Realtime and Onboard fault diagnosis of UAV motors using RNN prediction model

M. Boe, A.Y. Mersha, N. Alachiotis

[a]Saxion University of applied sciences, Ariensplein 1, Enschede, 7511 JX, , The Netherlands
[b]University of Twente, Drienerlolaan 5, Enschede, 7522 NB Enschede, , The Netherlands

Abstract

In recent years public interest in UAVs (unmanned aerial vehicles) has increased. Existing solutions are used to reduce risks and detect critical failures on system level or single output sensors. This research presents a solution for fault diagnosis of UAV motors using RNN prediction models. The work presented distinguishes itself from standard and existing solutions by not only performing fault diagnosis on a component level rather than using system models or analytical redundancies but also presenting on board, in real-time, physical experiments. The fault diagnosis accuracy is evaluated for several types of input functions including hand-flown flights and the fault diagnosis timing aspects are evaluated using the onboard processor. Results show on a static test setup, input data with high dynamic nature impose problems for the developed prediction model. The physical hand-flown experiments show that when the input data is generated by the PX4 flight controller, 100% detection rate is achieved. The developed prediction model runs at 3322Hz on the selected RPI4B and theoretical approaches are presented to calculate the response time of the system and calculate a theoretically maximum network size based on measurements.

Keywords: Fault-Diagnoses, Real-Time, Machine-learning

1. Introduction

In recent years Unmanned Aerial Vehicles (UAVs) technologies have advanced to such a degree that companies and universities are researching the possibility of UAV integrating technologies into our day-to-day life[1]. Numbers show that in 2020, the number of registered UAVs, also known as drones, already counts a total of 2.851 million in the US alone[2]. This shows that the public interest for using drones is increasing. The usage and integration of drones are hindered due to restrictions and regulations. For example, Locally in the Netherlands, a drone must be remotely piloted and stay within visual line-of-sight. Moreover, the drones may only be flown within designated areas, and usage of autonomous systems is limited because a pilot needs to be present during flight operations. Such restrictions limit the vast use potential of UAVs in terms of speed, safety, limited workspace, cost-effectiveness, and overall efficiency [3] [4][5].

On the 11th of June 2019, the European Union Law Commission published new EU-wide drone regulations that aim to make drone regulations among state members uniform and facilitate the position of the European drone market[6]. The new regulations relax restrictions that prohibited flights in populated areas and autonomous flights that go beyond visual line-of-sight. The usage of drones in civil applications has therefore become more feasible.

These regulations classify UAVs based on their weight and level of autonomy. These classifications still impose limitations on the usage of UAVs in or near urban areas to minimize risks. Currently, most drones are equipped with redundant sensors such as accelerometers, gyroscopes, and magnetometers[7] however, drones are not equipped with redundant actuators, speed controllers, or batteries. Failure of these components can lead to critical failure. New techniques help to overcome risks associated with the failure of sensors and actuators [8] [9]. Simple and standard measures have been used for the past few years to address priory known challenges, such as flyaway, RC communication loss, critical battery level [10][11][12]. The application of these techniques and measure help to reduce the risk when using drones however unforeseen circumstances can cause a drone to malfunction during flight and become a falling object.

According to a study[13], there is a direct translation between kinetic energy and levels of injuries. According to this study, we can assume that at 19.8J, a crossover is reached at which skull fractures occur, and above 99J, fatal injuries can occur. From calculations, we analyse that a drone weighing 1.9KG in a free fall crosses the skull fracture threshold after only 0.4 seconds and becomes a fatal impact after only 1.2 seconds of falling. For example, a parachute, giving enough time for a malfunction to be detected, the parachute to be deployed, and the terminal velocity to be reached, can reduce impact energy to such levels that the impact would no longer cause skull fractures.

The detection of a failure in actuators and propellers, according to this study[14] is done by measuring the system dynamics as a whole. Fault isolation can be performed by inversion of the dynamic model. This limits a system

into a static configuration and needs to be modelled by a modelling expert. This is a limitation in the current technological era, where rapid prototyping and Agile workflows are the new standard.

During this research, work is done to provide a potential solution towards a system capable of fault diagnosis on component level rather than system level that does not require modelling expertise mentioned earlier. A component-based off-the-shelve solution would maintain the benefit of rapid prototyping whilst maintaining safety guarantees. The proposed solution uses a recurrent neural network based solution to create a prediction model that predicts the velocity, power consumption behaviour of the motor and temperature of the ESC based on historical data and the throttle set point. A significant deviation between the predicted and measured response indicates faulty behaviour.

During this research the following research question will be answered:

- To what extend can neural network based prediction models be used to reliably perform real-time fault diagnosis of UAV actuators on component level on board and in real-time?

In order to answer this main research question, the following sub questions are formulated:

- How accurate does a prediction model have be met to preform fault diagnosis reliably?
- What timing constraints have to be satisfied to meet the required reliability?
- Can a processor based solution be used to meet the timing and reliability constraints?

In order to answer the research questions proposed above the following steps are taken that contribute to the final results

- A test setup was made to create training data sets in a static situation
- Various network architectures are trained and compared, and the prediction behaviour and its error is analysed with propellers in various conditions of damaged and undamaged.
- The timing statistics of the prediction model are analysed and a processing platform is chosen to use the prediction model onboard and in real-time
- The system was built and mounted on a UAV, and test flights were performed with propellers in various conditions, and the results have been analysed.

Section 2 describes the background of this project, Section 3 describes the test setup, data collection process and different network architectures, Section 4 describes the evaluation methods used to answer the research questions, Section 5 Evaluates the results obtained during testing, in Section 6 we concluded on the research and evaluate the research questions and recommendations for future work are given.

2. Background

This Section will describe the theoretical and mathematical background required for this research. In Subsection 2.1 the motivation that led to this research is described, Subsection 2.2 describes the related work and gap this thesis is going to fill, Subsection 2.3 gives a brief introduction into the mathematics of the used neural network architectures, and in Subsection 2.4 the feasibility of component-based fault diagnosis is explored based on ESC measurement data.

2.1. Motivation

UAV Motors and propellers inherently have a high dynamic behaviour that depends on many external disturbances like, for example, airflow, angle and forces on the rest of the UAV. Because of this, fault diagnosis is usually done using model prediction, or analytical redundancy [14]. Existing literature focuses on finding and isolating faults on system level, which means that for a given UAV, system models are created[15][16][17]. In this research, we will focus on fault diagnosis at component level. The theoretical benefit of fault diagnosis at component level is that the system will be configurable at component level, and no further work on fault diagnosis needs to be performed.

Model prediction methods that have been proposed[18], [19], [20], [21], [22], predict the dynamic output of a system and classify correct or faulty behaviour based on a relative threshold. From the comparison results presented in Section 3, a Recurrent Neural network(RNN) solution provides the best results as a prediction model for the given motor and propeller setup.

2.2. Related work

This subsection summarises related work and looks at their result. From this, a conclusion is drawn about the current state of the art. Table 1 provides an overview of the related work and shows what gaps the proposed work will fill.

A survey has shown that prediction models offer the most promising results over Analytical redundancy and signal processing [14]. Therefore, we will focus on several prediction model methods in this section.

Zhang et al. 2020[22] propose a Kalman filter(KF) based solution to do model prediction on component level. The paper provides a set of state transition equations describing a quadrotor UAV's behaviour. The results

Table 1: Contributions of related work

Paper	Model prediction method	fault diagnosis abstraction level	Input dimensions	Onboard	Realtime	Experimental validation
Zhang et al. 2020	KF	System level	2D	n	n	y
Wang et al. 2019	LSTM	Component level	1D	n	n	y
Fu et al. 2019	CNN-LSTM hybid	System level	2D	n	n	y
Liu et al. 2018	SVM	Component level	1D	n	y	y
Avram et al. 2017	non-linear adaptive estimator	System level	2D	n	n	n
Proposed work	RNN	Component level	2D	y	y	y

of the KF-based system are tested against actual sensor measurements. Fault diagnoses of individual components are done based on system inversion. The results of the paper promise good function bases on simulation but does not provide actual statistics.

Wang et al. 2019[18] propose an LSTM solution to do model prediction on component level. The paper presents a system architecture and training framework to reason about fault diagnosis in sensor data. Results in the paper show a near-perfect fault diagnosis in a sample set of about 1000 samples. The input to the system is a 1D input stream over time, and the results are based on offline real flight data.
Fu et al. 2019[19] propose a hybrid CNN-LSTM solution to do model prediction on system level. The paper gives a structure used to combine the RNN and LSTM techniques. The paper describes the experimental setup where one of the propellers is cut to provide less torque in one of the data sets. The results of this paper show an average of 92.74% accuracy based on 11 element input stream over time based on offline real flight data.

Liu et al. 2018[20] propose a State vector machine(SVM) to do model prediction on component level. The paper gives an architecture and instruction set implemented and tested on an FPGA. Results in this paper show a false positive rating of 5.88% and a false negative rating of 2.2%. The implementation on FPGA shows a significant speed-up of 2.6 and a quantization error in the order of 10^{-5}. The input to the system is a 1D input stream over time, and the results are based on offline real flight data.

Avram et al. 2017[21] propose a non-linear adaptive estimator to do model prediction on system level. A set of dynamic equations are given provided to model a quad-rotor. The control signals and sensor measurements are fed to the estimator, and a decision schema are used to do fault diagnose and to estimate the magnitude of fault. The results show improving results as the magnitude of faults increases. The results are based on real-time and online CPU based computation.

Finally a overview in Table 1 of the related work discussed in Section 2.2. The Table classifies for each paper what model prediction method is used, the fault diagnosis abstraction level, whether the input dimension is single-(1D) or multi-dimensional(2D) , whether the fault diagnosis was performed onboard on physical hardware, the fault diagnosis was performed in real-time and finally whether an experiment was done to validate the proposal. From the third column, red and green colours indicate whether the proposal meets the requirement that started this research.

From the Table and summaries of each contribution, we can conclude that existing contributions can not be directly used to do hardware implementation of fault diagnosis of UAV actuators on component level. The proposed work in this research can be seen to cover all those criteria in the last row.

2.3. Neural networks

To understand the concept of RNN, we first have to understand Artificial Neural networks, that are also called Neural networks(NN)[23][24]. A NN is an artificial representation of biological neurons and is used to solve problems in various fields that are non-trivial to solve with more classical approaches. A biological neuron is imitated in a single so called node. This node has various amounts of inputs that all get multiplied by a weight. All the multiplied inputs are passed through a transfer function whose output is passed through a non-linear activation function. The transfer function usually is a sum function of all the inputs. A training algorithm determines the bias and the weights for each node. The output of this activation function, in turn, is the node's output. A visual representation of a single NN node can be seen in Figure 1.

The chosen activation function is the SELU activation function described by Equation 1. The SELU activation function has self normalizing properties which work well in the presence of noise and perturbations compared to other activation functions.[25]:

$$\sigma(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{1}$$

Where $\alpha = 1.67326324235437728481704299167 17$ and $\lambda = 1.05070098735548049341933498529 46$.

A NN with three layers where each layer has a size of at least three nodes, as shown in Figure 2 can be used to

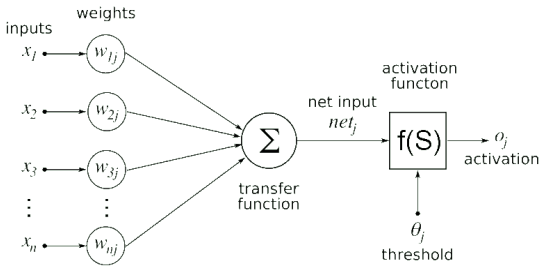


Figure 1: A single node of a neural network (Adapted from [26].)

model any non-linear second order system[23]. When the amount of layers and the size of each layer is increased, a NN can be used to model higher order systems and include dynamic behaviour. As classical NN have no memory, previous inputs and elapsed time are not considered when calculating the output.

