Neural-Symbolic Fault Tolerant Control for Quadcopter Trajectory-Following Tasks

Yves Sohège[1] and **Marcos Quiñones-Grueiro**[2] and **Gregory Provan**[1]
[1]Computer Science and IT, University College Cork, Cork, Ireland
e-mail: {y.sohege,g.provan}@cs.ucc.ie
[2]ISIS, Vanderbilt University, Nashville, USA
e-mail: {marcos.quinones}@vanderbilt.edu

Abstract

Many fault-tolerant control (FTC) applications are moving from a traditional model-based approach to one that learns the FTC actions, termed data-driven FTC. While data-driven FTC does not require models, it requires a significant amount of nominal and fault data, and training may be expensive. We develop an architecture for combining model-based and data-driven FTC that aims to make use of the best aspects of each approach. The architecture learns a supervisory controller for switching weights across multiple model-based low level controllers. We demonstrate our approach on learning trajectories for a quadcopter that must follow a safe region even though it experiences rotor faults. We empirically show that our hybrid learning approach converges to safely follow given trajectories, whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all).

1 Introduction

Developing models for fault tolerant control of complex systems is challenging, in that model-based approaches suffer from models being incomplete and often not fully suited to real-world, dynamic environments. Data-driven approaches, on the other hand, require significant amounts of data, are time-consuming to train, and are limited to relatively simple systems. In this paper, we propose an approach that integrates model-based and data-driven control methods, in an attempt to leverage the best of both methods. We directly address how to integrate model-based inference and learning, known as neural-symbolic learning (Besold *et al.*, 2017) or model-based/physics-guided learning (Rai and Sahu, 2020).

Currently a great deal of research is directed towards this topic. The majority of work in neural-symbolic diagnosis at present focuses on using physics and deep learning in condition-monitoring of rotating machines, e.g., (Wang *et al.*, 2016); this approach is called physics-based preprocessing in (Rai and Sahu, 2020). This work uses physical principles of machinery and signal processing to assist with deep learning methods for diagnosing faults in simple machines, e.g., (Luo *et al.*, 2020; Waziralilah *et al.*, 2019).

A second approach, less common in diagnosis, is constraining deep learning with physical principles (Nautrup *et al.*, 2020; Iten *et al.*, 2020). Here, physics-based equations are used as an additional regularization term in the loss function of the neural networks. This approach has been used for diagnosis (Bunte *et al.*, 2019; Zhai *et al.*, 2016) and prognosis (Chao *et al.*, 2020).

We adopt an approach that is different to either of these methods: we decompose the fault-tolerant control (FTC) task into high-level and low-level inference, and use well-known physical controllers for the low-level control, and learning for the high-level FTC. This approach enables us to employ well-understood PID controllers with well-defined input parameters for low level control, and we then *learn to tolerate faults* using a high-level controller. Our approach is significantly simpler than methods that require learning all control parameters for FTC applications, since we reduce the number of parameters to be learned to a small number of high-level (abstracted) parameters.

We demonstrate our approach on a quadcopter that is subjected to rotor faults of various magnitudes.

Our contributions are as follows.

- We develop an architecture for combining model-based and data-driven control for FTC that aims to make use of the best aspects of each approach;
- We demonstrate our approach on learning safe trajectory following for a quadcopter that has significant faults in its rotors;
- We empirically show that our hybrid learning approach converges to safely follow given trajectories whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all).

This article is organised as follows. Section 2 describes the FTC approach from purely model-based and data-driven frameworks, and how we integrate these for a hybrid approach. Section 3 introduces the quadcopter and the faults we inject. Section 4 outlines the experiments that we perform in trajectory following given rotor faults. Section 5 presents our empirical results. We conclude in Section 6.

2 Approach

This section describes the general issues concerning hybrid learning for fault-tolerant control (FTC).

