

Technische Universität Chemnitz
Fakultät für Informatik
Professorship of Neurorobotics



Forschungspraktikum

Fault Tolerant Quadrotor Control

Supervisors: Prof. Dr. Florian Röhrbein, , Dr.-Ing. Sven Lange, Saranraj Nam-busubramaniyan

Student:

Hassanien, Ahmad

ahmad.hassanien@s2021.tu-chemnitz.de

ID:696652

Table of contents

1	Abstract.....	1
2	Introduction.....	1
	2.1 Basics of quadrotor flight.....	1
	2.2 Position and orientation	2
3	Related Works and contribution	2
4	Control Methods	3
	4.1 PID controller.....	3
	4.2 Model Predictive Control.....	4
	4.2.1 Inertial Properties	6
	4.2.2 Simulation Results.....	7
	4.3 PPO (Proximal Policy Optimisation).....	8
	4.3.1 Recurrent PPO	10
	References	12

1 Abstract

This research project was about developing a solution to maintain flight of a faulty drone. Two approaches were developed, one using a neural network trained with PPO, and a Model Predictive Controller.

2 Introduction

Quadcopters have many civil and military applications, they can be used for rescue missions, environmental monitoring, and surveillance. Quadrotors have a high maneuverability, they can take off and land vertically unlike fixed wing planes. They can carry sufficient payload to perform indoor and outdoor applications. Another advantage is the simple design of the motors and propellers, which have a fixed pitch and rotate in one direction. There is also no need for hinges or flaps, this makes quadrotors quite robust. However this doesn't come without limitations, since quadrotors have don't have wings, they are less energy efficient. The required sensors such as cameras or Lidar sensors can add a significant payload on the drone, in addition to consuming more power.

Another issue is the failure of any of the rotors. Classical quadrotor controllers such as PID controllers cannot handle faulty rotors. It would be desirable to be able to still control the quadcopter, even in the case of rotor failure, so as to at least recover the drone safely.

2.1 Basics of quadrotor flight

Having only 4 rotors makes quadrotors cheaper than hexa-copters, but this also could be a point of weakness for the design, as there is no redundancy. The quadrotor has 6 degrees of freedom, but only 4 rotors for control input, this makes it an underactuated system. Direct horizontal translation is not possible, the drone must first tilt in the desired direction to achieve horizontal translations.

Assuming a non-faulty case, controlling the pitch, roll, yaw and thrust can be done by varying the speeds of the rotors as follows:

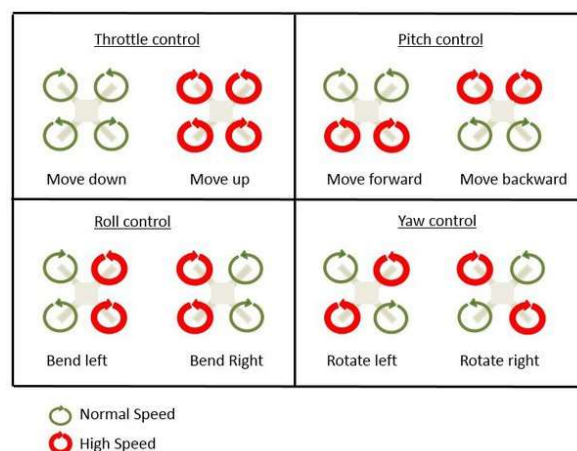


Figure 1: How a quadrotor is able to control the pitch, yaw, roll and elevation by varying the RPMs

But in the case of the failure of any one of the rotors, it is no longer possible to control the yaw angle, so the quadrotor will spin around itself. However, one can still control the drone to reach a desired height or position in space, as we will see in this report.

2.2 Position and orientation

We have a simple way to define the position of the quadrotor in the environment

$$x = (x, y, z)^T$$

And the velocities

$$\dot{x} = (\dot{x}, \dot{y}, \dot{z})^T$$

The orientation of the drone can be described with 3 angles, roll, pitch, and yaw

$$o = (\phi, \theta, \psi)^T$$

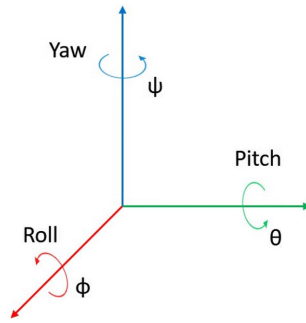


Figure 2: The 3 angles describing orientation

Similarly, the change in these angles can be written as

$$\dot{o} = (\dot{\phi}, \dot{\theta}, \dot{\psi})^T$$

3 Related Works and contribution

An earlier approach to tackle fault tolerant flight in quadrotors was from [1], which described the use of Non Linear Dynamic Inversion along with an uncertainty and disturbance estimator to achieve fault tolerant control. The idea behind NDI is to modify the differential equations of the quadrotor with a virtual control input, which simplifies the control task. The NDI is combined with an Uncertainty and Disturbance Estimator (UDE), which estimates the disturbances applied on the quadrotor, so that the disturbances can be countered.