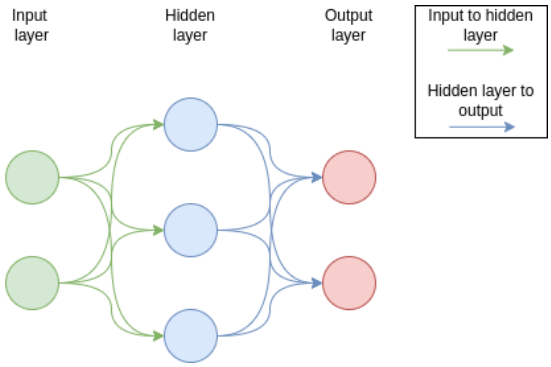


Figure 2: A neural network with an input layer, one hidden layer and one output layers (Adapted from [27].)

As described above, we need to take previous inputs into account. We can do this by either using previous inputs such as $x_t, x_{t-1}...x_{t-m}$ for a history of $m+1$ values and the *th* value. Alternatively, create a RNN that includes intermediate feedback where the input to either the input layer or one of the hidden layers includes the output from the previous iteration. A sample RNN network can be seen in Figure 3. The benefit of doing this is that we automatically incorporate state information as input[28]. A downside of using RNN is that a feedback loop can result in saturation or exploding gradients during training and, at some points, even cause overflow issues in hardware [29].

Another flaw of RNN, as opposed to NN, is that back-propagation through the network used to take time or previous states into consideration also feeds back error signals[31]. Error signals flowing backwards in time tend to explode or vanish, resulting in unstable systems. LSTM networks, a special kind of RNN, can be used to solve this vanishing/exploding gradient problem. An LSTM cell is shown in Figure 4 and its structure can be described by the following set of equations[19]:

$$f_t = \sigma(W_{xf}x_t + W_{sf}s_{t-1} + b_f) \quad (2)$$

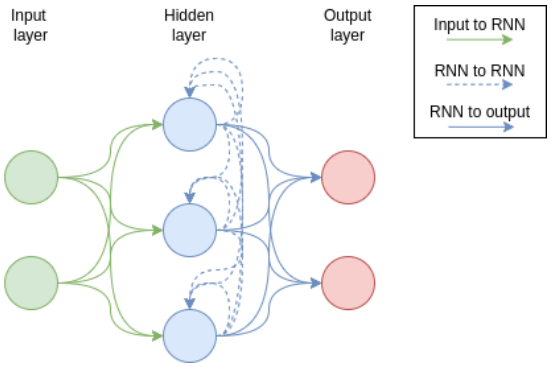


Figure 3: Recurrent neural network illustration where the hidden layer is recurrent with itself (Adapted from [30].)

$$i_t = \sigma(W_{xi}x_t + W_{si}s_{t-1} + b_i) \quad (3)$$

$$c_t^* = tanh(W_{xc}x_t + W_{sc}s_{t-1} + b_c) \quad (4)$$

$$o_t = \sigma(W_{xo}x_t + W_{so}s_{t-1} + b_o) \quad (5)$$

$$c_t = f_t \circledast c_{t-1} + i_t \circledast c_t^* \quad (6)$$

$$s_t = o_t \circledast thanh(c_t) \quad (7)$$

where the subscript t indexes the timestamp, i_t, f_t and o_t are input gate, forget gate and output gate. The σ denotes the activation function, and $\circledast$ denotes the convolution operator. W is the set of weights, and b is the bias. A sample LSTM cell can be seen in Figure 5.

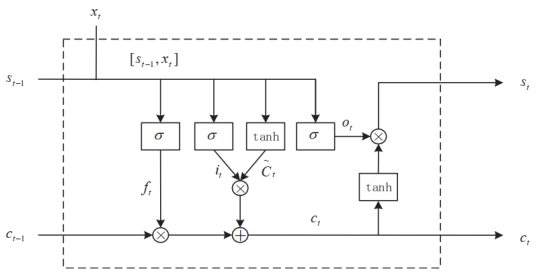


Figure 4: LSTM cell architecture (Adapted from [19].)

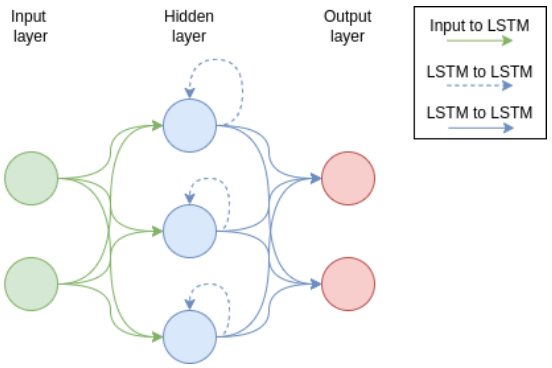


Figure 5: LSTM neural network illustrated where the LSTM cells are recurrent with them self.

2.4. Qualitative data analysis

To verify the feasibility of performing fault diagnosis, we first examine the output of the available data. The Myxa CAN ESCs described in Section 4.1 provide details about the status of the ESC at 100 Hz. The information on the CAN bus is available using the UAVCAN V1 standard in the format[32] sportance of feen in Listing 1:

Listing 1: UAVCAN status message format

```
uint32 error_count
float16 voltage
float16 current
float16 temperature
int18 rpm
uint7 power_rating_pct
uint5 esc_index
```

From this data, we can reason about the performance of the motor by monitoring the RPM, voltage and current. Any significant deviation in the rotational speed of the motor or power consumption can indicate faulty functioning. To verify this hypothesis, 5 sets of 10 tests are executed for a combined of 50 tests. Each set of tests is performed with a different propeller. Three different undamaged propellers (labelled T, S1 and S2) are used as a control group. One slightly chipped propeller (labelled D1) and one modified propeller (labelled D4), which is shortened by 5 millimetres on both sides, are used. All 5 propellers can be seen in Figure 6.

An input pattern is generated a priory and used for all tests. The input pattern is a time series of the function that is a random combination of constant, ramp and step functions, see Figure 7.

After a series of 50 tests, we can plot the RPM of the motor and power consumption of the ESC over time, see Figures 8 and 9 on page 6 respectively. From these graphs, an immediate observation can be made that some of the tests have significant outliers during the first stage. However, after frictions have been overcome and the controller has stabilised, the signals all follow the same relative pattern.

When zooming in on a smaller part of the graphs see Figures 10 and 11, it becomes clear that the deviations that were slightly visible in the larger graphs seen in Figures 8 and 9 on page 6 actually contain distinguishable differences. From the zoomed-in plots, the observation can be made that the control group propellers have a high power consumption and a lower RPM compared to the damaged propellers. It can also be observed that as the propeller becomes more damaged, the RPM increases and the power consumption decreases, but the motion profile is maintained.

A prediction model capable of predicting the behaviour of the control group good enough can use data outside of a range around its prediction and compare it to a measurement to execute fault diagnosis.

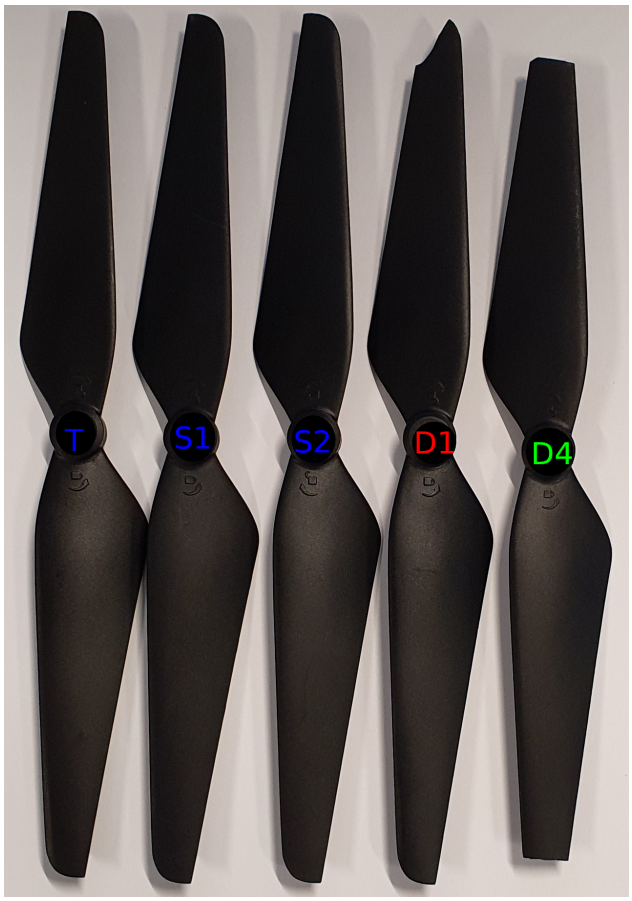


Figure 6: propellers used during data collection

3. Prediction Model

In this Section, the process of developing the prediction model is described. Firstly in Subsection 3.1 the different architectures are proposed, in Subsection 3.2 explains how some parameters are set to values limiting the solution space, in subsection 3.3 it is explained how the training data is gathered and labelled and finally in Subsection 3.4 it is explained how the final architecture is chosen.

3.1. Architectures

For the prediction model, three types of architectures are considered. A classical Feed Forward Neural network see Subsubsection 3.1.1, A Recurrent Neural Network see Subsubsection 3.1.2 and a LSTM Recurrent Neural network see Subsubsection 3.1.3. GRU[33] or Transformer[34] type networks are not considered to reduce the solution space.

The available input data for the prediction model are the motor speed set-point and historical measurement data. The input value X_t at a specific step in time is defined as seen in Equation 8 where dt_t is the time difference between the current and the previous measurement, rpm_t the RPM measurement of the motor, v_t the voltage measurement of

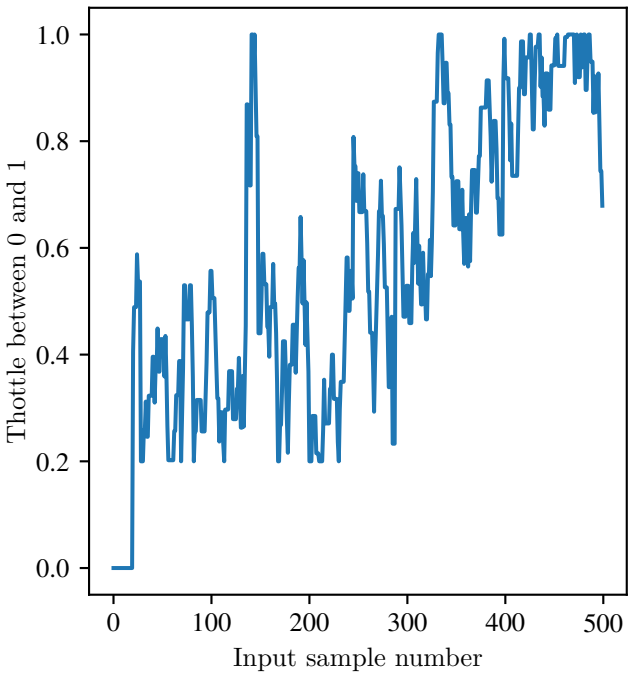


Figure 7: 50 tests input pattern

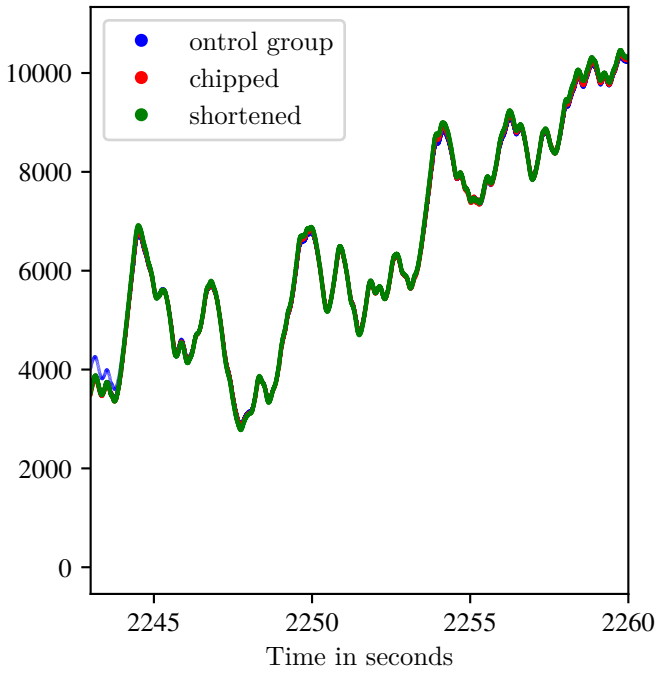


Figure 8: RPM graph

the ESC, c_t the current measurement of the motor and finally $temp_t$, the temperature measurement of the ESC.