FTC is the task of using an anomalous observation $\boldsymbol{s}_t \in \mathscr{S}$ of a system at time t (indicating a fault state) to generate a control $\mathbf{a}_t \in \mathscr{A}$ that drives the system to a desired ("safe") state $\boldsymbol{s}_{t+1} \in \mathscr{S}$. We can denote this as a function $f^\theta : \boldsymbol{s}_t \times$

$\boldsymbol{\theta} \rightarrow \mathbf{a}_t$, where θ denotes the parameters of function f that drive the system to a safe state such that $f_T : \boldsymbol{s}_t \times \mathbf{a}_t \rightarrow \boldsymbol{s}_{t+1}$, with f_T representing the system dynamics also known as state transition function of a system.

Model-based FTC

The typical FTC algorithm uses a forward model to isolate a fault, and then generates a control output given that fault.

A model-based (MB) diagnosis forward model $f_T^{\boldsymbol{\theta_T}}$ maps state variables $\boldsymbol{s}_t$ and model parameters $\boldsymbol{\theta_T}$ to outputs (observable variables $\hat{\boldsymbol{y}}$): $f_T^{\boldsymbol{\theta_T}} : \boldsymbol{s}_t \times \boldsymbol{\theta_T} \rightarrow \hat{\boldsymbol{y}}_t$. Predictive modeling in a model-based approach entails calibrating model parameters $\boldsymbol{\theta_T}$ using observational data. The diagnosis process consists of using the *residual*, or difference between observed data $\boldsymbol{y}$ and predicted data $\hat{\boldsymbol{y}}$, to diagnose the fault responsible for the anomalous readings $\boldsymbol{\nu}_t$ (Blanke *et al.*, 2016).

The FTC aspect of this approach means to adapt the parameters $\boldsymbol{\theta_{MB}}$ of a control law represented by a function f_{MB} such that

$$f_{MB} : \boldsymbol{\nu}_t \times \boldsymbol{\theta_L} \rightarrow \mathbf{a}_t, \tag{1}$$

where $\boldsymbol{\nu}_t$ represents the information of the diagnosed fault.

Data-Driven FTC

We formulate our data-driven approach in terms of an agent using reinforcement learning (RL). RL is a branch of machine learning concerned with the design of methods that allow an agent to learn how to solve a task by interacting with an environment. At each time step t, the agent observes the state of the environment $\boldsymbol{s}_t \in S$ and it generates an action $\mathbf{a}_t \in A$. The agent then receives information about the next state $\boldsymbol{s}_{t+1}$ of the environment and an immediate scalar reward $r_t \in \Re$. The decision made by the agent is based on a policy function $f_{ML} : S \rightarrow A$ that maps the state space to the action space.

The environment is modeled as a Markov decision process (MDP) defined as follows:

Definition 1 (Markov Decision Process)**.** *A Markov decision process is defined by a four tuple: $M = \{S, A, T, R\}$ where S represents the set of possible states that the environment can reach. The transition function $T : S \times A \times S \rightarrow [0, 1]$ estimates the probability of reaching state $\boldsymbol{s}'$ at time $t + 1$ given that action $\mathbf{a} \in A$ was chosen in state $\boldsymbol{s} \in S$ at decision epoch t, $T = P(\mathbf{s}'|\boldsymbol{s}, \mathbf{a}) = P\{\boldsymbol{s}_{t+1} = \boldsymbol{s}'|\boldsymbol{s}_t = \boldsymbol{s}, \mathbf{a}_t = \mathbf{a}\}$. The reward function $R : S \times A \rightarrow \Re$ estimates the immediate reward $R \sim r(\boldsymbol{s}, \mathbf{a})$ obtained from choosing action $\mathbf{a}$ in state $\boldsymbol{s}$.*

A task $\mathcal{T}$ is defined in this context by the tuple

$$\mathcal{T} = (R_{\mathcal{T}}, P_{\mathcal{T}}(\boldsymbol{s}_t), P_{\mathcal{T}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \mathbf{a}_t), H) \tag{2}$$

where $R_{\mathcal{T}}$ is a reward function, H represents the duration of the task, and $P_{\mathcal{T}}(\boldsymbol{s}_t)$ and $P_{\mathcal{T}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \mathbf{a}_t)$ determine the dynamic of the system for task $\mathcal{T}$.