The main inspiration for this research project was [2], where the authors built a Non-Linear MPC to achieve fault tolerance in a quadrotor, in the case of the complete failure of one rotor. They additionally used Incremental Non Linear Dynamic inversion to adjust the control signal from the NMPC to account for additional uncertainties such aerodynamic effects.

We replicated similar results to [2], where our controller can handle (partial) failure of up to two rotors by

building our own Non-Linear MPC. The novel contribution of this project was to then perform the same task of Fault Tolerance using Deep Learning methods.

Another inspiration was [3], where the authors developed a Linear Model Predictive controller and compared it to a PID controller. However, the equations of motion used in the MPC in this paper were simplified, resulting in a linear MPC.

In a paper that integrates both fault prediction and fault tolerance [4], the authors concurrent learning to perform fault detection, and a Non Linear Model Predictive Controller to achieve fault tolerance. Concurrent learning estimates the loss of effectiveness of the now broken quadrotor actuator from the historical data. Fault diagnoses is needed because one needs to adjust the parameters of the NMPC to reflect the change in the dynamics of the craft.

In [5], the authors used a neural network to estimate the uncertainties in the model of the quadrotor, as well as estimate uncertainties due actuator faults. The network was used with an adaptive control method called virtual parameter estimation.

4 Control Methods

4.1 PID controller

The most common controller on quadrotors is a Proportional Integral Derivative Controller (PID). In the case of no fault in the rotors, the PID controller can handle the drone and reach the desired states:

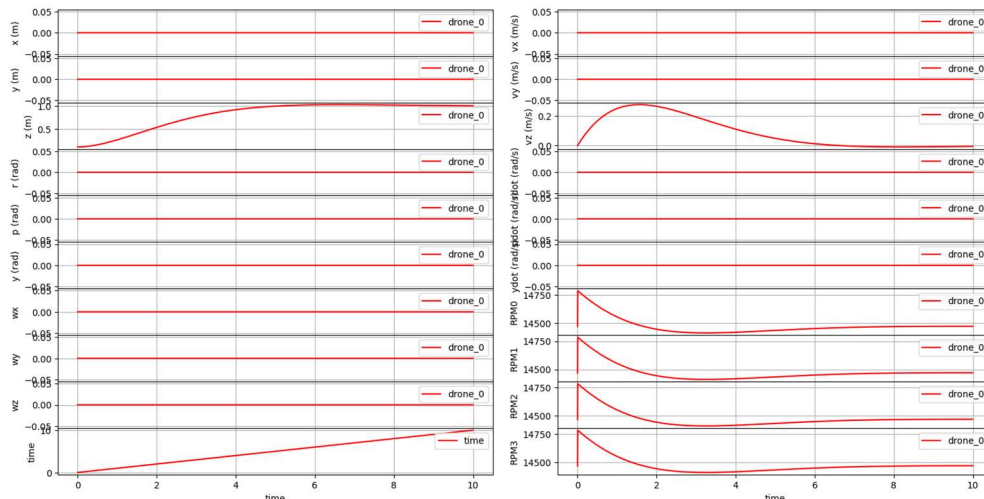


Figure 3: Takeoff and hover at 1 meters, with PID, non-faulty drone

Lets now look at the faulty case. With all 3 controllers examined in the report, we used the same fault in the rotors: Two diagonally placed rotors (Rotor 0 and Rotor 2), are set to function at 90 percent capacity.

