Learning a Low-Level Motor Controller for UAVs

Joseph Lorenzetti

***Abstract*— Many control algorithms for Unmanned Aerial Vehicles (UAVs) have been proven to be effective for standard flight tasks under nominal conditions (i.e. when effects from complex interactions with the environment are small). For UAVs, and quadcopters in particular, these complex dynamics generally originate from aerodynamic effects and internal motor dynamics. Accounting for these effects *during* controller design and implementation is very difficult, and therefore a modular low-level controller is explored in this paper that *corrects* the true control inputs to account for unmodeled effects. The proposed low-level controller will map the desired inputs from the primary controller directly to a motor command. Three different models for the low-level motor controller are proposed, and they are compared against the standard model structure used in the open-source PX4 autopilot firmware.**

I. Introduction

Over the last two decades, the development of powerful, small, and cheap electronics has driven advances in increasingly versatile autonomous systems. For example, UAVs have become increasingly viable platforms for many commercial and military tasks including package delivery, power line inspection, wildlife conservation, building inspection, precision agriculture, and surveillance. However, as the demand for UAV operations moves into more complex and high performance environments, the autonomous control systems must also become more advanced. Beyond UAVs, this is also true for industrial and home robotics, self-driving cars, commerical aerospace applications, and more.

For UAVs, control systems have been developed using techniques from classical control theory (e.g. PID control), modern control theory (e.g. model predictive control (MPC)), and learning based control. Classical control techniques are widely applied in standard open-source hobby autopilot systems because of their effectiveness and simplicity, but are typically limited to small nominal operating regimes. Modern control techniques, such as MPC, introduce a notion of control optimality and can explicitly handle state and control constraints. This makes these types of techniques attractive for high performance tasks, but can end up being computationally constrained and often rely on dynamics models which are sometimes inaccurate.

A broad array of learning based control schemes for UAVs have also been proposed. Some approaches use learning methods for high level task control, such as converting image data into high level plans or even low level motor commands using convolutional neural networks [1]. Others use reinforcement learning techniques, such as a Guided Policy Search to learn how to map sensor readings directly to motor commands [2], or to train neural networks to map vehicle state to motor commands [3]. Beyond learning to control directly, machine learning techniques have also been used to learn models of the dynamics, which can then be used for predictive control [4]–[6]. The primary advantage of data driven techniques is that the true system behavior can be inferred without developing extremely complex models. However the disadvantages generally stem from the large amounts of data required, the inability to rigorously guarantee state and control constraints will remain satisfied (and in general verifiability), and in some situations high computational cost.

While many techniques (i.e. classical, modern, and learning based) have been proposed for designing UAV controllers, additional work has gone into developing algorithms that *augment* existing controller designs to improve performance. For example, [7], [8] propose a way to generate trajectories that can easily be tracked by the controller via learning by demonstration. Trajectory tracking is also addressed in [9] with the use of a deep neural network (DNN) for controller input augmentation. A DNN is also used in [10] to learn a disturbance model that is coupled with a nonlinear controller that helps quadcopter landing performance by correcting for the well known aerodynamic effect called the "ground effect".

Contributions: Similarly, this work does not focus on controller design, but rather on a modular component that can improve the performance of any given controller. In particular, a low-level motor controller is proposed that takes the output of the primary controller and maps it to motor commands. This low-level controller is modular in the sense that it could be used with *any* type of primary controller, and by using a data-driven approach it has the potential to *a posteriori* correct for complex effects that may not be known by the primary controller.

II. Problem Description

As mentioned previously, the proposed low-level motor controller maps the output of the primary controller to motor commands. For this work, which focuses specifically on quadcopter control, the output of the primary controller is assumed to be the desired acceleration along the quadcopter's thrust axis along with the desired angular acceleration about each of the three rotational axes. For better physical intuition these outputs will be referred to as the desired *net* force, F_z, and the desired *net* moments, $M_{x,y,z}$. The kinematics and dynamics of the quadcopter system will now be discussed, followed by a discussion on how the low-level controller will be designed.