The goal of the agent is to learn an optimal policy function $f_{ML}^{\theta_H}$ that maximizes the expected return (the cumulative sum of rewards) for a given task $\mathcal{T}$.

$$V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) = \max_{\theta_H \in \Theta_H} V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) \ , \ \forall \boldsymbol{s}_t \in S \tag{3}$$

where $V^{f_{ML}^{\theta_H}} : S \rightarrow R$ is called value function and it is defined as

$$V^{f_{ML}^{\theta_H}}(\boldsymbol{s}_t) = E\left[\sum_{i=0}^{H} \gamma^i R(\boldsymbol{s}_i, \mathbf{a}_i)|s_0 = \boldsymbol{s}_t\right] \ , \ \forall \boldsymbol{s}_t \in S \tag{4}$$

where $0 < \gamma \leq 1$ is called the discount factor, and it determines the importance assigned to future rewards such that it decays with time.

In deep RL, we learn a neural network model $f_{ML}^{\theta_H}$ with parameters θ_H that represents the optimal policy function which solves the task $\mathcal{T}$ such that

$$f_{ML} : \boldsymbol{s}_t \times \boldsymbol{\theta_H} \rightarrow \mathbf{a}_t. \tag{5}$$

Hybrid FTC

In our hybrid approach, we develop a hybrid model that composes a model-based and a data-driven model. We use the output of f_{MB} as an input to f_{ML}, i.e.,

$$f_{Hybrid} : \boldsymbol{s}_t \times \boldsymbol{\nu}_t \times (\theta_L \times \theta_H) \rightarrow \boldsymbol{a}_t, \tag{6}$$

where $f_{Hybrid} = f_{MB} \circ f_{ML}$, and $(\theta_H \times \theta_L)$ is the composite parameter space. The machine learning component will thus encapsulate the remaining unmodeled complexity of the system in a lumped form. In this approach, the model acts in a complementary fashion to enable adequate FTC.

The key is to define a compositional approach that takes advantage of each approach. The next sections shows the architecture that we use for this decomposition.

3 Quadcopters

3.1 Quadcopter Dynamics

Quadcopters are unmanned aerial vehicles that use four propellers to maneuver and have gained increased attention in the research community in recent years. These vehicles have only four actuators used to control six variables, the coordinates x, y, and z, and the roll, pitch, and yaw angles of the quadcopter, denoted ϕ, θ, and ψ, respectively. The dynamic equations of a quadcopter are complex, due to the highly coupled state-space. Due to space limitations, we give a brief summary of quadrotor dynamics and details of how rotor faults are represented, and refer the reader to (Özbek *et al.*, 2016) for details.[1]

We define the dynamics of the quadcopter in the non-linear discrete state space form

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t) + g(\boldsymbol{s}_t)(1 - \varsigma)\mathbf{a}_t, \tag{7}$$

where $\boldsymbol{s}_t = [x \ \dot{x} \ y \ \dot{y} \ z \ \dot{z} \ \phi \ \dot{\phi} \ \theta \ \dot{\theta} \ \psi \ \dot{\psi}]^T$ is the state vector, control input $\mathbf{a}_t = [U_1 \ U_2 \ U_3 \ U_4]^T = \varrho(\upsilon_1 \ \upsilon_2 \ \upsilon_3 \ \upsilon_4)$, ϱ is a non-linear function in the angular velocity of motor i, and we denote a multiplicative fault model with parameter $0 \leq \varsigma_i \leq 1$ for $i = 1, ..., 4$, where ς_i = 0 corresponds to nominal function and ς_i = 1 to total failure.

3.2 Architecture

A quadcopter is a highly-coupled, under-actuated, nonlinear system whose control architecture can be divided into two subsystems: an attitude system and a position system. The rotational motion, also known as attitude, is independent of the position, but the translational motion is dependent on the attitude of the aircraft. Using this, we can derive the motion of the quadrotor given the position and attitude and hence define inner- and outer-control loops as the attitude and position control, respectively. The physics of a quadcopter model are well understood, and an industry-standard control

[1]The Python-based simulation codebase necessary to run the experiments in this article is available under github.com/YvesSohege/DX20-Simulation.