For the case of the PID, the controller was not able to maintain stable take off and hover with

faulty rotors:

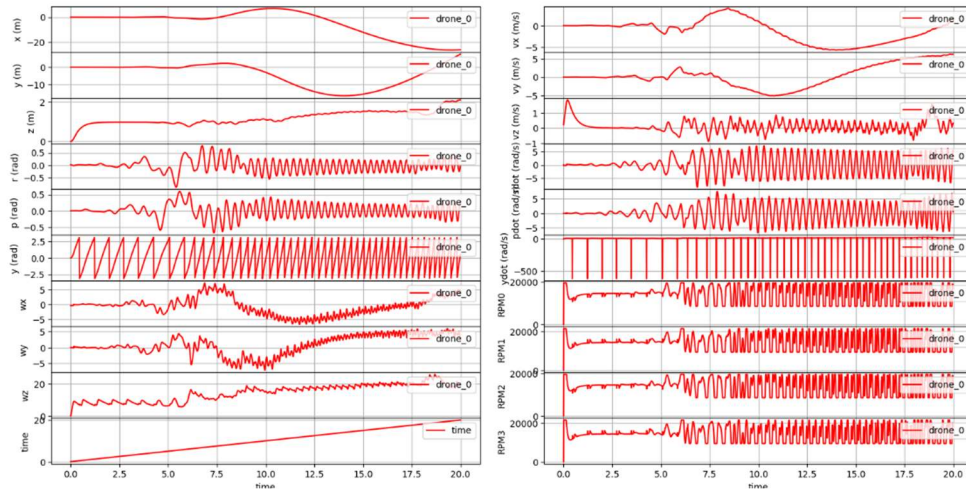


Figure 4: Crash recording from faulty drone with PID controller. The quadrotor completely misses the desired position

4.2 Model Predictive Control

As the name suggests, a model predictive controller uses a physical model of the drone to predict how the drone will move, and thus predict the optimal control signal. The controller forecasts the future behaviour of the drone, applies the first action, then recalculates the prediction of future behaviour (up to a certain point in time, called the Horizon) [7], [9]. We define the state error as the difference between the current state (for example, the elevation) and the desired state.

The control error is the difference between the desired control signal (derived from the mathematical model of the drone), and the actual control signal.

The controller constantly looks at the state error and the control error, aiming to minimize both.

$$\text{cost function} = \text{state}_{\text{error}} \cdot Q + \text{control}_{\text{error}} \cdot R$$

We can modify the behaviour of the controller by changing the two tuning matrices, Q and R. The ratio between these two values will decide how the controller will behave. For example, a higher Q value will cause the controller to more aggressively attempt to follow the desired state trajectory. Having a higher R value will cause the controller to put more importance on minimizing the control effort done by the actuators (so less aggressive control inputs, which could extend the longevity of the motors).

Another advantage that MPC is the ability to define constraints. One can define constraints on the states, for example to limit the velocity of the drone within a certain range or limit it to a certain maximum height. One can also define constraints on the control signal, such that the controller knows the maximum and minimum thrusts that the motors of the drone can generate [8].

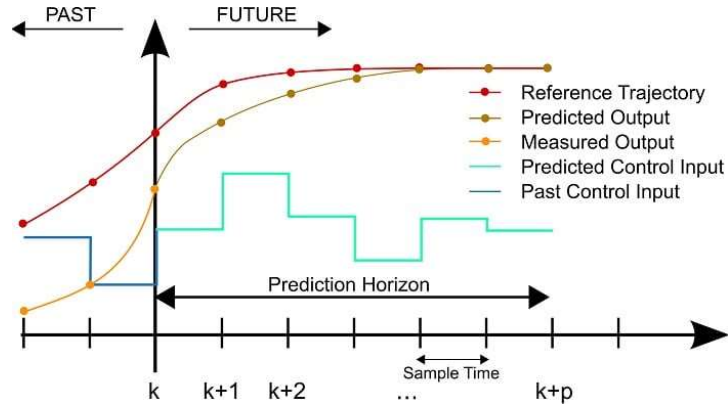


Figure 5: Basic working principles of MPC. Robust Model Predictive Control for Autonomous Spaceships with Failing Sensors. Sinan Harputluoglu

In the figure above, p defines the future prediction horizon mentioned earlier.

The whole optimization process is repeated at each control step k , as the horizon shifts.

