$), $s$ and $u$. For this, we minimize the mean squared prediction error (MSE) over a training set of collected data, solving:

$$\min_{\alpha_1} \sum_{t=1}^{T} \frac{1}{T} \left\| \tilde{f}_{p,t} - f_{p,t}(s_t, u_t; \alpha_1) \right\|_2^2 \tag{7}$$

where $\tilde{f}_{p,t}$ are the observed values of $f_p$. A similar optimization problem can be defined for $f_v$ and $f_\omega$

# 5 Neural Network Implementations

## 5.1 Same model as used in reference [1]

In order to reproduce the results of [1], we first tried the neural network architecture which the authors had used. The system identification problem which they had tackled is same, but the equations governing the quadrotor dynamics were different from (6)

$$\dot{s} = \begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{\zeta} \\ \dot{\omega} \end{bmatrix} = f(s, u; \alpha) = \begin{bmatrix} v \\ f_v(s, u; \alpha_1) \\ \hat{R}\omega \\ f_\omega(s, u; \alpha_2) \end{bmatrix} \tag{8}$$

where $\hat{R}$ is the rotation matrix used to get the Euler rates($\dot{\zeta}$), which can be obtained by rotating the angular velocities from the body-fixed frame to the inertial-frame[4] as follows:

$$\dot{\zeta} = \hat{R}\omega, \qquad \hat{R} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \tag{9}$$

Thus, their system identification problem only consists of determining $\alpha_1$ and $\alpha_2$(in our case $\alpha_2$ and $\alpha_3$ resp.)

The neural network architecture which they proposed consists of a single hidden layer with ReLu activation function and an output layer. The model can be represented algebraically as:

$$f_v(\beta; \alpha_1) = w^T \phi(W^T \beta + B) + b \tag{10}$$

where $f_v$ represents the unknown linear acceleration component in (8). This is modeled by a neural network whose input is $\beta := [v \quad \omega \quad \sin(\zeta) \quad \cos(\zeta) \quad u_1]$, a 13-dimensional vector; a hidden layer with $N := 100$ units with weight matrix $W \in \mathbb{R}^{|\beta| \times N}$ and bias vector $B \in \mathbb{R}^N$ , and a linear output layer of 3 units with weight matrix $w \in \mathbb{R}^{N \times 3}$ and bias vector $b \in \mathbb{R}^3$. $\phi$ is the ReLu activation function, i.e., $\phi(\cdot) = \max(0, \cdot)$.

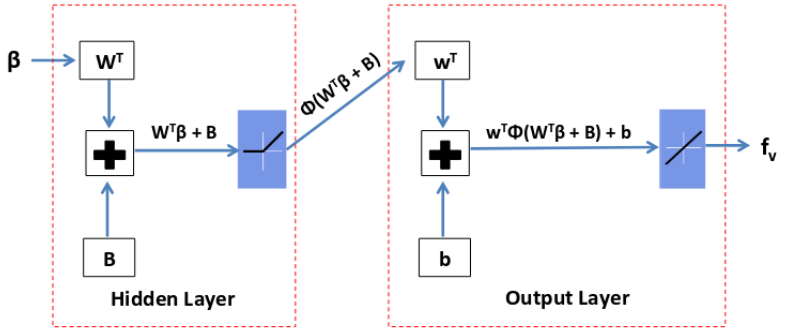


Figure 3: The neural network architecture used to learn $f_v$. The NN consists of two layers, a hidden ReLu layer and an output layer. The parameters to be learned during the training process are $\alpha_1 = (W, B, w, b)$.[3]

[3]Image and caption source: [1]

They used a similar architecture to learn $f_\omega$. The only difference was the input, which was $\beta := [v \;\; \omega \;\; \sin(\zeta) \;\; \cos(\zeta) \;\; u_2 \;\; u_3 \;\; u_4]$, a 15-dimensional vector. The dimensions of weight matrix $W$ and bias vector $B$ were also changed accordingly.

The authors then used the above two networks to generate a feasible trajectory and desired controls from an input reference trajectory using the optimization problem:

$$\begin{aligned} &\underset{s^{N_H}, u^{N_H}}{\arg\min} \sum_{n=0}^{N_H} \|s(n) - s_d(n)\|_2 \\ &\quad \text{s.t.} \;\; s(n+1) - s(n) = f(s(n), u(n); \alpha)\Delta t, \;\; n = 0, \ldots, N_H - 1 \end{aligned} \tag{11}$$

where, $n$ indexes the time step, $\Delta t$ is the sampling rate, $s(n)$ and $u(n)$ are the state and input of the quadrotor at time $n\Delta t$, $\alpha = (\alpha_1, \alpha_2)$ are the parameters learned during the neural network training, $s_d^{N_H} := \{s_d(0), s(1), \ldots, s_d(N_H)\}$ is the reference/desired trajectory over a horizon $N_H$.