$$X(t) = \begin{pmatrix} dt_t & rpm_t & v_t & c_t & temp_t \end{pmatrix} \tag{8}$$

For all architectures, the network is going to predict the output $\hat{Y}_t$, see Equation 9.

$$\hat{Y}_t = \begin{pmatrix} rpm_t & v_t & c_t & temp_t \end{pmatrix} \tag{9}$$

All architectures will map to some function PM, see Equation 10 where $input(t)$ can be different based on the architecture, this will be explained in each respective Subsubsection.

$$\hat{Y}_t = PM(input(t)) \tag{10}$$

3.1.1. NN

The first to consider architecture will be a classical feed forward neural network. The input is defined as seen in Equation 11 where u_t is the motor throttle set-point, X_t is as described in Equation 8 and $nHist$ the number of historical inputs used.

$$input(t) = \begin{pmatrix} u_t & | & X_{t-1} & | & X_{t-2} & | & ... & | & X_{t-nHist-1} \end{pmatrix} \tag{11}$$

The input size of the network depends on the value of $nHist$ and can be calculated using $(nHist * 5) + 1$. The number of hidden layers and the size of those layers will

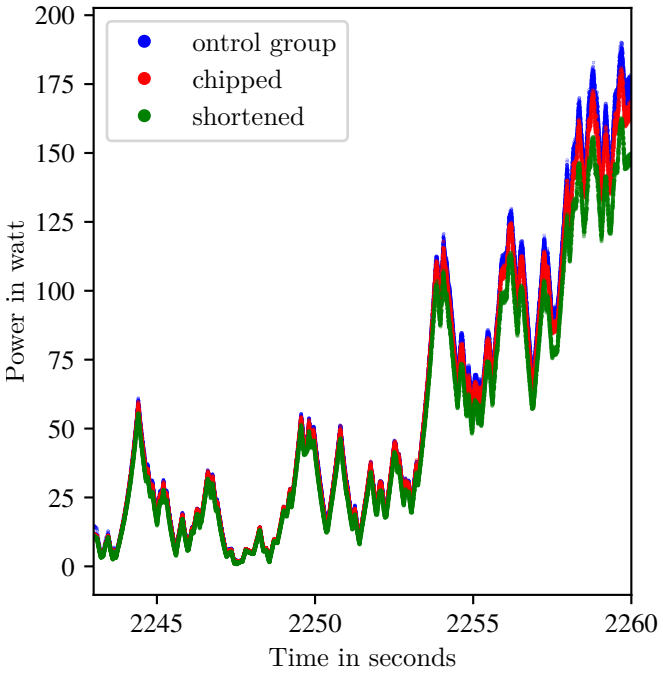


Figure 9: Power graph

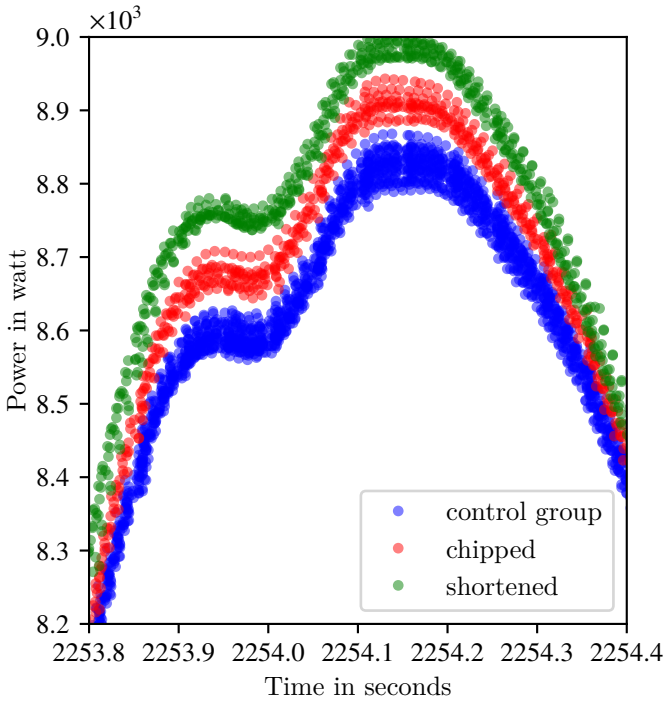


Figure 10: RPM graph zoomed in

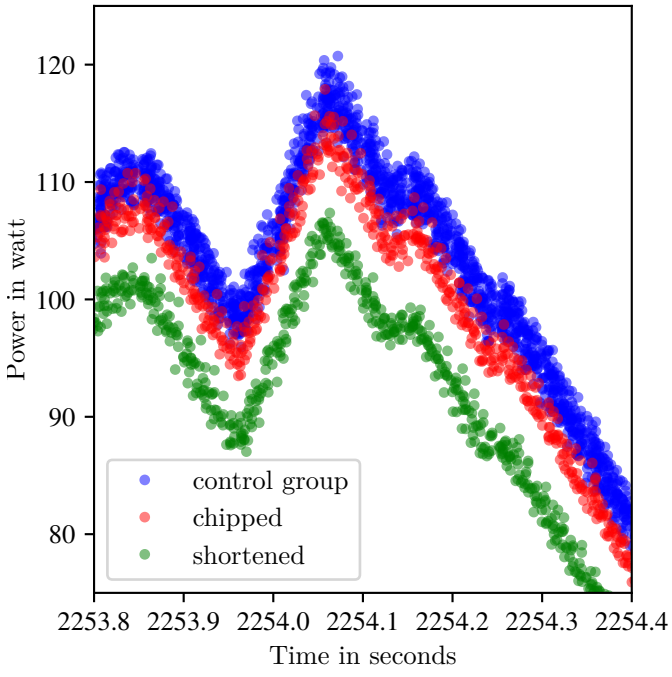


Figure 11: Power graph zoomed in

depend on the training process. This will be further explained in Subsection 3.3. A visualisation of the network architecture can be seen in Figure 12.

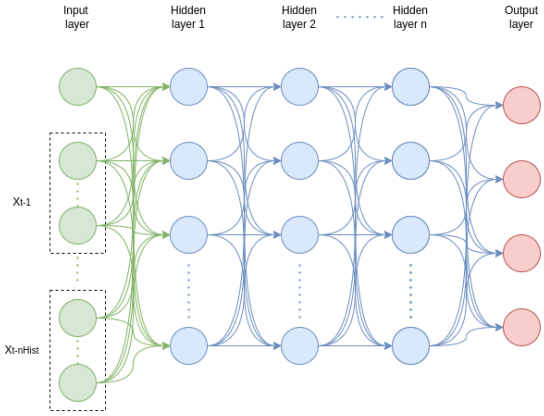


Figure 12: A feed forward architecture that takes previous inputs into account.

3.1.2. RNN

The second to consider architecture is the Recurrent Neural Network. The input is defined as seen in Equation 12 where the variable are similar to the ones presented in Subsubsection 3.1.1. When using recurrent neural networks, the input shape varies as opposed to the one used for the feed forward neural networks. In this case, each row in the matrix will be a separate iteration of calculations where the recurrent feedback will substitute as state feedback. The initial values rnn_{t0}^{out} will be set to 0 on each iteration.

$$input(t) = \begin{pmatrix} u_t & | & X_{t-1} \\ u_{t-1} & | & X_{t-2} \\ \dots & | & \dots \\ u_{t-nHist} & | & X_{t-nHist-1} \end{pmatrix} \tag{12}$$

The input size of the network is always 6 because of the number of measurable inputs, and the number of sample iterations depends on the value of *nHist*. The number of hidden layers and the size of those layers will depend on the training process, as explained in Subsection 3.3. A visualisation of the network architecture can be seen in Figure 13.

3.1.3. LSTM

The third and final considered architecture is the LSTM neural network. The input, initial states, input sizes, amount of hidden layers and hidden layer sizes are defined similarly to the RNN architecture as seen in Equation 12 and explained in Subsubsection 3.1.2. A visualisation of the network architecture can be seen in Figure 14.

3.2. Limiting parameters

To reduce the solution space of different networks, some parameters will be set to specific values. Small trial

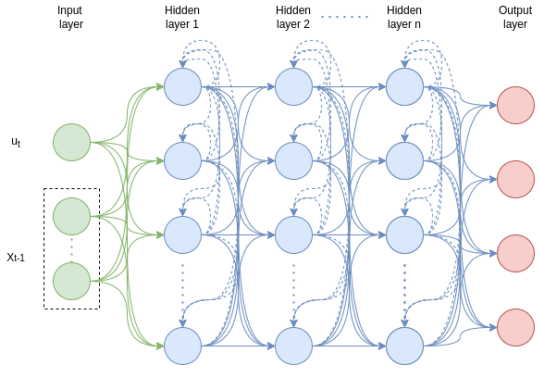


Figure 13: A recurrent network architecture that takes previous inputs into account.

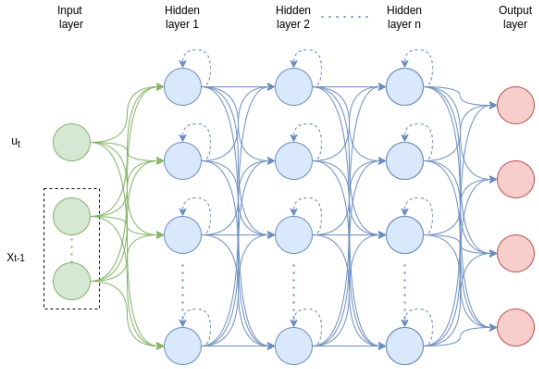


Figure 14: A LSTM network architecture that takes previous inputs into account.

on qualitative error processes obtain the chosen values. However, only the final results are shown.

The weight vector $w = \begin{pmatrix} 0.3 & 0.3 & 0.3 & 0.1 \end{pmatrix}$ for rpm, voltage, current and temperature respectively. During training, it was observed that the RPM prediction was matched reasonably quickly with the RPM measurements, but the power profile was not. It was also observed that the deviations of temperatures are extremely small and no notable local differences can be found. The assumption can be made that only significant differences (order of magnitude or larger) may reason about the functioning of the ESC. That's why emphasis was given to the power profile prediction, which is the product of voltage and current.

The number of historical measurement points $nHist = 5$ is used because it was observed that a low value of historical measurement points, 2 to 3, already produces satisfactory results. The value of 5 was chosen to have a safe value with some margin for error in the estimate.

During training the batch size $bs = 128$, the maximum number of Epochs $Epoch_{max} = 3000$, the initial learning rate $lr_{initial} = 0.001$ and the learning rate decay rate $lr_k = 0.001$.