We can describe how the drone will behave using the following equations:

$$\ddot{z} = \frac{t}{m} - g$$

$$\ddot{x} = \sin(\theta) \cdot \frac{t}{mg} \cdot \cos(\psi)$$

$$\ddot{y} = \sin(\phi) \cdot \frac{t}{mg} \cdot \cos(\psi)$$

$$\ddot{\phi} = \frac{(\dot{\theta} \cdot \psi \cdot (I_{yy} - I_{zz}) + I_a \cdot \phi)}{I_{xx}}$$

$$\ddot{\theta} = \frac{(\dot{\phi} \cdot \psi \cdot (I_{zz} - I_{xx}) + I_a \cdot \theta)}{I_{yy}}$$

$$\ddot{\psi} = \frac{(\dot{\phi} \cdot \dot{\theta} \cdot (I_{xx} - I_{yy}) + \psi)}{I_{zz}}$$

θ : Pitch

ϕ : Roll

ψ : Yaw

m : Mass

t : Thrust

I_a : Arm length

I_{yy}, I_{xx}, I_{zz} : Moment of Inertia constants

The states of the quadrotor are:

x, y, z

$\dot{x}, \dot{y}, \dot{z}$

θ, ϕ, ψ

$\dot{\theta}, \dot{\phi}, \dot{\psi}$

The MPC monitors the states (y) of the drone, and finds the optimal control signal (u), which in this case would be the overall thrust + 3 torques to be applied on the body of the drone.

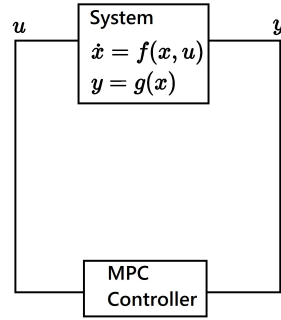


Figure 6: Feedback loop between the drone and MPC

Finally we mix the 3 torques and the overall thrust using the mixer matrix, to get the desired RPMs.

One disadvantage of the MPC is the computational complexity. Since it has to repeatedly perform the optimization process each time step.

To summarize, the complete control loop can be seen here:

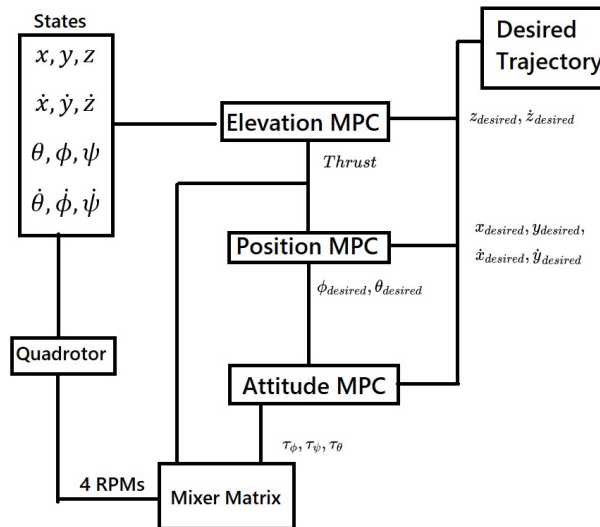


Figure 7: Control diagram for MPC approach

4.2.1 Inertial Properties

As the name suggests, the MPC needs a model to describe how the drone will behave in order to control it. One aspect that we need to know is how fast the angular acceleration of the drone will be, if we apply a certain amount of torque on any of the 3 axes. This property of the drone can be described by the moment of inertias on the 3 axes. They are constants, which can be calculated with an accurate SolidWorks model of the drone that describes the mass and lengths of the arms

[6].

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Parameter	Value
I_{xx}	$2.3951 \cdot 10^{-5} \text{ kg} \cdot \text{m}^2$
I_{yy}	$2.3951 \cdot 10^{-5} \text{ kg} \cdot \text{m}^2$
I_{zz}	$3.2347 \cdot 10^{-5} \text{ kg} \cdot \text{m}^2$
<i>weight</i>	27g
$Ia(\text{length})$	0.046



Figure 8: CrazyFlie 2.0 Drone which was used in simulation.

<https://www.bitcraze.io/products/old-products/crazyflie-2-0/>

4.2.2 Simulation Results

In a final experiment the drone with two compromised rotors (rotor 0 and rotor 2), the drone was tasked to take off and reach a desired location in space. A fault in any of the 4 rotors results in the loss of control over the yaw of the quadrotor, despite this, the MPC was able to control the drone to achieve successful takeoff and reach the desired position.