The main problem which we faced while implementing [1] was the method which they were using to solve the optimization problem (11). This problem is briefly described in section 6.1. This led us to redefine our quadrotor system equations and system identification problem, resulting in equations (6) and (7).

## 5.2 Our Implementation

After failing to further implement [1], we modified the system identification equations to (6) and (7). This means that now, instead of learning two neural networks, we would learn three different networks to generate the state-derivative from the current state and controls of the quadrotor. The function $f_\zeta$ is taken directly from equation (8), i.e.,

$$\dot{\zeta} = f_\zeta(s) \qquad ; \qquad f_\zeta(s) = \hat{R}\omega \qquad (\hat{R} \text{ from equation (9)})$$

To keep things simple, we used almost the same neural network architecture for all the three cases. The neural network for $f_p$ and $f_v$ has an input of the form $\beta := [p \;\; v \;\; \zeta \;\; \omega \;\; u_1]$ a 13-dimensional vector, four hidden layers with 128 units each having weight matrices $W_{ih_1} \in \mathbb{R}^{|\beta| \times 128}, W_{h_1h_2}, W_{h_2h_3}, W_{h_3h_4} \in \mathbb{R}^{128 \times 128}$, and bias vectors $b_{h_1}, b_{h_2}, b_{h_3}, b_{h_4} \in \mathbb{R}^{128}$, and an output layer of 3 units with weight matrix and $W_{h_4o} \in \mathbb{R}^{128 \times 3}$. We have used the ReLu activation function for all the hidden layers.

We also used a similar architecture to learn $f_\omega$. The only difference was the input, which was $\beta := [p \;\; v \;\; \zeta \;\; \omega \;\; u_2 \;\; u_3 \;\; u_4]$, a 15-dimensional vector. The dimensions of weight matrix $W_{ih_1}$ and bias vector $b_{h_1}$ were also changed accordingly.

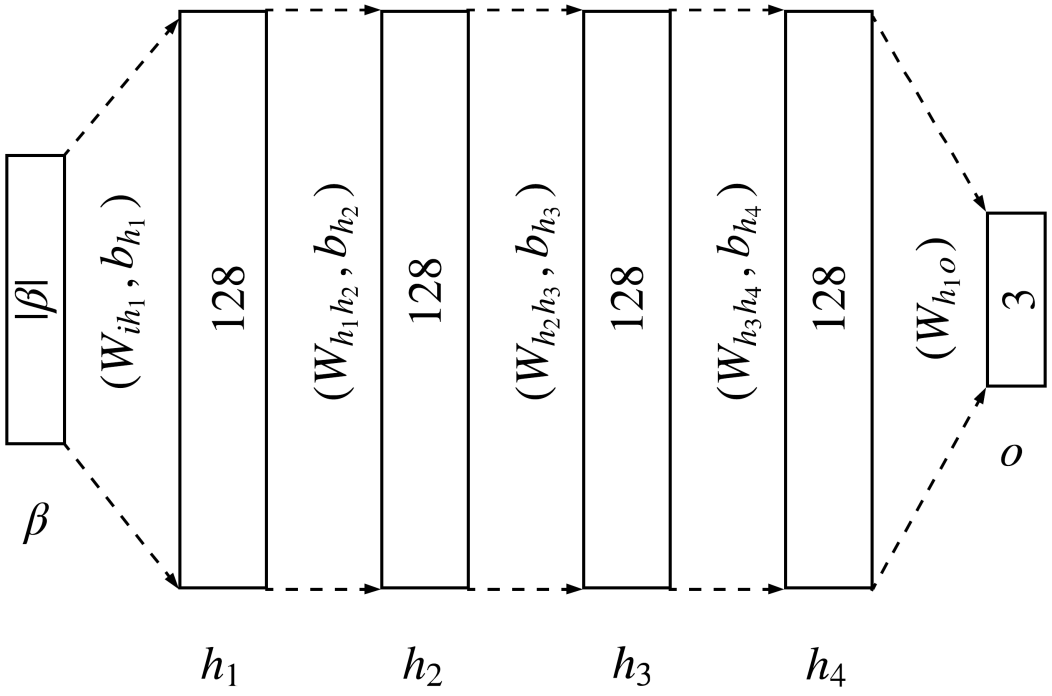


Figure 4: The final neural network structure which we used for $f_p$, $f_v$ and $f_\omega$. The parameters for this neural network structure are $\alpha_j = (W_{ih_1}, b_{h_1}, W_{h_1h_2}, b_{h_2}, W_{h_2h_3}, b_{h_3}, W_{h_3h_4}, b_{h_4}, W_{h_1o})$ where $j = 1, 2$ or 3. ReLu activation function is used for each hidden layer.