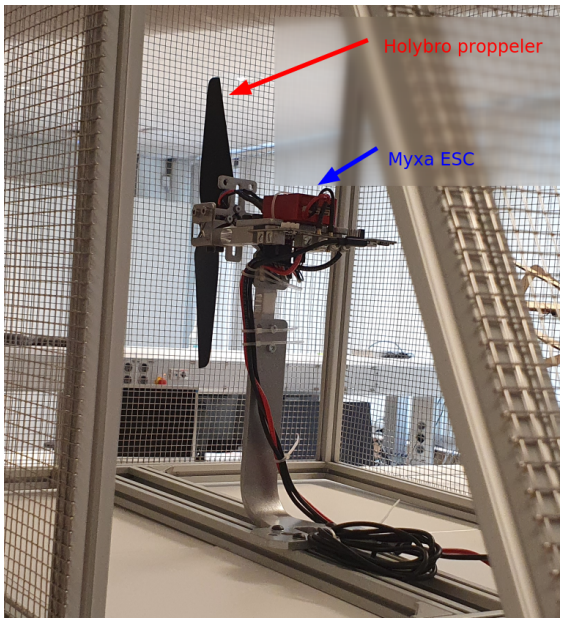


Figure 15: Caged test setup used to collect measurement data.

The inputs used as motor set-points u_t are bound between 0.2 and 0.8 duty cycle during training and validation of the network. During the evaluation of the transfer-ability in Subsection 5.4, motor inputs are generated by the flight controller and thus not limited.

3.3. Gathering training data

A static test setup was created and used to generate the data set for training and evaluation purposes. The test setup is a mechanical arm that mounts the motor and ESC in a cage to use the motor in a safe and controlled manner. The mechanical structure is similar to what one might find on a UAV limiting airflow obstructions. Figure 15 shows the test setup with the Myxa ESC and Holybro propeller, more information on the hardware can be found in Subsection 4.1.

To generate a sample set that is going to be turned into a data-set, a set-point profile is generated with 136,000 inputs which translate to 22 minutes and 40 seconds of inputs at 100 Hz. The input file is a collection of constant inputs (see Figure 16 on page 9), ramp inputs (see Figure 17 on page 9) and step inputs (see Figure 18 on page 9). As explained in Subsection 3.2, the inputs are bound between 0.2 and 0.8.

Via the CAN adapter (see Subsection 4.1), the motor set-points are given, and the motor status is sampled. All this data is logged in a text file and later post-processed. As the real-time data samples are collected in order, a data-set can easily be generated and labelled as the output of the network should is simply the subsequent measurement. This means that a data collection of N samples can provide a data-set of $N - nHist - 1$ training inputs.

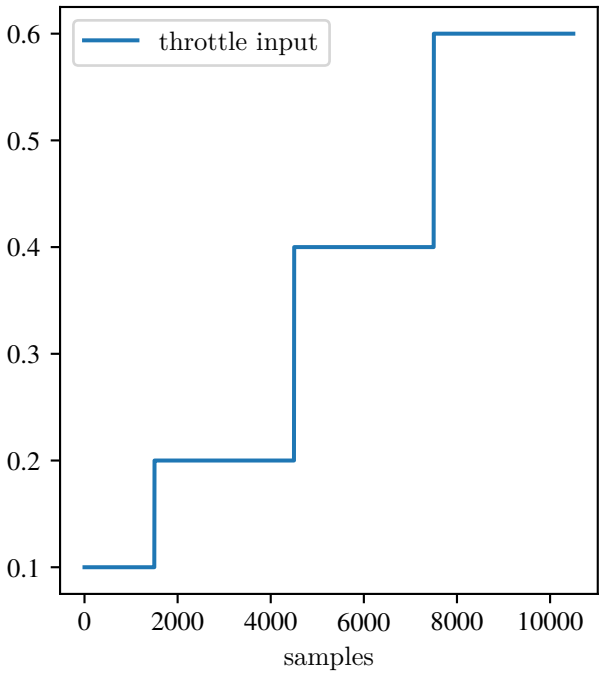


Figure 16: Constant input data example

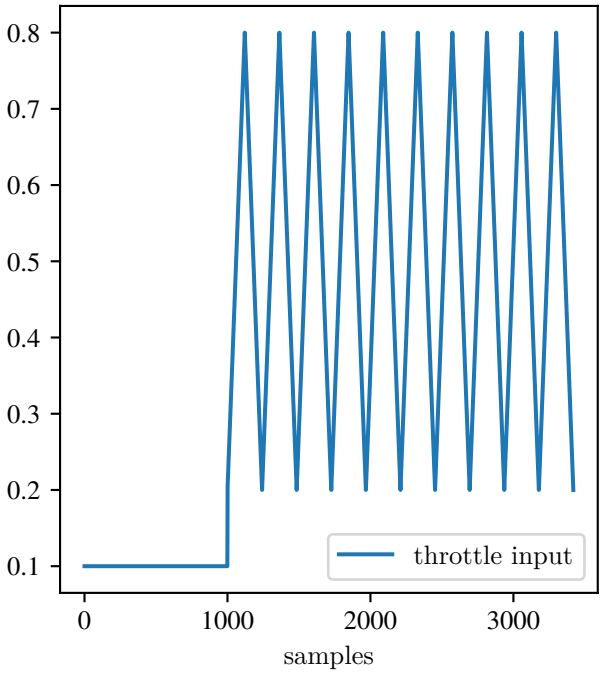


Figure 17: Ramp input data example

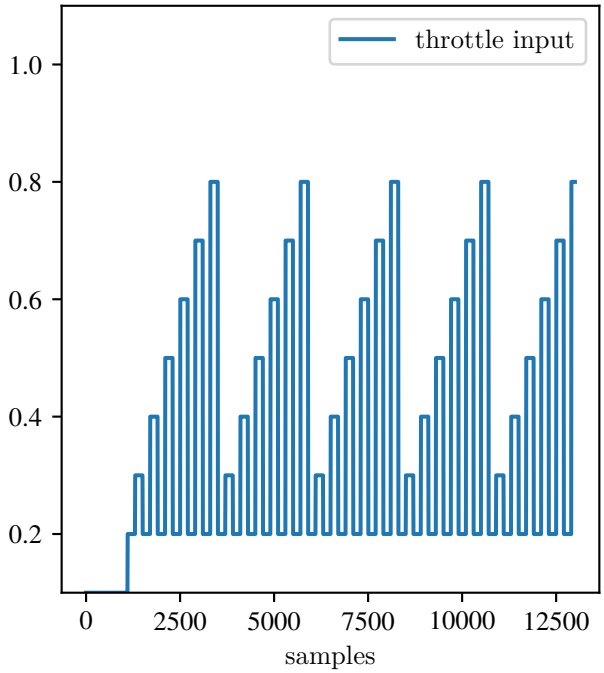


Figure 18: Step input data example

After Labeling, the data set is shuffled to reduce the time continuity and increase variety within training batches.

3.4. Training the different architectures

During training, it became clear that TimeSeriesForecaster from AutoKeras[35] which would be the go to automatic training framework, was not suitable. The TimeSeriesForecaster feature had issues at the time of this research being worked on. To circumvent these issues, a similar manual approximation was performed. The following steps take the manual process and the results of this process can be seen in Subsection 5.1:

1. Start with guaranteed overestimate of the network and train
2. Take away an entire hidden layer and train again
3. If results perform similar or better repeat last step, if only 1 hidden layer is left continue to the next step.
4. Reduce the number of cells of all hidden layers (16 cells were taken away every time) and train again
5. If results perform similar or better repeat last step, otherwise continue
6. If the number of cells in each layer was reduced, add an extra hidden layer with the same amount of cells as the other hidden layers and train again
7. If results perform similar or better repeat last step, otherwise stop

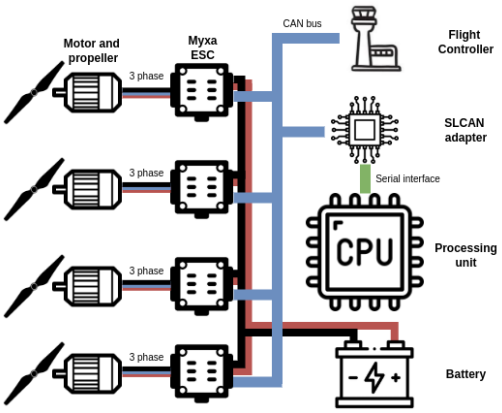


Figure 19: System diagram

It is noted that a larger network always has the capability of performing at least similar to any smaller network. However, with a given limited training time, results may not converge as well[36].

4. Methods

In this Section, the methods used to evaluate the results will be described. The evaluation method for training is explained in Subsection 4.2, the evaluation of the model accuracy is described in subsection 4.3, the evaluation method of timing characteristics is described in 4.4 and in Subsection 4.5 the evaluation method of transferability is described.

4.1. Equipment

The used equipment is split up into two separate setups. The first setup is the testing setup that was already shown in Figure 15 and the second setup is an UAV with a X500 v1 frame from holybro[37] with 4 2216 KV880 Motors from holybro, 4 Myxa model A2 ESCs[38] with UAVCAN support, a Pixhawk 4 [39] as flight controller, and a CAN adapter[40] to interface with the CAN bus, a Raspberry Pi Model 4B[41] as processing unit. A system diagram can be found in Figure 19 and the UAV can be seen in Figure 20.

During data gathering and evaluation of the training process in Subsection 5.1, accuracy in Subsection 5.2, and timing in Subsection 5.3 the testing setup is used with only a single motor. The UAV will be used to evaluate the transfer ability in Subsection 5.4.

4.2. Training

To evaluate the neural network's performance and compare the different types of network architectures, the Keras Neural network framework is used with Tensorflow. A weighted euclidean distance cost function is used. Equation 13 shows the used cost function where y is the measured output, $\hat{y}$ is the predicted output, and w is the weight vector as explained in Subsection 3.2.

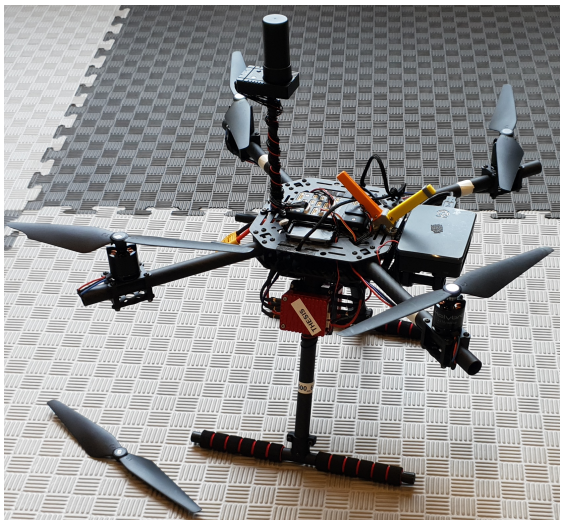


Figure 20: The UAV used for testing

$$D(y,\hat{y}) = \sqrt{\sum_{i=1}^{N} w(y-\hat{y})^2} \tag{13}$$

For training, the Adam optimizer[42] is used with the default tuning values and an exponentially decaying learning rate that corresponds with the current epoch. See Equation 14 where i is the current epoch, $lr_{initial}$ is the initial learning, and lr_k is the decay rate as previously explained in Subsection 3.2.

$$lr(i) = lr_{initial} e^{-lr_k * i} \tag{14}$$

To evaluate the different networks, we first compare the lowest achieved loss of all networks in the same network architecture and then compare the best network of each architecture to the other architectures.

4.3. Accuracy

To evaluate the accuracy of the Prediction model and to use a set of 7 different propellers is used, three propellers as a control group and four different types of damage. The propellers can be seen in Figure 21 on page 11.