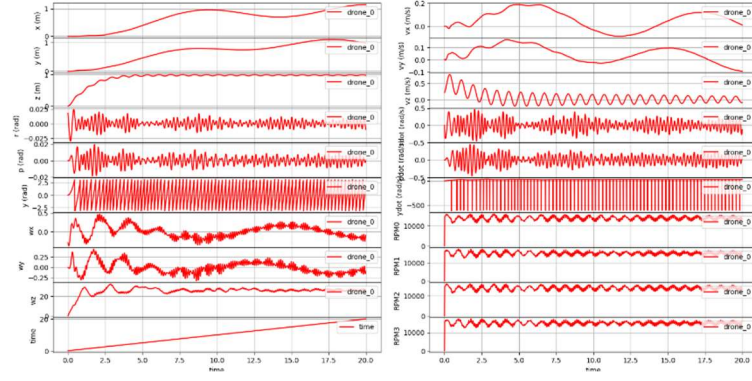


Figure 9: MPC controlled faulty drone reaches desired positions in the xy plane and height

In another experiment to demonstrate safe take off and landing despite damaged rotors:

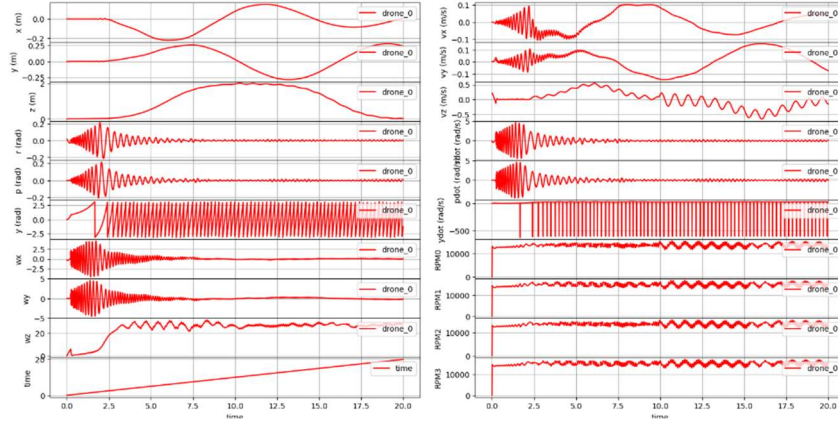


Figure 10: Smooth take-off and landing with damaged rotors

4.3 PPO (Proximal Policy Optimisation)

In the second approach we used a neural network to perform the same task as the MPC, to control the drone with a faulty rotor. We can think of the drone as an **agent**, the agent acts according to a **policy**, to try to maximize some kind of **reward**.

The policy basically looks at the current states of the drone (in our case, the states are position, orientation, and velocities of the drone), and decides what action to do (a velocity vector, telling the drone where it should be heading). How the policy behaves depends on how the parameters in the neural network are set. So now our goal, is to find the best parameters leading to the best performance.

PPO is a Reinforcement Learning technique that was developed in 2017 [10] ,[11] ,[12]. It was developed as an improvement on another algorithm, called Trust Region Policy Optimization (TRPO). One benefit of PPO over TRPO is that it is simpler to implement while performing just as well. PPO is on-policy. This means that the policy we want to improve, is the same policy we will use to collect the data to learn from. PPO can be used with discrete action spaces (for example a Board game) or continuous action spaces (like our drone).

Our parameters are all the weights in the neural network. The goal is to adjust these parameters, such that we end up with a well-tuned neural network that can control the drone. Adjusting the parameters directly affects the performance of the network. The policy gradient tells us in which direction we should adjust the parameters to improve the return. We might want to take big steps towards the positive gradient, however, big changes in the parameters of the network can sometimes cause a policy collapse. This is where policy suddenly performs worse.

To prevent this, PPO tries to avoid stepping too far from the current policy. Let's take a look how this done:

$$\theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{s,a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)]$$

θ_{k+1} tells us how the new network weights will be adjusted (i.e how the policy will be updated). The goal is to maximize the objective L .

Before we move on we should understand what is the Advantage $A^{\pi_{\theta_k}}(s, a)$. The Advantage basically looks at the state we are in, s , and the action that was taken, a , and compares that action to the average of all other actions available in that state. So if the advantage is positive, it means this specific action is better than the average and vice versa.

There are two cases for how the objective will look like: when the Advantage A is positive :

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, (1 + \epsilon) \right) A^{\pi_{\theta_k}}(s, a)$$

and when it is negative:

$$L(s, a, \theta_k, \theta) = \max \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, (1 - \epsilon) \right) A^{\pi_{\theta_k}}(s, a)$$

The ratio $\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}$ compares how different the new policy is from the old policy. So now, the objective is clipped between $(1 + \epsilon)$ and $(1 - \epsilon)$.