A. Quadcopter Dynamics

For a quadcopter, the state of the system can be defined by its position, orientation, and their rates of change. To identify position and orientation of a rigid body two reference frames are defined: an inertial frame $\mathcal{W}$ that is fixed, and a body-fixed frame $\mathcal{B}$ that is considered attached to the rigid body. Using the conventions from [6], these reference frames are

defined as shown in Figure 1. The inertial frame $\mathcal{W}$ and

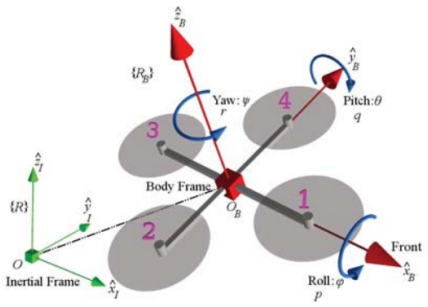

Fig. 1: Quadrotor Reference Frame

the body fixed frame $\mathcal{B}$ are then related using a standard Euler angle formulation (Yaw-Pitch-Roll) (ψ, θ, ϕ) or via the rotation matrix

$$R_{\mathcal{W}}^{\mathcal{B}} = \begin{bmatrix} c_\psi c_\theta & c_\theta s_\psi & -s_\theta \\ c_\psi s_\theta s_\phi - c_\phi s_\psi & c_\psi c_\phi + s_\psi s_\theta s_\phi & c_\theta s_\phi \\ s_\psi s_\phi + c_\psi c_\phi s_\theta & c_\phi s_\psi s_\theta - c_\psi s_\phi & c_\theta c_\phi \end{bmatrix} \quad (1)$$
$$R_{\mathcal{B}}^{\mathcal{W}} = R_{\mathcal{W}}^{\mathcal{B}\ T}.$$

The quadrotor translational dynamics are given by

$$m\ddot{r} = -mg\hat{z}_{\mathcal{W}} + F_T \hat{z}_{\mathcal{B}} + F_{\text{aero}}, \quad (2)$$

where r is the position vector $r = [x, y, z]^T$, m is the mass of the quadcopter, g is the acceleration due to gravity, $\hat{z}_{\mathcal{W}}$ is an inertially fixed unit vector pointing upward, and $\hat{z}_{\mathcal{B}}$ is a unit vector fixed in the body frame pointing along the body z axis. The scalar input F_T corresponds to the net *thrust* produced by the propellers and the vector F_{aero} encompasses the aerodynamic lift and drag that affect the system.

The rotational dynamics are given by

$$\dot{\omega} = I^{-1}\left[-\omega \times I\omega + \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}\right]. \quad (3)$$

where I is the inertia matrix, defined in the body fixed frame, and $M_{x,y,z}$ are moments that are applied about each body fixed frame axis. The angular velocity ω is defined in the body frame as

$$\omega = p\hat{x}_B + q\hat{y}_B + r\hat{z}_B.$$

B. Problem Formulation

As mentioned, it is assumed that the primary controller outputs a desired *net* force, F_z, and the desired *net* moments, $M_{x,y,z}$ which will produce the desired motion. These values must then be mapped to commands that will be sent to the motors. The challenge is that accurately determining the correct motor commands can be a complex task due to unknown effects from the aerodynamics of the quadcopter and propellers, and the dynamics of the motors themselves. For example, consider the following scenarios:

1) The controller commands the quadcopter to perform a vertical climb with some fixed acceleration.
2) The controller commands the quadcopter to land.
3) The controller commands the quadcopter to fly at high speed at a constant altitude to a new location.
4) The quadcopter battery levels are running low.

In the first scenario, as the quadcopter speeds up the drag will increase, and therefore the propellers will have to do more work to maintain a constant acceleration. In the second scenario, the propeller "ground effect" would cause the quadcopter to experience a higher acceleration for the same motor command as it gets closer to the ground. Similarly, in the third scenario propeller effects would have a tendency to produce lift during high speed translational motion. Finally, in the fourth case the thrust produced by a propeller could have decreased for the same motor command.

It would be desireable to have a low-level motor controller that can account for these effects, in order to make the quadcopter react as closely as possible to the desired motion from the primary controller. To accomplish this, a learning based approach will be used to generate a model g such that

$$\omega_m = g(F_z, M_x, M_y, M_z, \omega, r, \dot{r}, \phi, \theta, \psi, \dots). \quad (4)$$

Three different models are proposed, with varying structure and inputs. The set of possible inputs includes information about the state of the quadcopter that would be available online, and primarily includes the angular velocity ω, position r, velocity $\dot{r}$, and orientation (ϕ, θ, ψ). From a practical perspective the structure of the models are also limited in complexity due to the runtime requirements. At the very low level of the flight stack that this controller would be implemented, operations must be performed on the order of hundreds of times per second.