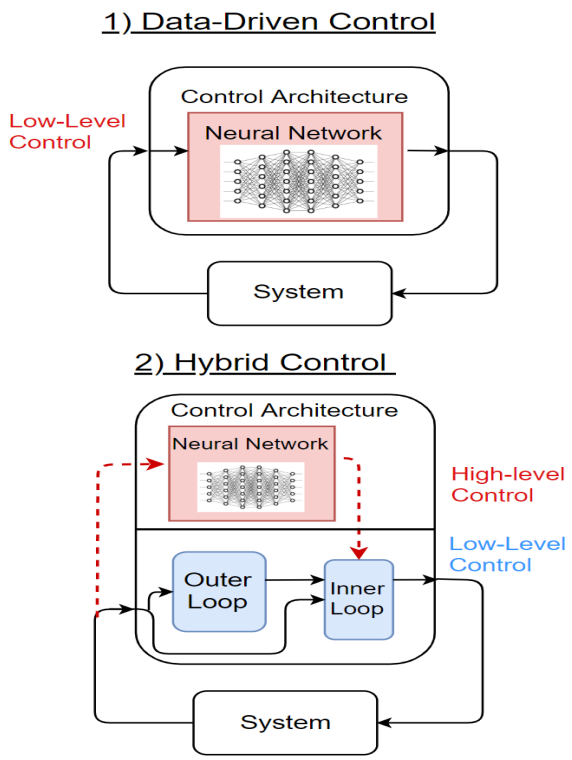

Figure 1: Two Control architectures for quadcopters showing Data-driven control (red) and Model-based control (blue) parts.

approach is to use PID or PD controllers for both low-level inner- and outer-loop control (Özbek *et al.*, 2016). Fault tolerance is typically achieved by tuning additional PID controllers and using a residual-based high-level (supervisory) controller that switches between active controllers.

Figure 1 (top) shows a data-driven architectures that can be used for quadcopter control, where a neural network learns all control parameters in a single black-box model. The issue with this learning task for complex domains is the size of the parameter space and of the data required. Figure 1 (bottom) shows a hybrid approach, where for low-level control we use a model-based PID architecture that decomposes the system into inner- and outer-loop sub-controllers, using methods common to control theory, e.g., (Xia *et al.*, 2017). A high-level data-driven controller then learns to tune the low-level architecture.

When replacing any part of the lower-level control loop with data-driven approaches, several challenges arise: (1) whereas physics based stability proofs exist to ensure real-world safety, data-driven controllers do not have such proofs of stability and stability cannot be empirically guaranteed outside of the training scenarios; (2) training is usually conducted in high-end simulation environments and then transferred to a real quadcopter, which creates a well known simulation to reality gap; (3) data-driven controllers must be re-trained when the environment changes.

To address these drawbacks, we reformulate our system as a hierarchical control architecture with a subset of high- (θ_H) and low-level (θ_L) parameters, i.e., such that $\theta = \theta_H \cup \theta_L$. We use our physics-based controller to take care of low-level control and hence only the parameters of the high-level controller must be learned. Our task is thus reduced to learning θ_H and tuning θ_L. The benefit of this approach is that (1) we can use a low-level, physics-based controller whose properties as well understood and whose parameters, θ_L, are relatively easy to tune using well known methods for different scenarios, and (2) we can learn over a significantly smaller parameter space ($|\theta_H| \ll |\theta|$) for a controller whose physics is less well understood.

3.3 Comparison of Learning Tasks

This section compares the parameter spaces of our learning tasks, i.e., purely data-driven vs. hybrid learning parameter spaces. The full state space is given by the state vector $\boldsymbol{s} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$, which is a 12-tuple. If we want to achieve trajectory tracking, we need to add additional information about the target location, i.e., $[x_{target}, y_{target}, z_{target}]$. Hence the full length vector has 15 parameters, each with a continuous-valued range of possible values.