The individual tests are performed for each propeller where the inputs are:

- A set of constant inputs at a slow 0.2 speed and fast 0.7 speeds
- A set of ramp inputs at different ramping rates between 0.2 and 0.8
- A set of step functions at different varieties of differences between 0.2 and 0.8

Figure 21: The props used for validation

The output Y is recorded similarly to during network training, and the best-trained model is used to compute the predicted output $\hat{Y}$. To compare the accuracy of the different propellers we are interested in The maximum error value, the mean and the standard deviation to reason about performance. As explained in Subsection 3.2, the attributes that will hold the most information are the power and RPM measurements. Because of this we will compare the normalized error for Power E^p, see Equation 16, and the normalized error for RPM E^{rpm}, see Equation 15.

$$E^{rpm} = \left|\frac{\hat{Y^{rpm}} - Y^{rpm}}{\hat{Y^{rpm}}}\right| \tag{15}$$

$$E^{p} = \left|\frac{\hat{Y^{v}}\hat{Y^{c}} - Y^{v}Y^{c}}{\hat{Y^{v}}\hat{Y^{c}}}\right| \tag{16}$$

$$E^{temp} = \left|\frac{\hat{Y^{temp}} - Y^{temp}}{\hat{Y^{temp}}}\right| \tag{17}$$

4.4. Timing

To evaluate the timing characteristics of the prediction model, we can let the CPU measure the execution time of the prediction model process. With the current equipment, see Subsection 4.1, the maximum measurement sample rate is 100 Hz. A lower processing frequency could still be used to perform fault diagnosis but would reduce the response time.
To get a realistic time measurement, the start time will be recorded, and then the prediction model will be processed 100,000 times, after which the end time will be recorded. By doing this, a more realistic time measurement is taken, and the time each iteration takes can simply be calculated by dividing the total elapsed time by 100,000. This test will be repeated three times for validity. The time measurement algorithm can be seen in Algorithm 1.

Algorithm 1: Time measurement algorithm

start time t^s
for $m \in \{1, \ldots, 10,000\}$ do
  $\hat{y} := PM(input(i))$
end for
end time t^e
iteration time $t^{pm} := \frac{t^e - t^s}{10,000}$

As the timing results depend on the processing unit and methodology of processing, the target device and method of processing will be determined during evaluation in Subsection 5.3.
The response time t^r of the Prediction model will depend on the processing time of the network t^{pm}, the delay caused by potential required filtering and the accuracy of the network. The response time can be calculated by multiplying the response time and the delaying factor caused by filtering d^{filter} see Equation 18 where t^{pm} is the processing time of the network as explained in Subsection 4.4. The average expected detection time of the system would depend on the accuracy and response time of the Prediction model.

$$t^r = t^{pm} d^{filter} \tag{18}$$

During this evaluation, the timing of the prediction model will be analysed, and timing aspects for data acquisition will not be taken into account.
Finally, To answer the research question related to timing,"What timing constraints have to be satisfied to meet the required reliability?", an evaluation will be performed about what would happen during a crisis and if the X500 V1 particularly would fall from a great height. During this evaluation, the impact energy mentioned earlier in Section 1 is estimated using a simple falling object model and a response strategy using a parachute is explored.

4.5. Transfer-ability

Finally, this research aims to do fault diagnosis onboard and in real-time. The final evaluation performed during this research is aimed at how well the prediction model performs when transferred from the test setup that was used to generate the training data sets to the UAV described in Subsection 4.1. The accuracy of fault diagnosis will be evaluated during flight in which several short hand flown flights will be flown using the same propellers used in Subsection 4.3, see Figure 21. Because these tests will be hand flown, the inputs generated by the flight controller will not be repeatable for each test.

5. Evaluation

In this section, the results of this research will be shown. Subsection 5.1 shows the results of the different architectures produced during the training process, Subsection 5.2

n_l/n_n	NN	RNN	LSTM
6/128	1.6621	1.2252	1.352
4/128	1.6237	1.1627	1.321
2/128	1.5934	0.9413	1.232
1/128	1.3284	0.7316	0.9424
1/102	1.0123	0.6899	0.8782
1/96	0.9672	0.6894	0.8186
2/96	1.2527	-	-
1/72	1.0032	0.6656	0.8466
2/72	-	0.6693	0.7558
3/72	-	-	0.8342
1/48	-	0.7287	0.8224

Table 2: Table of final validation losses for different network archetypes

goes into detail about the prediction accuracy of the Prediction model, Subsection 5.3 shows the results of a timing analysis of the Prediction model, finally in Subsection 5.4 shows test results of UAV flight tests and reasons about the applicability of using the prediction model on UAVs.

5.1. Training

Table 2 shows the best evaluation loss values for each type of network architecture in their different configurations using the training procedure described in Subsection 3.4 and the evaluation method described in Subsection 4.2. The first column shows the configuration n_l/n_n where n_l is the number of hidden layers in the network, and n_n is the number of nodes in each layer.

From the results, we can read that configuration 1/96 provides the best results for the Feed forward network architecture described in Subsubsection 3.1.1. Configuration 1/72 provides the best results for the RNN network architecture described in Subsubsection 3.1.2. Configuration 2/72 provides the best results for the LSTM network architecture described in Subsubsection 3.1.3. Between the three proposed network architectures. The RNN network architecture with configuration 1/72 configuration results in the overall best result and will therefore be implemented as the Prediction model when evaluating the Accuracy in Subsection 5.2, the timing statistics in Subsection 5.3 and the transfer-ability in Subsection 5.4.

5.2. Accuracy

In the testing process the input patterns described in Subsection 4.3 are tested and recorded for every propeller shown in Figure 21.

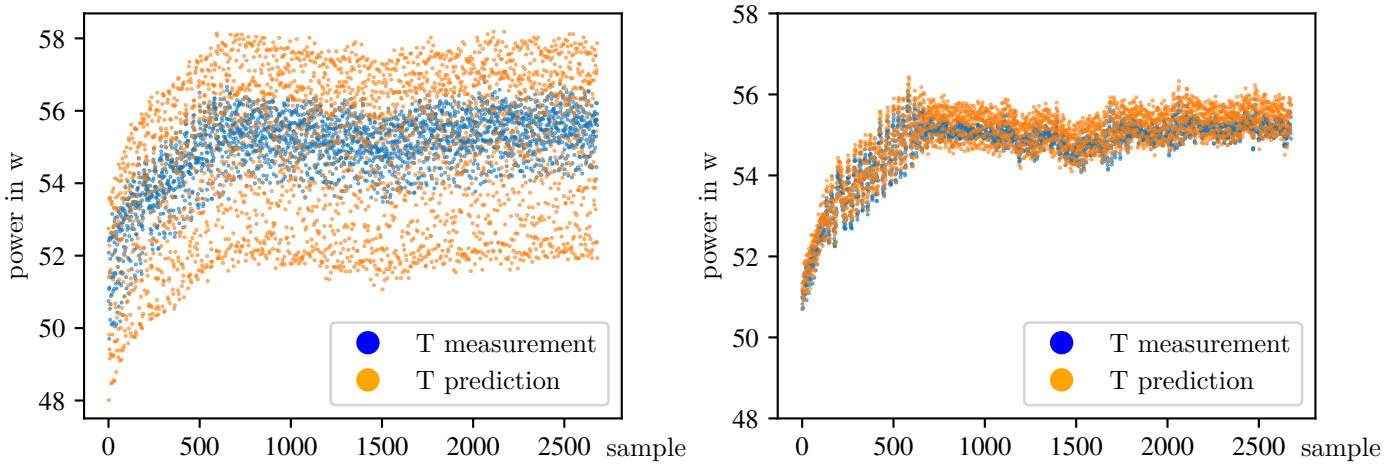
During this accuracy testing phase, it was discovered that the measurement data includes high-frequency noise. This high-frequency noise was captured in the training data, and the prediction model has learned about this noise. However, this noise is amplified because the prediction model contains a feedback loop, as earlier explained in Subsection 2.3. The noise and its amplification can be observed in Figure 22. To mitigate the noise, a moving average filter was used as described in Equation 19 where u_t is the input at time t and n_{inpf} is the size of the average filter. Based on observations the size of $n_{inpf} = 10$ was chosen, and the results after filtering can also be observed in Figure 22 on page 22.

$$u_t = \frac{1}{n_{inpf}} \sum_{i=0}^{n_{inpf}} u_{t-i} \tag{19}$$

Introducing the average filter on the input data means that a new dataset had to be collected, and the network had to be retrained. Due to time constraints, the same 1/72 RNN network was used without taking the steps explained in Subsection 3.4.

The measurement results for the constant inputs can be seen in Table 4, The measurement results for the ramp inputs can be seen in Table 5 and the measurement results for the step inputs can be seen in Table 6, all tables can be found on page 17. In the tables, reasonable deviations from the undamaged propeller labelled "T" are coloured in a light red colour, and significant deviations are coloured in a darker shade of red. From these, it can be read that in all measurements, the maximum values, mean and standard deviation of the more significantly damaged propellers D2 and D3 exceeds that of the undamaged propeller. In several cases, the mean of damaged propellers exceeds that of the maximum of undamaged propellers. From this, a simple classification can be performed by thresholding. In other cases, further research needs to be done into classification approaches or how damaged a propeller needs to be even to constitute action.

To get a less abstract grasp of the normalized error data, all sets of data can be seen in Figure 25 on page 18 where the Y Axis is the normalized error and the X axis are the samples. These figures show nine separate plots, where each Row corresponds to RPM, Power or Temperature measurements, and each column represents the output for Static, Ramp or Step inputs. From These plots we can conclude that even though Tables 4, 5 and 6 suggest a clear distinction between the Undamaged propeller and damaged propeller can be made, this is not the case. One more thing to note is that there is no linear correlation between the normalized error and the speed at which the motor turns. For example, look at the first column of plots, the propeller labelled D2 has significantly larger errors compared to the other propellers. It can be concluded that this higher deviation must be a result of the unbalanced centre of mass introduced by the damage only on one side.



(a) unfiltered

(b) average filter with size 10

Figure 22: Power measurement and prediction for constant input

5.3. Timing

As described in Subsection 4.4, a target processing device was chosen during the timing evaluation. An implementation of the RNN prediction model was created in the language C, see Algorithm 2. This algorithm uses steps to calculate the output; during the initialization, the weights are loaded where w^i are the input kernel, w^r is the recurrent kernel, b^r is the bias for the hidden layer, w^o is the output kernel, and finally b^o is the output bias. All weight and bias variables used are stored in one dimensional arrays. This listing does not show the data sampling and assumes that the data is correctly put into a FIFO.

Algorithm 2: RNN algorithm.

load weights w^i, w^r, b^r, w^o, b^o
repeat
  $X := X[1:]+x$ {shift in fifo data}
  $O^r := 0$ {reset initial recurrent values}
  for $i \in \{1, \ldots, nHist\}$ do
    for $j \in \{1, \ldots, nNodes\}$ do
      $O^h_j := \text{selu}(\sum_{k=0}^{nInputs} X_{i,j} w^i_{j,k} + \sum_{k=0}^{nNodes} O^r_{i,k} w^r_{j,k} + b^r_j)$
    end for
    for $l \in \{1, \ldots, nOutputs\}$ do
      $O^o_k := \text{selu}(\sum_{h=0}^{nNodes} O^h_{l,h} w^o_{h,l} + b^o_l)$
    end for
    $O^r := O^h$
  end for
  Fault diagnosis of proppeler using O^o
until {The system is unpowered}

During development, a timing test using the algorithm described in Algorithm 1 on an HP Zbook Studio G5 with an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, resulted in a processing rate of 12,000 Hz. Because this exceeds the maximum sampling frequency of the Myxa ESC by a factor of 120, the choice was made to use an (RPI4B) Raspberry Pi model 4B, with a Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.

test	1	2	3
time(s)	30.0929	30.1063	30.0859

Table 3: Results of the timing test

Repeating the timing test on the RPI4B results in the values seen in Table 3. From these results, we can see that the three tests result in very similar results, with an average of 30.095 seconds resulting in a processing rate of 3322,811 Hz exceeding the maximum sampling frequency of the Myxa ESC by a factor of 33.2.