III. Dataset

The dataset is from flight experiments[1] of a Pelican quadcopter manually piloted by an experienced pilot. This dataset includes kinematic data (i.e. position, velocity, orientation) from a motion capture system and motor command data logged on the flight computer. In total there are over 250 minutes of flight data, comprised of more than a million data points from 55 flights. Figure 2 shows typical flight data.

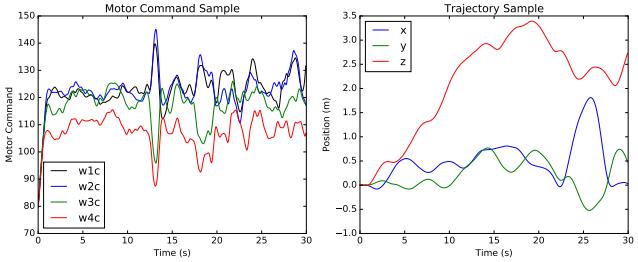

Fig. 2: Example data: motor commands and position trajectories.

From this core dataset additional information is also computed. First, the translational and angular accelerations are computed using numerical differentiation[2] Next the actual net thrust and moments can be computed from the dynamics equations (2, 3) from the kinematic data. Note that computing the net *thrust* F_T is difficult without knowledge of F_{aero}. However the objective is to learn a mapping where the input

[1] github.com/wavelab/pelican_dataset

[2] After differentiation, a local polynomial regression smoothing algorithm (LOESS [11]) is also used.

is the desired net *force* F_z, which could potentially include aerodynamic forces. The desired net *force* is therefore computed from the kinematic data as

$$F_z = (m\ddot{r} + mg\hat{z}_{\mathcal{W}}) \cdot \hat{z}_{\mathcal{B}}, \tag{5}$$

which is simply the *net* force projected onto the body z axis (and therefore captures forces from any type of effect).

In addition to the net force F_z and moments $M_{x,y,z}$, two additional quantities are computed as possible features for the models. These quantities are the axial velocity, V_a, and transverse velocity, V_t seen by the propellers. These quantities are computed as follows

$$\begin{aligned} V_a &= \dot{r} \cdot \hat{z}_{\mathcal{B}}, \\ V_t &= ||\dot{r} - V_a \hat{z}_{\mathcal{B}}||_2. \end{aligned} \tag{6}$$

Once the data processing is complete the available variables that can be used as features include the position r, velocity, $\dot{r}$, acceleration $\ddot{r}$, orientation (ϕ, θ, ψ), angular velocity ω, angular acceleration $\dot{\omega}$, and the velocities (V_a, V_t). All of the data was also normalized using the emprical mean and standard deviation.

IV. Models

In this work three different models for the low-level motor controller are proposed, and they are compared against the standard model structure used in the open-source PX4 autopilot firmware.

A. Baseline Model

In current open-source autopilots, the low-level motor controllers map desired thrust and moment commands to motor commands based on a very simple model of how thrust correlates to motor commands for a given propeller. This relationship is generally determined using static thrust stand testing, which removes many of the extra effects that are seen in real flight scenarios.

The baseline model first uses a linear mapping to convert the net force and moments into thrust for each individual motor. This linear mapping is given by

$$\begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{4k_T} & 0 & \frac{-1}{2k_T L} & \frac{-1}{4k_M} \\ \frac{1}{4k_T} & \frac{1}{2k_T L} & 0 & \frac{1}{4k_M} \\ \frac{1}{4k_T} & 0 & \frac{1}{2k_T L} & \frac{-1}{4k_M} \\ \frac{1}{4k_T} & \frac{-1}{2k_T L} & 0 & \frac{1}{4k_M} \end{bmatrix} \begin{bmatrix} F_z \\ M_x \\ M_y \\ M_z \end{bmatrix} \tag{7}$$

or its inverse

$$\begin{bmatrix} F_T \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ 0 & k_T L & 0 & -k_T L \\ -k_T L & 0 & k_T L & 0 \\ -k_M & k_M & -k_M & k_M \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4, \end{bmatrix} \tag{8}$$

where k_T is a thrust coefficient, k_M is a moment coefficient, L is the distance between the propeller axis and the center of mass, and o_i is the output command for propeller i.

The output for propeller i is then related quadratically to the motor command $\omega_{m,i}$

$$o_i = p_0 + p_1 \omega_{m,i} + p_2 \omega_{m,i}^2. \tag{9}$$

The models (7) and (9) can then be combined, and the parameters combined, to generate the full model

$$\omega_m = k_0 + \sqrt{k_1 + Au} \tag{10}$$

where

$$A = \begin{bmatrix} m_0 & 0 & -m_1 & -m_2 \\ m_0 & m_1 & 0 & m_2 \\ m_0 & 0 & m_1 & -m_2 \\ m_0 & -m_1 & 0 & m_2 \end{bmatrix}, \quad u = \begin{bmatrix} F_z \\ M_x \\ M_y \\ M_z \end{bmatrix}$$

and the square root is element-wise.