Model-Based

In a purely model-based FTC approach that uses PID control at the low level and a supervisory controller, we must manually tune the PID controllers and the supervisory controller. Further, if we pre-define the fault controllers for the rotor faults, we must tune these controllers as well. The drawback to using specific fault controllers is the cost of tuning each of these controllers, as well as the limitation of the approach to single-fault scenarios, since it is impossible in typical control frameworks to "merge" the outputs of these controllers (Özbek *et al.*, 2016). To overcome this, a technique known as Blended Control (Provan and Sohège, 2019) which uses a convex combination of controller outputs is used in this article. However, even for this latter approach one cannot adapt to changing environmental conditions or novel faults, as one can by dynamically learning controllers. PID controllers require three gain parameters to be tuned, for which established mechanisms exist (Lunze, 2016).

Data-Driven

For the data-driven task, we must learn the direct motor commands applied to the rotors of the quadcopter, i.e $\boldsymbol{a}_t = [U_1\ U_2\ U_3\ U_4]$. We limit the speed of each of the four rotors to 10000 rpm, creating an action space of four actions with range [0-10000]. In the case of optimal attitude control, there is little tolerance and flexibility as to the sequence of control signals that will achieve the desired attitude (Koch *et al.*, 2019). For example, to achieve a stable hover all four motors much spin at exactly the same speed, which is trivial to define for model-based methods but difficult for a data-driven controller to learn due to the large continuous state and parameter space.

Hybrid

For the hybrid task, the supervisory controller uses a weighted combination of pre-defined PID controllers (Sohège *et al.*, 2020). This Randomized Blended Control (RBC) architecture samples the blending weights used for the convex controller combination from a probability distribution (Provan and Sohège, 2019). The learning task in the hybrid approach is to *learn an optimal probability distribution* for RBC which can be defined by mean and standard deviation, $\theta_H = [\mu, \sigma]$. This reduces the size of the action space of the agent to two parameters with range [0,1].

We tune the parameters for the low-level model-based controllers, θ_L, using standard mechanisms prior to training the high-level controller.

Learning randomized blended control is thus a significantly smaller problem (in terms of parameter space) than data-driven methods; it also has an inherently safe action space, i.e., there is no way the agent's actions can crash the quadcopter unless one of the low-level controllers performs an unsafe action and even then there is only a small random chance that this action is fully selected by the high-level controller.

4 Hybrid Quadcopter Control

In this section, we will describe the implementation of the presented approach on top of an open source Python-based Quadcopter simulation (Majumdar, 2018). The description will be broken down into the model-based low-level control architecture, the learning-based high-level controller and how the two components integrate together through randomized blended control.

4.1 Model-based Low-Level Control

Researchers have successfully learnt the dynamics model of quadcopters through DRL but they are not comparable to traditional controllers yet (Becker-Ehmck *et al.*, 2020; Koch *et al.*, 2019). Our decomposition of the FTC task enables the integration of learning-based low-level control mechanisms into our approach, once they become a competitive solutions. For nominal trajectory tracking only a single roll and pitch controller is needed. Fault tolerance is provided by additional PID controllers tuned for general fault conditions such as a high-gain controller that responds more aggressively to wind disturbances or rotor faults. In this article, we focus on roll and pitch attitude control and hence add a high-gain roll and pitch PID controller for more aggressive maneuvers. For clarity we referred to this controller as C1 and for the nominal conditions (lower-gain) controller as C2.

Axis of control	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
X-Position	300	0.04	450
Y-Position	300	0.04	450
Z-Position	7000	4.5	5000
Roll - C1	24000	0	12000
Pitch - C1	24000	0	12000
Roll - C2	4000	0	1500
Pitch - C2	4000	0	1500
Yaw	1500	1.2	0

Table 1: PID parameters for low-level model-based control architecture.

4.2 Learning-based High-Level Control

We use an actor-critic neural network implemented using the stable-baselines framework (Hill *et al.*, 2018). This network consists of two hidden layers of 64 neurons. The observation space of the agent consists of a subset of the state variables as well as the target position, namely $[x\ y\ z\ \phi\ \theta\ \psi\ x_{target}\ y_{target}\ z_{target}]$. Quadcopter FTC requires real-time actions from the high-level controller, as any delays in action during a fault could, for example, cause a crash. This direct control approach avoids delays associated with fault isolation in traditional FDI methods.