# 6 Challenges

## 6.1 SCP Implementation in Reference [1]

This was one of the major challenge which consumed a lot of our time. The authors of [1] have proposed to use the sequential convex optimization technique to solve (11), as the output of neural networks is non-linear, thus making it a non-convex problem. They haven't provided a lot of details about the way they solved this problem, and have suggested the reference for interested readers as [5].

However, even after reading it, we were not able to translate the optimization problem (11) into SCP. The authors of [5] have used the proposed algorithm on one-dimensional constraints which are definite in nature (like $x \leq \gamma$ for some $\gamma$ where $x \in \mathbb{R}$). They did not talk about having a vector or about a neural networks in the constraints. This lack of information proved a hurdle in our project, as we were unable to implement the optimization problem (11) in any way. Ultimately, we decided to change our quadrotor system equations and to use some alternate ways to solve the optimization problem.

## 6.2 Required Data for training

Unlike the authors of [1] who flew a real Crazifly-2.0 to collect the required data, we used a simulation to gather the data required for training. This simulation was run using `Mavros`[4], PX4[5] and Gazebo using the PX4-ROS(Mavros) interface[6].

The training data which we used is of the form:

$$\beta_1 = [p \;\; v \;\; \zeta \;\; \omega \;\; u_1]$$
$$\beta_2 = [p \;\; v \;\; \zeta \;\; \omega \;\; u_2 \;\; u_3 \;\; u_4]$$

where $\beta_1$ was used for training the neural networks for $f_p$ and $f_v$, and $\beta_2$ was used for $f_\omega$. This required data was collected through the following mavros topics[7]:

- $p = (x, y, z)$ was collected from rostopic `/mavros/local_position/pose`
- $v = (v_x, v_y, v_z)$ was collected from rostopic `/mavros/local_position/velocity`
- $\zeta = (\phi, \theta, \psi)$ was collected by converting the quaternions obtained from rostopic `/mavrox/imu/data/Quaternion` into Euler angles using python's `tf.transformations` library[8].
- $\omega = (\omega_x, \omega_y, \omega_z)$ was collected from the rostopic `/mavros/imu/data/angular_velocity`
- $u = (u_1, u_2, u_3, u_4)$ was collected from the rostopic `/mavros/act_controls`

The architecture of the required mavros topics in the PX4-ROS interface is shown in figure 5. Note that the PX4 does not publish the actuator controls as for now, thus we had to create a new message[9] as well as its publisher for the topic `/mavros/act_controls` in the PX4 architecture.

The output ground truth values for $\dot{s}$ was created by taking the approximate derivative of the next state with the current state with respect to time, i.e.,

$$\dot{s}_n = \frac{s_{n+1} - s_n}{\Delta t} \tag{12}$$

where $\Delta t$ was the time difference between two consecutive observed states. All the mavros publishers were synchronized to publish the data at 50Hz, thus making $\Delta t = 0.02$ seconds.

We collected all this data for 23 different trajectories, which includes sinusoid trajectories in XY, YZ and XZ directions, and some completely random trajectories, trying to cover all the possibilities of the states and controls.

[4] `http://wiki.ros.org/mavros`
[5] px4.io/
[6] `https://dev.px4.io/en/simulation/ros_interface.html`
[7] To read about ROS Topics, visit `http://wiki.ros.org/Topics`
[8] `http://docs.ros.org/jade/api/tf/html/python/transformations.html`
[9] To read about ROS messages, visit `http://wiki.ros.org/Messages`