When developing this model it is important to note that the square root is only defined for non-negative arguments, which must be considered in the training process.

B. Proposed Models

Three new models are now proposed in an attempt to improve upon the performance of the baseline model. The first two models are variants of the baseline that incorporate additional features but keep the same structure, and the third uses a completely different structure (specifically a neural network).

The advantage of using the same model structure for the first two models lies in the inherent simplicity of the model structure. As mentioned previously, the models need to be evaluated at extremely high rates and so low complexity is desired. Additionally, keeping the same structure and adding features will provide some additional insight into whether the model structure is inherently useful or not. Additionally, in each model the linear mapping (7) is used.

1) Baseline + Feature Subset: The first proposed model keeps the same structure as the baseline, but includes a hand-picked subset of the available features. In particular, the features that are included are the altitude z, the velocities (V_a, V_t), and the angular velocity ω. These features will be denoted as the vector $x_1 = [z, V_a, V_t, \omega]^T$. Additionally, second order polynomial features of x are included (i.e. $x_i x_j$ and x_i^2). The combined set of features are denoted as $\phi(x)$. Finally, these features are used to augment the original mapping between o_i and $\omega_{m,i}$

$$o_i = p_0 + p_1 \omega_{m,i} + p_2 \omega_{m,i}^2 + w^T \phi(x_1), \tag{11}$$

such that the model becomes

$$\omega_m = k_0 + \sqrt{k_1 + Au + w^T \phi(x_1)}. \tag{12}$$

2) Baseline + Features: The second model is identical to the first model proposed, except that all available features are used. Namely $x_2 = [z, r, \dot{r}, \ddot{r}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, \omega, \dot{\omega}, V_a, V_t]^T$. Once again second order polynomial features of x are also included, with all features denoted by $\phi(x_2)$. The model is then defined by

$$o_i = p_0 + p_1 \omega_{m,i} + p_2 \omega_{m,i}^2 + w^T \phi(x_2), \tag{13}$$

such that the model becomes

$$\omega_m = k_0 + \sqrt{k_1 + Au + w^T \phi(x_2)}. \tag{14}$$

3) Neural Network: Apart from the initial linear mapping, the neural network model deviates in structure from the previous models and includes more parameters. For this model a bias term is also added to the initial linear mapping, such that

$$o = Au + b \tag{15}$$

where b is a vector of bias terms. Then, the individual motor outputs o are combined with the full set of features x_2 to become the input to the neural network. The same neural network is used for each motor, such that for a neural net f,

$$\omega_{m,i} = f(o_i, x_2). \tag{16}$$

The network used has a single hidden layer with 10 nodes and ReLU activation, and a linear output layer.

C. Training

The baseline model and the extended baseline models with features were trained using mini-batch stochastic gradient descent. The learning rate and batch sizes were tuned for good training performance, and the learning rate was decayed after each training epoch. A ReLU activation function was also added to the baseline and extended baseline models to ensure the square root argument was non-negative during the training phase. The neural network was trained using the Adam optimization technique, also with tuned hyperparameters and learning rate scheduling.

The training and validation datasets were the same for each model, and were selected by randomly taking samples from the full dataset in a 60/20/20 split for training, validation, and testing. For each case the mean squared error loss was used for training and for evaluation. This loss is defined as

$$l(\Theta) = \frac{1}{m}\sum_{i=1}^{m} ||\omega_m - \hat{\omega}_m(\Theta)||_2^2 \tag{17}$$

where m is the size of the batch, ω_m is the data point, and $\hat{\omega}_m(\Theta)$ is the prediction from the model parameterized by Θ. In each case the optimization algorithm is a first-order method, meaning that they rely on the gradient of the loss function with respect to model parameters. For this work the entire training procedure was peformed using the open source deep learning platform PyTorch [12].

Unfortunately, the loss function in these cases are non-convex and therefore the training process is susceptible to local minima. A stochastic optimization method such as SGD or Adam can sometimes help in these situations, but since the models are relatively small and do not take long to train, a batch of models were simultaneously trained. In this particular case instead of using the entire batch of trained models in an aggregate or bagged way, only the model with the lowest validation score is used.