When analysing the RNN architecture, the number of multiply accumulate instructions (MAI) can be calculated using the formula seen in Equation 20 where n_n is the number of nodes per hidden layer, n_i is the number of inputs, n_o is the number of outputs, and n_l is the number of hidden layers and n_h the number of historical samples defined as *nHist* in Subsection3.2.

$$n_{MAI} = n_n n_h (n_i + n_o + n_n(2n_l - 1)) \tag{20}$$

When calculating the formula with, $n_n = 72, n_i, 6, n_o = 4, n_l = 1, n_h = 5$ it results in a total of 29520 MAI per iterations of the Recurrent neural network. Theoretically, the RPI4B could execute up to $29520 * 33.2 = 980,064$ MAI per second on each of its four cores. The largest sized network used during training (see Subsection 2) is the 6 hidden layers of 128 nodes each configuration calculates (using Equation 20) to a total of 907,520 MAI. This leaves plenty of room for future architecture improvement while

using an RPI4B.

The testing data presented in Subsection 5.2 shows that the fault diagnosis classification method is not straightforward. Thus, the system's response time can not be based on test data. Instead, a theoretical estimate of the delaying factor for the average filter can be calculated using Equation 21. Where E^f is the threshold at which the normalized error is coincided to be a fault, E^g is the expected normalized error average, and E_t is the Normalized error value that is to be detected.

$$d^{filter} = \lceil n_{inpf} \frac{E^f - E^g}{E_t - E^g} \rceil \tag{21}$$

For example when the expected normalized error average $E^g = 0.05$, the fault diagnosis threshold $E^f = 0.1$ and the error value $E_t = 0.12$ and average filter size $n_{inpf} = 10$ we calculate a delaying factor $d^{filter} = 8$ and the response time of the system is $t^{pm} d^{filter} = 0.01 * 8 = 0.08s$

To estimate the impact energy of the UAV if it were to fall out of the sky, a simplified dynamic model of a free falling object[43] is used to estimate its falling velocity and can be seen in Equation 22:

$$v_{i+1} = v_i + \delta t(g - \frac{kv_i^2}{m}), \quad g = 9.81m/s^2, \quad m = 1.9kg \tag{22}$$

Where v_i is the objects velocity at sample i, t is time, g is an acceleration constant due to gravity, m is the object mass and k is a collection of constants expressed by Equation 23 and Equation 24

$$k = \frac{c_w \rho (A_d + A_p)}{2}, \quad \rho = 1.225 \frac{kg}{m^3} \tag{23}$$

$$c_w = 2.2, \quad A_d = 0.058487m^2 \tag{24}$$

Where ρ is the air density coefficient, c_w is the drag coefficient, A_d is the area of the X500 v1 UAV, and A_p is the area of the parachute, which for now will be set at 0. Finally, the impact energy can be calculated using Equation 25.

$$j_i = \frac{mv_i^2}{2} \tag{25}$$

Figure 23 shows the falling velocity and impact energy over time assuming that $t_0 = 0$ and $v_0 = 0$. It can be seen from this plot that after only slightly more than 0.4 seconds of free fall, the threshold at which skull fractures start occurring has already been passed, and after around 1.2 seconds of free fall, lethal impacts can start occurring. As no action is taken, the fall velocity will keep increasing until the terminal velocity is reached.

To model the parachute mentioned in Section 4.1 a simple change in the object's surface area is sufficient. Based on the information provided by the vendor and analysis of the product videos on the website, a conclusion can be made that the parachute takes approximately 1.5 seconds to deploy, whereas the parachute starts to unfold in the

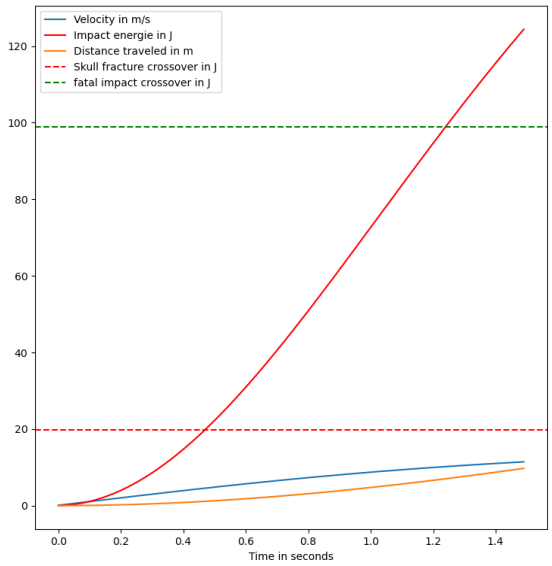


Figure 23: Object velocity and impact energy during a free fall

last 0.1 seconds[44]. If we assume 0.1 seconds for detection time, the area of the parachute is a simple linear model given by Equation:

$$A_i^p = \begin{cases} 0 & t_i < 1.5 \\ \frac{(t(i)-1.5)0.8938319931}{1.6-1.5} & 1.5 \leq t_i \leq 1.6 \\ 0.89383199312 & \text{otherwise} \end{cases} \tag{26}$$

When we integrate the parachute into the model shown earlier, we get the following plot shown in Figure 24. From this plot, the conclusion can be made that if the model is a good enough representation of the physical system, the system reaches a terminal velocity below the thresholds at which skull fractures start occurring after approximately 2.1 seconds. Another observation that can be made is that it before the falling velocity converges to a stable falling speed, the UAV has been falling for at least 13.5 meters. This means that if we assume a minimum flying height of 20 meters, a safety margin of 33.5% is introduced, and the maximum response time of the parachute deploy time is increased to 0.6 seconds.

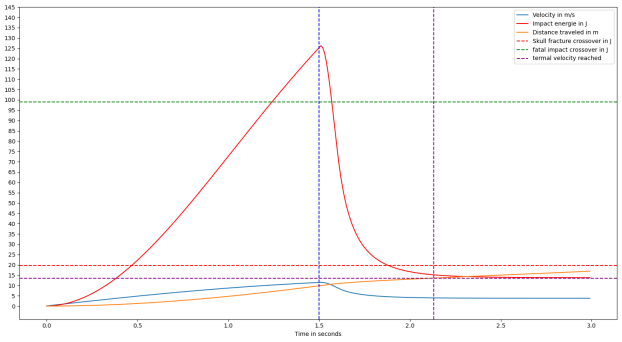


Figure 24: Object velocity and impact energy during free fall including a parachute

5.4. Transfer-ability

The test flights were performed as described in Subsection 4.5, and 12 tests were performed, of which three tests were performed with the undamaged propeller and two tests each with the four damaged propellers D1 - D4. The tests performed with the propeller D2 were severely limited in duration because the damaged propeller caused unstable behaviour in the UAV during flight. One of the tests after recording only lasted about 1.5 seconds, and the other stopped after approximately 10 seconds. On flight with the D3 propeller was also shorter and lasted only around 7.5 seconds. This instability might have been prevented by better ESC PID tuning or flight PID controller tuning, but this was outside of the scope of expertise during this research. The reduced sample set can be seen in the results described below.

Table 7 on page 17 shows the maximum value, mean and standard deviation based on the normalized error between the prediction and measurement obtained during live flight. From these measurements, it can be seen that there is a significant difference between the unbroken and broken propellers. Unlike the data obtained in the static propeller setup, the propeller labelled D2 has significant differences in all measurements and not only significant or small differences in most measurements. The mean normalized error for the power of the propeller labelled D2 even exceeds that of the propeller labelled T by a factor of 9.6.

Figure 26 on page 19 shows the plots for normalized error for RPM, Power and Temperature over time. The RPM plot shows that no simple classification can be performed similarly to the results shown in Subsection 5.2. However, this is where the results show that if we look at the power and temperature graphs, it can be seen that classification or fault diagnosis can be performed by simple thresholding. An example threshold has been added to these plots as a striped red horizontal line labelled 'E'. The normalized error data for the propeller labelled T is significantly closer to 0 in these graphs.

5.5. Discussion

In the first part of this Section, it was shown that several types of neural network architectures could be used to create the prediction model. The RNN network architecture performed best with the LSTM network architecture as a close second. However, the LSTM network architecture is much more resource intensive and thus should not be considered even if it would be slightly better than the RNN network architecture.

The results that are obtained through the static measurement setup show that it should be possible to perform fault diagnosis, but the methodology is not straight forward as expected at the start of this thesis project. The normalized error plots of the static input show that when the motor speed set-points are in the higher part of the spectrum, the damaged propeller labelled D2 can clearly be distinguished. However, the other plots show that no simple measure can be used for classification. Based on the Maximum, Mean and Standard deviation, fault diagnosis may still be performed but will require additional research.

The timing requirements are met with a factor of 33.2, an approximation of maximum network size is also considered, and a mathematical expression is given to calculate the number of MAI instructions performed for a specific network size.

During test flights, the results showed high potential where the power and temperature normalized error plots showed that simple thresholding techniques can be used to perform fault diagnosis. Compared to the static input, these results are much better. It can be assumed that the input signals generated by the flight controller are in a favourable range as to the generated training data. The accuracy evaluation was also performed on relatively dynamic data in the form of ramps and steps; however the flight controller generates a much less dynamic input. Because of the limited knowledge of ESCs and flight controller tuning, the test data was limited, and thus, further research will need to be done into the stability of the data.

It can also be noted the processing time for the prediction model is much shorter than the sampling time of the ESC, but a delaying factor is introduced by filtering the input data. The processing platform and ESCs used during this research are off-the-shelf products. It is expected that hardware created explicitly for this purpose would significantly outperform the system designed during this research.

6. Conclusion

To finalise this research, The proposed research questions are reviewed in Subsection 6.1 and potential future work is presented in Subsection 6.2.

6.1. Research questions

First, in this Subsection, the sub research question proposed back in Section 1 are addressed. After which, the main research question will be addressed.

- How accurate does a prediction model have be met to preform fault diagnosis reliably?

For a UAV to detect failure and respond accordingly, reliability is essential. As can be seen from the calculation in Subsection 5.3, a falling UAV turns into a dangerous object quickly. Given a limited allotted response, it can therefor be assumed that a very high detection rate (close

to 100%) is required. In other use cases where the risk is lower and the allotted response time is longer, a near 100% detection might not be required.

The essence of this research question refers to reliability, but as described above, the system's reliability depends on the use case. During this research, it is assumed that UAVs are used for civil applications. Thus, the risks of collateral damages are much higher, and the very high detection rates mentioned earlier are required.

- What timing constraints have to be satisfied to meet the required reliability?

Regarding timing constraints, as discussed in Subsection 4.4, there are two topics to consider. The processing time of each sample and the response time of the system. When the processing frequency of the prediction model is slower than the sensor's sample rate, only multiples of the maximum sample rate will be used, inherently increasing the response time of the fault detection. The response time has a lower bound of at least the processing time of the system and an upper bound potentially introduced by filtering.

The timing constraints are a product of the use case and the allotted time for response. The example given in Subsection 5.3 allows for a 0.6 second response time provided that the UAV flies at least 20 meters above objects that could potentially be harmed. In other use cases where the UAV flies much higher, this maximum response can be increased further.

- Can a processor based solution be used to meet the timing and reliability constraints?

In Subsection 5.3, it is explained that the processor-based RNN prediction model developed during this research runs on an RPI4B model runs at a factor of 33.2 faster than state-of-the-art ESC maximum sampling rate. From the results in Subsection 5.4 though limited data has been analysed thusfar, it can also be concluded that the reliable detection of damaged propellers during flight is also possible using simple classification methods such as thresholding.

- To what extend can neural network based prediction models be used to reliably perform real-time fault diagnosis of UAV actuators on component level on board and in real-time?

Based on the previous research questions, it can be concluded that given some limited testing data, it is possible to perform fault diagnosis of UAV actuators on board and in real-time. It is to be noted that during the hand-flown test flights presented in Subsection 5.4, the flight controller's input data was relatively non-dynamic. If the input data in other use-cases has a more, then based on the results shown in Subsection 5.2 we can conclude that the prediction model used during this research is not reliable enough.

6.2. Future work

Based on the results, discussion and conclusion the following topics are proposed as future work:

Classification

Data from live flights have shown that simple classification such as thresholding would suffice for reliable fault diagnosis. However, data obtained through testing on the static propeller setup has shown that more advanced classification methods would be required when the motor speed set-points are more dynamic.