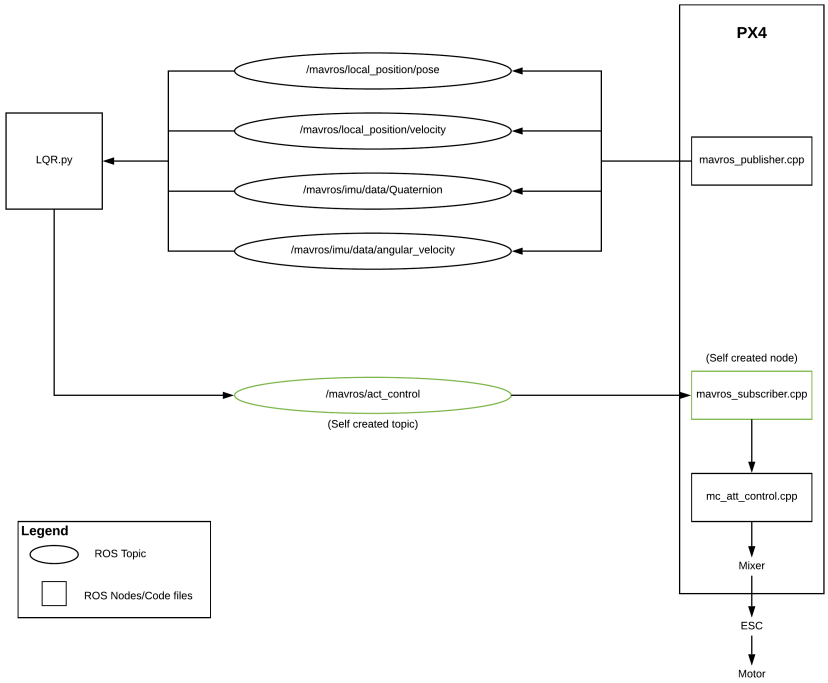


Figure 5: The schematic diagram of the PX4-Mavros interface which is used to collect the data for training. The green colored topic and node were not initially present in PX4. They were created to help us collect the data for actuator controls.

# 7 Results

After training the above networks, we tested the output generated by them for a circular trajectory(which was not present in the training dataset) and the results were quite satisfactory:

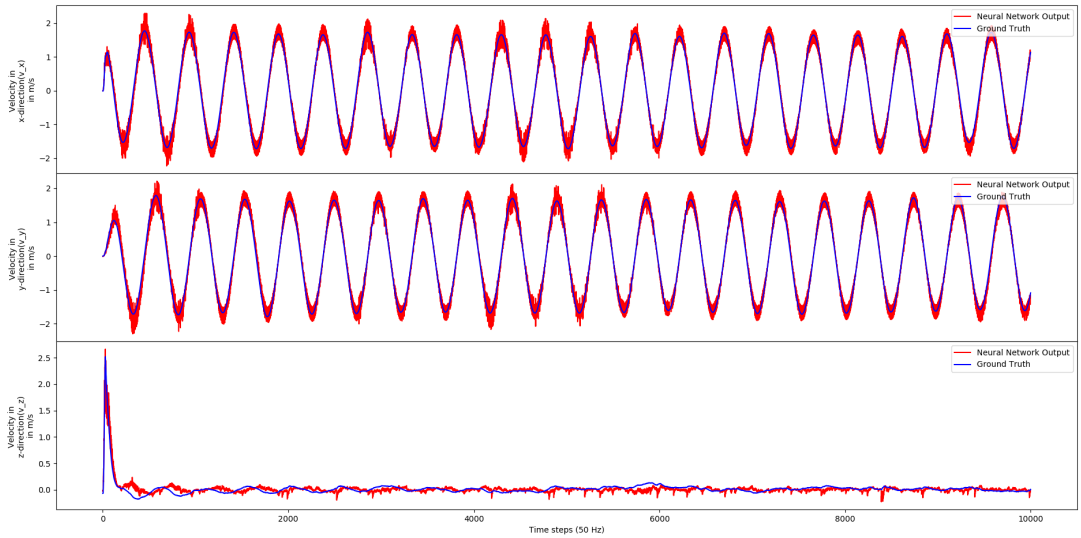


Figure 6: Plots for $f_p$. These plots represent the linear velocities of the next state as measured from the dataset(the blue plot) and the linear velocities of the next state computed by neural network $f_p$.