ϵ is a hyper parameter, which sets how much the new policy can differ from the old policy.

PPO is an actor critic method. This means one part of the network (the actor), will decide the action, while the critic network tries to estimate the value of this action.

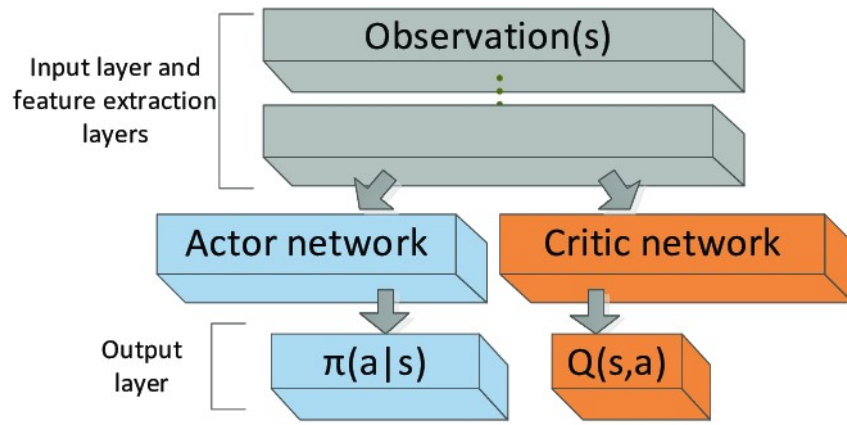


Figure 11: Overview of an actor critic network. Menghao, Wu & Gao, Yanbin & Jung, Alexander & Zhang, Qiang & Du, Shitong. (2019). The Actor-Dueling-Critic Method for Reinforcement Learning. Sensors. 19. 1547. 10.3390/s19071547.

In our specific case, the observations are the states of the quadrotor, and the action is a desired velocity vector, this then goes to a PID controller which finally sends the RPMs. We similarly summarize the whole control loop using the neural network approach here:

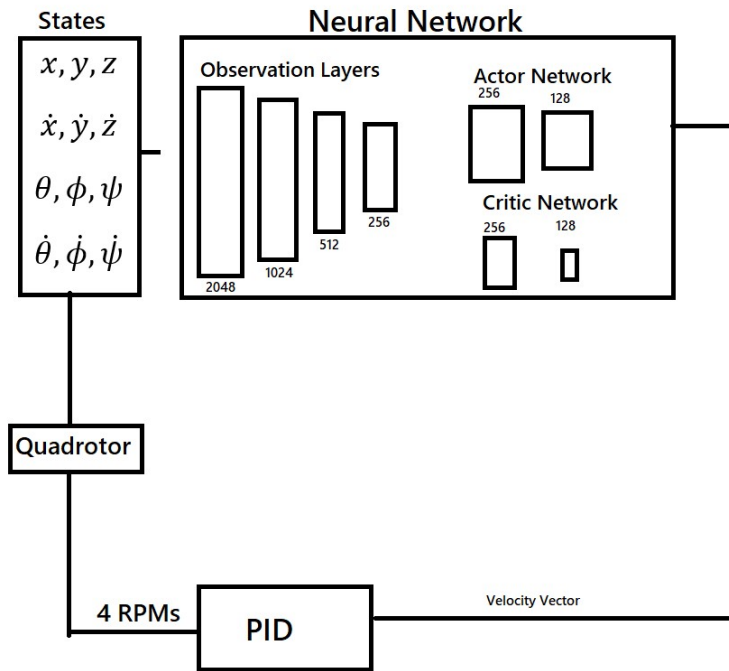


Figure 12: Control diagram for neural network approach

4.3.1 Recurrent PPO

Adding an LSTM (Long Short Term Memory) [13],[14],[15] layer enables our network to have memory. Adding the LSTM layer lets the network capture the temporal dependencies. This basi-

cally means that the network now looks at a sequence of past states rather than just the current state to determine the best control action to maintain flight.