Customized sampling hardware

The output quality of a system can only ever be as good as the quality of its input. Though excellent speed controllers, the state-of-the-art ESCs used during this research were not built with fault diagnosis based on sampling data in mind. Better quality sampling could significantly improve the reliability and response time.

Improvement of the prediction model, using more varied training data

Based on the results shown in Subsection 5.2, the prediction model performed poorly at correctly predicting the system's output whenever the motor set-points have a dynamic nature. The input generated during the training phases consists of only constant, ramp, step and randomized data. However, training data based on live flights generated by a flight controller was not considered.

Improvement of the prediction model, using training optimization

During the training process, a procedure involved picking the network sizes and architectures by hand. Optimization algorithms such as AutoKeras might prove to outperform this process. Different training optimization algorithms and parameters will also be considered in future work.

Extend use cases

During this research, and especially due to time constraints, only the surface of UAV motor fault diagnosis on component level was touched. It is to be researched whether the fault diagnosis system is still reliable in different use cases such as physical interaction or a change in system dynamics such as package delivery.

		max	mean	stdev
rpm				
	T	0.001114	0.0004867	0.0000874
	D1	0.001126	0.0004672	0.0000825
	D2	0.002202	0.0008750	0.0004501
	D3	0.002345	0.0006167	0.0001685
	D4	0.001323	0.0004134	0.0000960
power				
	T	0.092911	0.0115844	0.0083384
	D1	0.105042	0.0144945	0.0078959
	D2	0.113306	0.0575010	0.0348189
	D3	0.111283	0.0168913	0.0050223
	D4	0.105499	0.0186164	0.0067951
temp				
	T	0.001533	0.0002370	0.0001552
	D1	0.001004	0.0002159	0.0001376
	D2	0.009960	0.0043048	0.0041692
	D3	0.001423	0.0001931	0.0001562
	D4	0.001011	0.0001349	0.0001389

Table 4: constant input normalized error between prediction and measurement

		max	mean	stdev
rpm				
	T	0.002136	0.001054	0.000461
	D1	0.001896	0.001036	0.000378
	D2	0.006038	0.002357	0.001639
	D3	0.003237	0.001632	0.000776
	D4	0.003383	0.001395	0.000814
power				
	T	0.147440	0.080936	0.029019
	D1	0.154980	0.083917	0.031979
	D2	0.301802	0.131455	0.059312
	D3	0.163149	0.090605	0.033384
	D4	0.159425	0.096552	0.027612
temp				
	T	0.002172	0.001490	0.000349
	D1	0.001533	0.000708	0.000200
	D2	0.013349	0.003710	0.003778
	D3	0.002998	0.001165	0.000566
	D4	0.002067	0.000965	0.000274

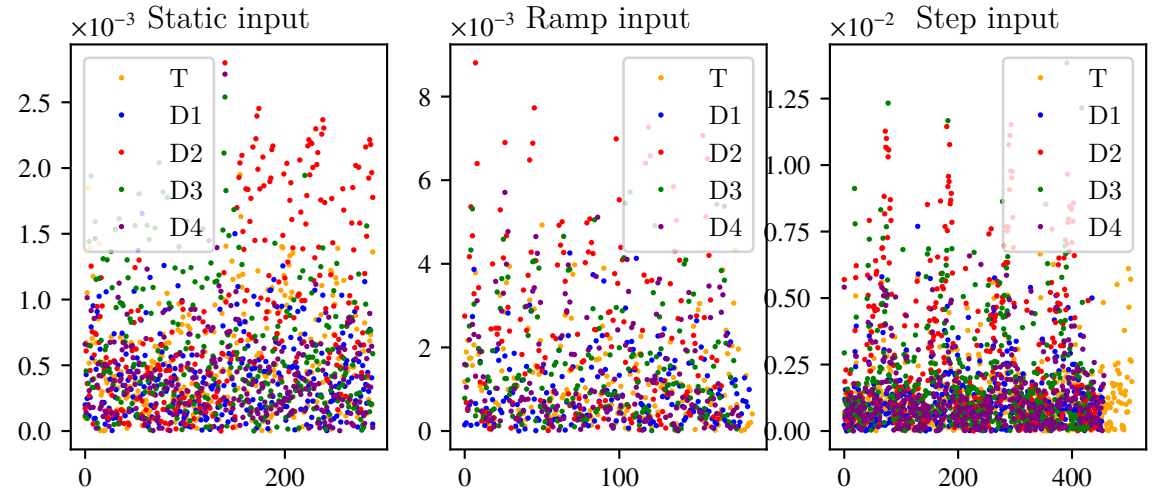
Table 5: Ramp input normalized error between prediction and measurement

		max	mean	stdev
rpm				
	T	0.002637	0.001209	0.000550
	D1	0.003098	0.001232	0.000623
	D2	0.009770	0.002982	0.002566
	D3	0.007304	0.001915	0.001393
	D4	0.003638	0.001402	0.000842
power				
	T	0.173714	0.051949	0.045161
	D1	0.168963	0.048598	0.043467
	D2	0.227429	0.079839	0.054583
	D3	0.183345	0.050118	0.041628
	D4	0.177913	0.049898	0.043880
temp				
	T	0.002104	0.001142	0.000353
	D1	0.001533	0.000834	0.000362
	D2	0.015572	0.002739	0.003792
	D3	0.005129	0.001070	0.000846
	D4	0.001743	0.000766	0.000383

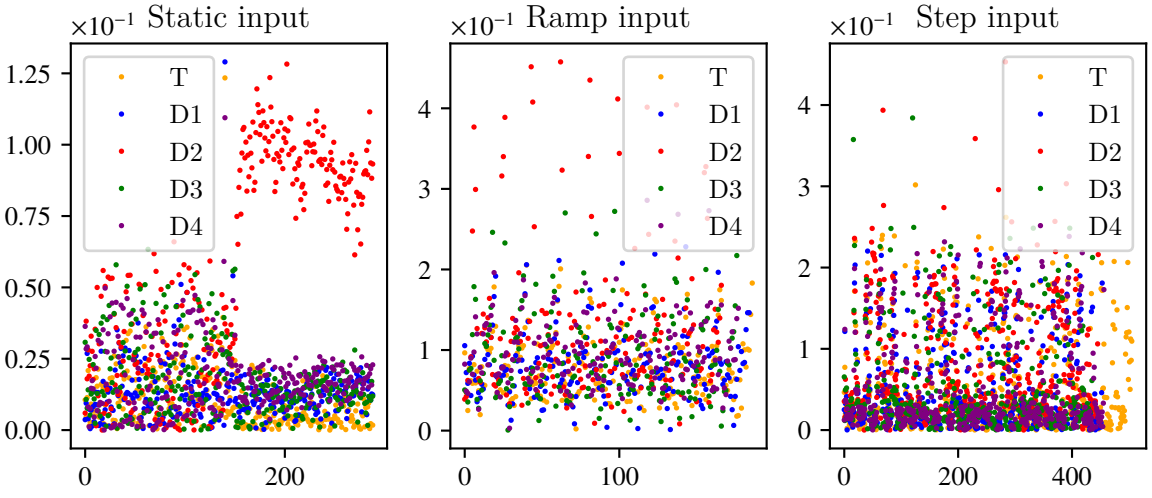
Table 6: Step input normalized error between prediction and measurement

		max	mean	stdev
rpm				
	T	0.005	0.0013	0.0009
	d1	0.0061	0.0016	0.0012
	d2	0.0106	0.0029	0.0019
	d3	0.0045	0.0015	0.0009
	d4	0.0064	0.0018	0.0013
power				
	T	0.0506	0.0217	0.0104
	d1	0.1066	0.0417	0.0128
	d2	0.3264	0.2084	0.0438
	d3	0.1126	0.0926	0.0083
	d4	0.1998	0.1174	0.0268
temp				
	T	0.0035	0.0023	0.0007
	d1	0.0041	0.0034	0.0003
	d2	0.01	0.0116	0.002
	d3	0.0063	0.0057	0.0007
	d4	0.0111	0.0086	0.0017

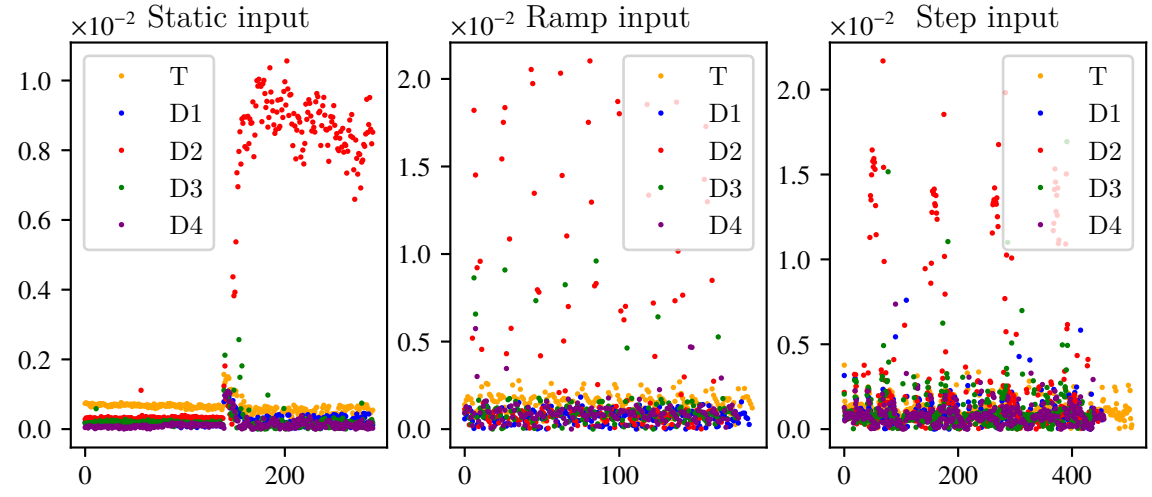
Table 7: live flight normalized error between prediction and measurement