The results of training each of these models can be seen in Figure 3, which shows the training loss over each optimization iteration. The bands represent the standard deviation and the solid line is the mean.

V. Results

After training and selection, the three proposed models and the baseline model were evaluated on the test dataset. Table I shows the resulting losses on the training, validation, and test datasets. For the training and validation sets, the presented value represents the mean loss over the batch of models trained. The loss figures for the test dataset are for the best model (i.e. the one with the lowest validation loss). This explains why the losses for the test set are typically lower. The variance that is seen is also likely due to the stochasticity inherent in the training process and the dataset.

	Training	Validation	Test
Baseline	0.599	0.568	0.411
Baseline + Feature Subset	0.483	0.505	0.426
Baseline + Features	0.393	0.422	0.389
Neural Network	0.240	0.264	0.257

TABLE I: Training, validation, and test dataset losses for each model.

From these results it can be seen that the neural network provides the best results, and in particular almost a 40% decrease in the loss value over the baseline. It also can be seen that the baseline model that is extended to include all features also performs slightly better, with about a 5% decrease in the loss. The baseline model that is extended to include the hand-picked features actually performed slightly worse on the test set. In theory the extended models should have strictly greater performance because the baseline model can be recovered by setting the additional feature weights to be zero. However as discussed previously, this does not always happen in practice due to the presence of local minima in the training process.

A. Other Performance Metrics

In addition to evaluating the performance of the models using the mean squared error loss score, a more qualitative analysis can be done. Inherently the mean squared error loss trains a model to have small error, and in particular penalizes larger errors worse than small errors. But it is also interesting to look at the actual errors and their distributions. The error for motor i is simply defined as

$$e_i = \omega_{m,i} - \hat{\omega}_{m,i}. \tag{18}$$

Over the test dataset the mean error and standard deviation of the error can also be computed

$$\begin{aligned} \mu_i &= \frac{1}{N}\sum_{i=1}^{N} e_i, \\ \sigma_i^2 &= \frac{1}{N}\sum_{i=1}^{N} (e_i - \mu_i)^2. \end{aligned} \tag{19}$$

These quantities are of interest because it is desireable that the mean be as close to zero as possible, along with a small variance. If the mean is not zero it suggests that there is some inherent bias in the model. It is also desireable to have a decreased variance because the hope is that the more complex models are able to better capture the effects causing the largest errors.

The mean and standard deviation of the error are computed for each model and the results are shown in Figure 4. Interestingly, the means seem to be quite varying. The neural network does seem to consistently perform pretty well, but in other cases the best performer is not clear. Although in each case the mean is still close to zero, which is desireable. It is likely that the varying performance is due to the fact that only one model is developed and applied to each motor individually. Therefore if each of the motors had slightly different performance, it would be difficult for any one model to generalize to all. The standard deviation plot also

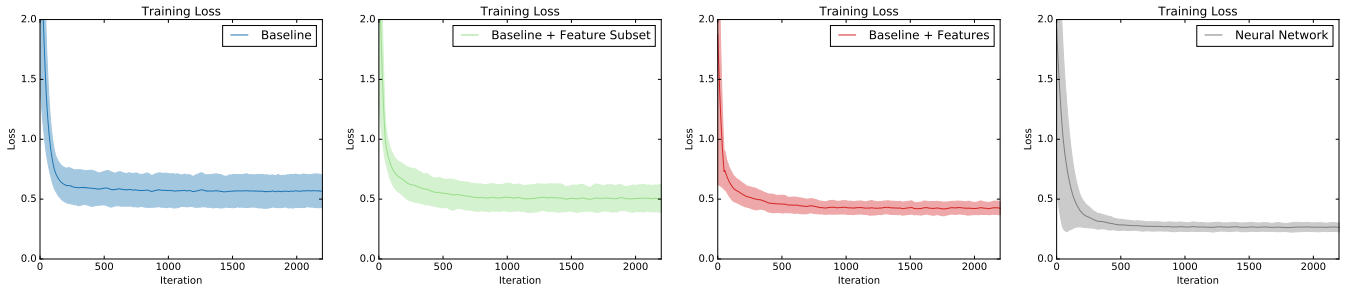

Fig. 3: Training loss at each optimization iteration.