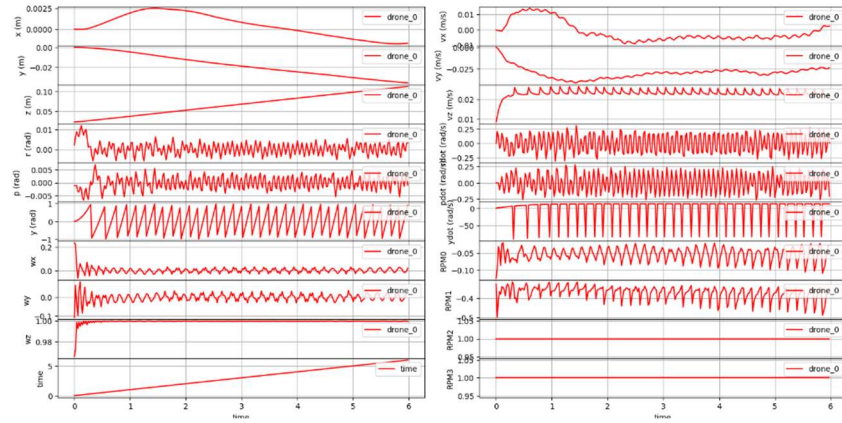


Figure 13: Take-off and hover with a neural network controller on a faulty drone

Similarly, we simulated a quadrotor with two faulty rotors, and noted a successful hover case using the neural network approach.

References

- [1] Dhadekar, D. D., & Talole, S. E. (2021). Robust fault-tolerant flight control of quadrotor against faults in multiple motors. Proceedings of the Institution of Mechanical Engineers Part G: Journal of Aerospace Engineering, 0954410021999547.
- [2] Nan, F., Sun, S., Foehn, P., & Scaramuzza, D. (2022). Nonlinear MPC for Quadrotor Fault-Tolerant Control. IEEE Robotics and Automation Letters, 7(2), 5047-5054.
- [3] Ganga, G., & Dharmana, M. M. (2017). MPC controller for trajectory tracking control of quadcopter. In 2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT) (pp. 1975-1983)
- [4] (2018). Fault-Tolerant Model Predictive Control of a Quadrotor with an Unknown Complete Rotor Failure. IEEE Transactions on Neural Networks and Learning Systems, 30(7), 1975-1983
- [5] (2019). Neuroadaptive Fault-Tolerant Control of Quadrotor UAVs: A More Affordable Solution. IEEE Transactions on Neural Networks and Learning Systems, 30(7), 1975-1983
- [6] Landry, B. P., et al. (2015). Robotics Institute, Massachusetts Institute of Technology. Model Predictive Control for Trajectory Tracking of AUVs. Page 39. https://groups.csail.mit.edu/robotics-center/public_papers/Landry15.pdf
- [7] Do-MPC. (n.d.). Theory of Model Predictive Control. Do-MPC Documentation. https://www.do-mpc.com/en/latest/theory_mpc.html
- [8] Control.com. (n.d.). What is Model Predictive Control (MPC)? Control.com Technical Articles. <https://control.com/technical-articles/what-is-model-predictive-control-mpc/#:~:text=MPC%20is%20an%20iterative%20process,is%20essential%20while%20implementing%20MPC.>
- [9] MathWorks. (n.d.). Understanding Model Predictive Control - Part 2: What is MPC? MathWorks Videos. <https://de.mathworks.com/videos/understanding-model-predictive-control-part-2-what-is-mpc--1528106359076.html>
- [10] OpenAI. (n.d.). OpenAI Baselines: PPO. OpenAI Research. <https://openai.com/research/openai-baselines-ppo>
- [11] OpenAI. (n.d.). Proximal Policy Optimization. OpenAI Spinning Up. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>
- [12] GeeksforGeeks. (n.d.). A Brief Introduction to Proximal Policy Optimization. GeeksforGeeks. <https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policy-optimization/>
- [13] Analytics Vidhya. (n.d.). LSTMs Explained: A Complete, Technically Accurate Conceptual Guide with Keras. Analytics Vidhya. <https://medium.com/analyticsvidhya/lstms-explained-a-complete-technically-accurate-conceptual-guidewith-keras-2a650327e8f2>
- [14] Reddit. (2020). Understanding PPO with Recurrent Policies. Reddit Reinforcement Learning Commu-

nity. https://www.reddit.com/r/reinforcementlearning/comments/gu4sjn/understanding_ppo_with_recurrent_policies/

[15] Stack Overflow. (2021). How to Use LSTM Recurrent PPO from SB3 Contrib. Stack Overflow. <https://stackoverflow.com/questions/77169707/how-to-use-lstm-recurrent-ppo-from-sb3-contrib>