(a) Normalized error for RPM

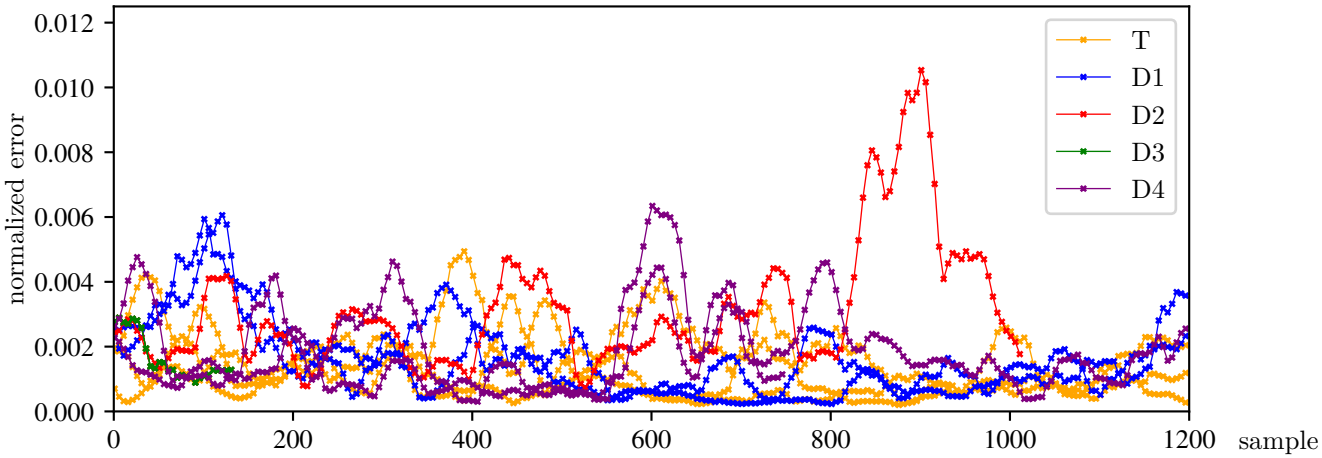


(b) Normalized error for Power

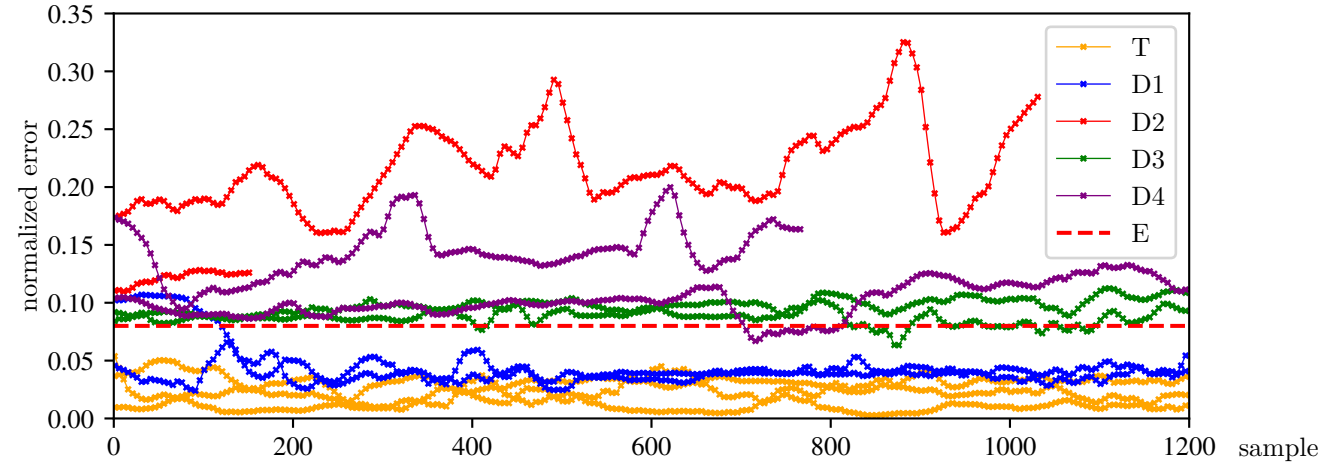


(c) Normalized error for Temperature

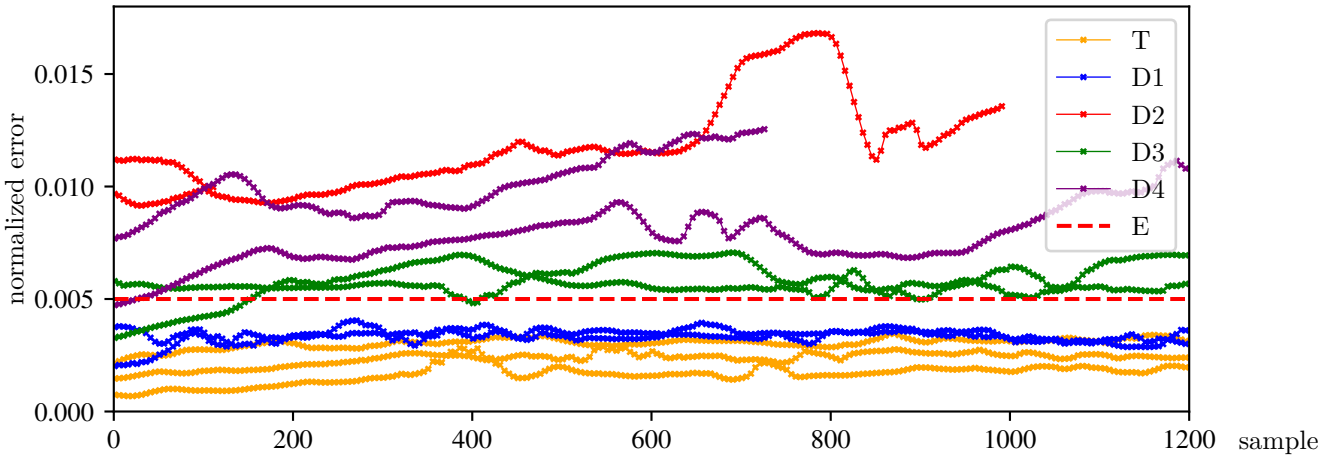
Figure 25: Measurement data on the propeller setup



(a) Normalized error for RPM



(b) Normalized error for Power



(c) Normalized error for Temperature

Figure 26: Measurement data during live flight

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, M. Guizani, Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges, Ieee Access 7 (2019) 48572–48634.

[2] D. Tezza, M. Andujar, The state-of-the-art of human–drone interaction: A survey, IEEE Access 7 (2019) 167438–167454. doi:10.1109/ACCESS.2019.2953900.

[3] A. Y. Mersha, Towards the first and Best EU-approved Autonomous Security drone for BVLOS flighT (The BEAST), 2019.

[4] L. Marconi, F. Basile, G. Caprari, R. Carloni, P. Chiacchio, C. Hurzeler, V. Lippiello, R. Naldi, J. Nikolic, B. Siciliano, et al., Aerial service robotics: The airobots perspective, in: 2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI), IEEE, 2012, pp. 64–69.

[5] A. Y. Mersha, On autonomous and teleoperated aerial service robots, Lateral 12 (2014) y0u.

[6] E. union law, Commission implementing regulation (eu) 2019/947 of 24 may 2019 on the rules and procedures for the operation of unmanned aircraft (text with eea relevance.) [cited 09.09.2021].
URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32019R0947from=EN

[7] W.-H. Lee, C.-G. Park, A fault detection method of redundant imu using modified principal component analysis, International Journal of Aeronautical and Space Sciences 13 (3) (2012) 398–404.

[8] Z. Tu, F. Fei, M. Eagon, X. Zhang, D. Xu, X. Deng, Redundancyfree uav sensor fault isolation and recovery, arXiv preprint 2018 (1812).

[9] S. Sun, G. Cioffi, C. De Visser, D. Scaramuzza, Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events, IEEE Robotics and Automation Letters 6 (2) (2021) 580–587.

[10] DJI, Dji [cited 17.09.2021].
URL https://www.dji.com/

[11] D. Volt, Spraying drone [cited 17.09.2021].
URL https://www.dronevolt.com/en/drone-volt-services-en

[12] Aerialtronics, Aerialtronics [cited 17.09.2021].
URL https://www.aerialtronics.com

[13] A. V. Shelley, A model of human harm from a falling unmanned aircraft: Implications for uas regulation, International Journal of Aviation, Aeronautics, and Aerospace 3 (3) (2016) 1.

[14] G. K. Fourlas, G. C. Karras, A survey on fault diagnosis methods for uavs, in: 2021 International Conference on Unmanned Aircraft Systems (ICUAS), 2021, pp. 394–403. doi:10.1109/ICUAS51884.2021.9476733.

[15] R. Da Costa, Q. Chu, J. Mulder, Reentry flight controller design using nonlinear dynamic inversion, Journal of Spacecraft and Rockets 40 (1) (2003) 64–71.

[16] S. Sun, M. Baert, B. S. Van Schijndel, C. De Visser, Upset recovery control for quadrotors subjected to a complete rotor failure from large initial disturbances, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 4273–4279.

[17] P. Lu, E.-J. van Kampen, Active fault-tolerant control for quadrotors subjected to a complete rotor failure, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015, pp. 4698–4703.

[18] B. Wang, Z. Wang, L. Liu, D. Liu, X. Peng, Data-driven anomaly detection for uav sensor data based on deep learning prediction model, in: 2019 Prognostics and System Health Management Conference (PHM-Paris), IEEE, 2019, pp. 286–290.

[19] J. Fu, C. Sun, Z. Yu, L. Liu, A hybrid cnn-lstm model based actuator fault diagnosis for six-rotor uavs, in: 2019 Chinese Control And Decision Conference (CCDC), IEEE, 2019, pp. 410–414.

[20] D. Liu, Z. Wang, S. Wang, Y. Pang, L. Liu, Uav sensor data anomaly detection using predictor with uncertainty estimation and its acceleration on fpga, in: 2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), IEEE, 2018, pp. 1–6.

[21] R. C. Avram, X. Zhang, J. Muse, Quadrotor actuator fault diagnosis and accommodation using nonlinear adaptive estimators, IEEE Transactions on Control Systems Technology 25 (6) (2017) 2219–2226.

[22] H. Zhang, Q. Gao, F. Pan, An online fault diagnosis method for actuators of quadrotor uav with novel configuration based on imm, in: 2020 Chinese Automation Congress (CAC), 2020, pp. 618–623. doi:10.1109/CAC51589.2020.9326877.

[23] S.-C. Wang, Artificial neural network, in: Interdisciplinary computing in java programming, Springer, 2003, pp. 81–100.

[24] B. Yegnanarayana, Artificial neural networks, PHI Learning Pvt. Ltd., 2009.

[25] A. D. Rasamoelina, F. Adjailia, P. Sinčák, A review of activation function for artificial neural network, in: 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), 2020, pp. 281–286. doi:10.1109/SAMI48414.2020.9108717.

[26] K.-E. P. Sang-Gu LEE with Jae Hwa LEE, Yoonmee HAM, Introductory Mathematics for Artificial Intelligence, 2021.

[27] Wikipedia, Connectionism [cited 08.11.2021].
URL https://en.wikipedia.org/wiki/Connectionism

[28] L. R. Medsker, L. Jain, Recurrent neural networks, Design and Applications 5 (2001) 64–67.

[29] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International conference on machine learning, PMLR, 2013, pp. 1310–1318.

[30] researchgate, The comparison between recurrent neural network (rnn) and feed-forward neural network (ffnn). it demonstrates in ffnn there is only one direction for the data to move, whereas in rnn there is a loop. [cited 08.11.2021].
URL https://www.researchgate.net/figure/The-comparison-between-Recurrent-Neural-Network-RNN-and-Feed-Forward-Neural-Network$_fig1_3$38672883

[31] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[32] UAVCAN, list of standard data types [cited 01.02.2022].
URL https://legacy.uavcan.org/Specification/7._List_of_standard_data_types/status-2

[33] R. Dey, F. M. Salem, Gate-variants of gated recurrent unit (gru) neural networks, in: 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017, pp. 1597–1600. doi:10.1109/MWSCAS.2017.8053243.

[34] Z. Wang, Y. Ma, Z. Liu, J. Tang, R-transformer: Recurrent neural network enhanced transformer, arXiv preprint arXiv:1907.05572 (2019).

[35] H. Jin, Q. Song, X. Hu, Auto-keras: An efficient neural architecture search system, in: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 1946–1956.

[36] S. Lawrence, C. L. Giles, A. C. Tsoi, What size neural network gives optimal generalization? convergence properties of backpropagation, Tech. rep. (1998).

[37] HolyBro, Holybro x500 v1 [cited 21.01.2022].
URL http://shop.holybro.com/x500-kit$_p$1180.*html*

[38] Zubax, Zubax myxa [cited 24.01.2022].
URL https://shop.zubax.com/collections/motor-controllers/products/zubax-myxa?variant=12528945496163

[39] P. autopilot, Px4 user guide [cited 24.01.2022].
URL https://docs.px4.io/master/en/flight_controller/pixhawk4.html

[40] Zubax, Zubax babel [cited 24.01.2022].
URL https://shop.zubax.com/products/zubax-babel?_pos=1_sid=b3a9b92db_ss=rvariant=6012823404573

[41] raspberri pi, Raspberrypi4b [cited 30.06.2022].
URL https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[42] Z. Zhang, Improved adam optimizer for deep neural networks, in: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018, pp. 1–2. doi:10.1109/IWQoS.2018.8624183.
[43] D. W. MacDougal, Galileo's great discovery: How things fall, in: Newton's Gravity, Springer, 2012, pp. 17–36.
[44] F. C. P. Manufacturer, Fruity chutes promotion video [cited 13.08.2022].
URL https://www.youtube.com/watch?v=6YrCD23PAAt=17sab_channel=FruityChutes-ParachuteManufacturer