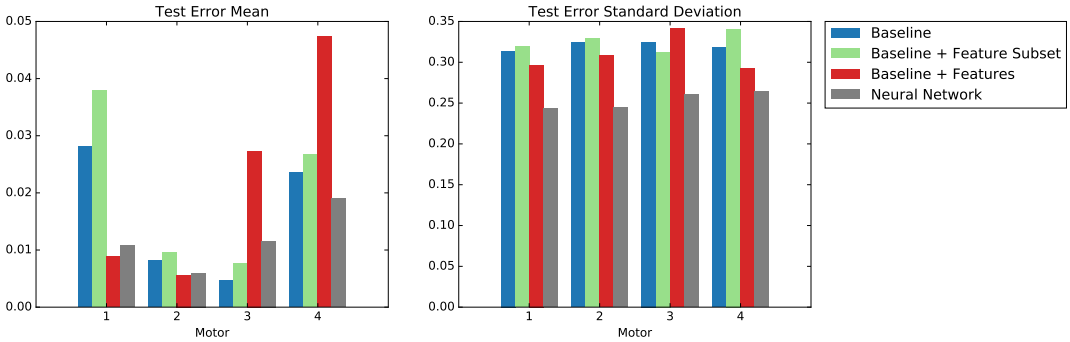

Fig. 4: Mean and standard deviation of the errors.

provides some qualitative information. It can be seen that the neural network consistently outperforms the others, which is expected from the lower MSE loss. Once again, among the other models it is not clear that any one outperforms the other.

The distributions of the errors can also be plotted to provide a more visual presentation of the information previously discussed. These plots are provided in Figure 5 to compare the best performing model (neural network) against the baseline model. As can be seen the neural network model

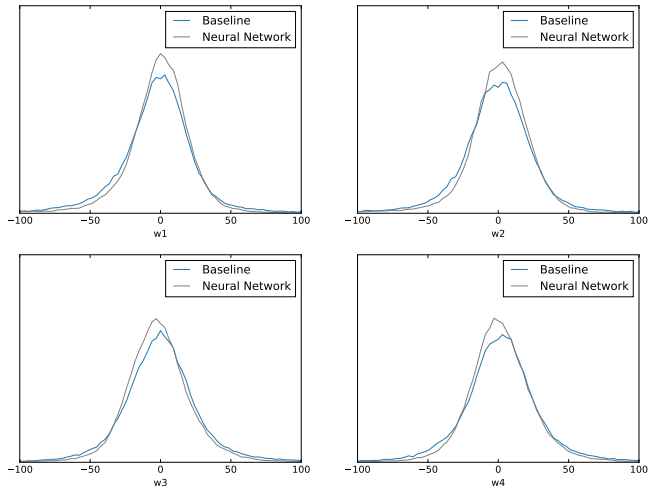

Fig. 5: Error distributions for the baseline and neural network models.

has tightened the distribution such that there are fewer data points with extremely high error.

VI. Conclusion

The objective of this work was to create a better performing low-level motor controller using a learning based approach. Where current low-level motor controllers fail is in the ability to capture off nominal effects, which can lead to poor performance in off nominal flight situations. From the results of the proposed models, simply augmenting the current methods to include additional features will not be sufficient to drastically increase performance. However the neural network, while only providing a moderate performance boost, provides some promising results that suggest learning based approaches may still be useful in this domain.

VII. Future Work

As discussed, the results suggest that learning based approaches to this probem may have potential but likely require more complex models. Some future directions will therefore definitely include exploring more expressive models, such as neural networks with more layers or potentially even recurrent neural networks. Recurrent neural networks may be effective because the regression is fundamentally on time series data, which suggests that a feedforward network is not utilizing all available information. Additionally, the dataset that was used was not collected for the specific purpose of this project. Therefore it would be interesting to explore data collection methods that may provide better training performance, especially if recurrent neural networks are used.

References

[1] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, and P. Campoy, "A review of deep learning methods and applications for unmanned aerial vehicles," *Journal of Sensors*, 2017.

[2] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 528–535.
[3] 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, Oct 2017.
[4] X. Bao, Z. Sun, and N. Sharma, "A recurrent neural network based MPC for a hybrid neuroprosthesis system," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec 2017, pp. 4715–4720.
[5] A. Broad, I. Abraham, T. Murphey, and B. Argall, "Structured neural network dynamics for model-based control," *arXiv preprint arXiv:1808.01184*, 2018.
[6] N. Mohajerin, M. Mozifian, and S. Waslander, "Deep learning a quadrotor dynamic model for multi-step prediction," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 2454–2459.
[7] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.
[8] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 144–151.
[9] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5183–5189.
[10] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," *arXiv preprint arXiv:1811.08027*, 2018.
[11] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American statistical association*, vol. 74, no. 368, pp. 829–836, 1979.
[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.