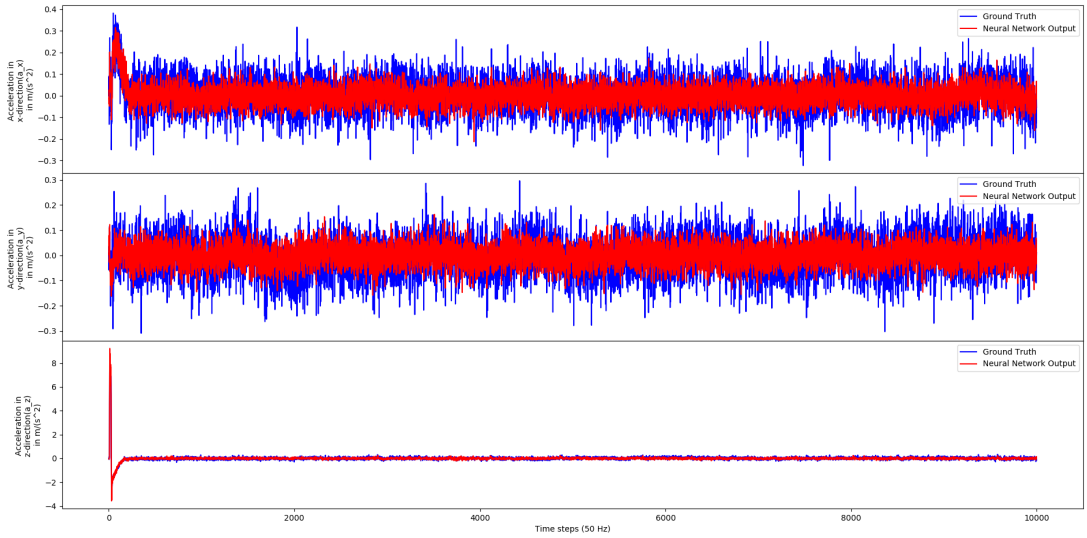


Figure 7: Plots for $f_v$. These plots represent the linear accelerations of the next state as measured from the dataset(the blue plot) and the linear accelerations of the next state computed by neural network $f_v$.

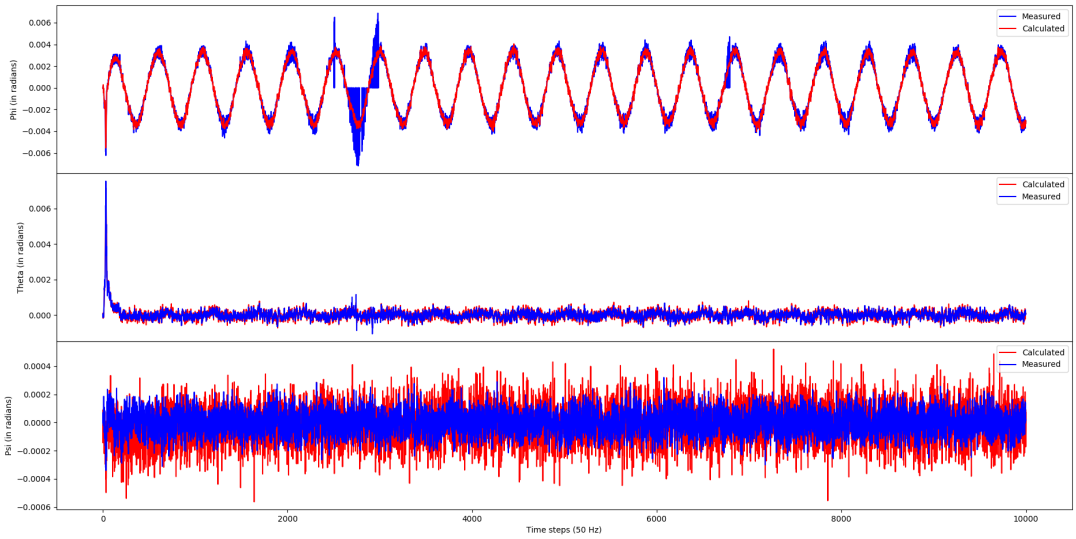


Figure 8: Plots for $f_\zeta$. These plots represent the Euler angles of the next state as measured from the dataset(the blue plot) and the angles of the next state computed by using $f_\zeta$.

# 8 Future Work

We will try to learn a controller using some deep learning techniques by trying to solve the optimization problem mentioned in equation (11), to generate discrete controls for the given horizon. We will also try to implement some Reinforcement Learning based algorithms for continuous control generation like Trust Region Policy Optimization (TRPO), Deterministic Deep Policy Gradient (DDPG). We will incorporate the effects of wind on these controls also.

# References


[1] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," *CoRR*, vol. abs/1610.05863, 2016.

[2] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[3] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX, The 9th International Symposium on Experimental Robotics [ISER 2004, Singapore, 18.-21. June 2004]* (M. H. A. Jr. and O. Khatib, eds.), vol. 21 of *Springer Tracts in Advanced Robotics*, pp. 363–372, Springer, 2004.

[4] R. Beard, "Quadrotor dynamics and control," 04 2018.

[5] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013* (P. Newman, D. Fox, and D. Hsu, eds.), 2013.