4.3 Randomized Blended Control

RBC draws the weighting vector from a probability distribution defined by mean μ and standard deviation σ at each cycle of the control loop. In this work the high-level controller learns the parameters defining the underlying probability distribution used for RBC in real-time as faults occur. RBC has been shown to be stochastically stable (Provan and Sohège, 2019), so flight stability is ensured even when it is learning to deal with non-terminal faults, i.e., faults for which no control is viable.

4.4 Training Details

We enforce a 1 meter *safety region* around the trajectory to evaluate control performance under fault scenarios. The task of the agent is to learn to distribute control in such a way that the quadcopter stays inside the safety region when experiencing faults. A negative reward is applied when the quadcopter drifts outside the safety region and a large positive reward is received when the quadcopter completes the entire trajectory inside the safety region. The trajectories used for training need to be diverse enough to expose the agent to a large variety of experiences. In this work, we use straight-line trajectories to a randomized destination point within a 7-meter bounding box. Proximal Policy Optimization (PPO) is selected as the training algorithm, with a learning rate of 0.1.[2] The agent was trained for a total of 10 million time steps, which is approximately 12000 episodes.

Fault Generation

Rotor faults are one of the most common faults experienced by quadcopters and the focus in this work. We investigate faults in any rotor and up to 30% loss of effectiveness (LOE). Greater than 30% faults will cause a crash, and 20% causes severe disruption to the flight path but is marginally controllable by the higher-gain controller. Instead of exposing the agent to the entire fault space randomly, we systematically divide the fault space into *levels* representing increments of 5% in the fault magnitude. We increase the fault level when the agent does not leave the safe zone for 20 consecutive episodes. The reward obtained for completing a trajectory is proportional to the level. Hence, as the agent progresses to higher fault magnitudes the reward also increases as seen in Fig. 2, where each data point shows the cumulative reward over the last 20 episodes. However, larger faults create more severe disturbances which is why episodes fail towards the end of the training cycle. Since the faults are of the same type but varying magnitudes, information learned on lower levels is relevant on higher levels. After 6000 episodes the mean cumulative rewards stops increasing showing that the agent converged.

4.5 Data-Driven Quadcopter Controller

To date, fully data-driven controllers are restricted to simple tasks, e.g., learning how to hover a quadcopter and simple trajectory tracking to a location (Becker-Ehmck *et al.*, 2020). Due to the lack of guarantees of stability (as exists for many model-based controllers), there exist no guarantees on the actions a learned controller will perform when presented with novel faults or unmodelled dynamics, as we do in our experiments.

[2]Fine-tuning the parameters of a learning algorithm is a challenging task which will be investigated in future work.

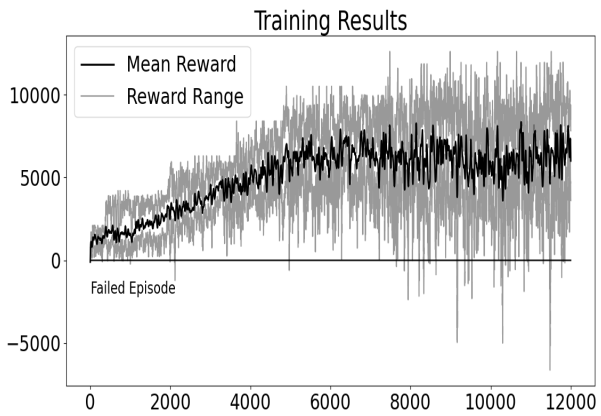

Figure 2: Mean Cumulative Reward obtained over 5 independent training cycles of 12000 episodes. Scores below 0 indicate the quadcopter left the safe zone during the episode.

5 Experiments

This section compares the performance of the trained agent with that of (a) the high-gain and low-gain PID controllers alone as well as (b) a uniform randomized blended controller. Intuitively, we expect the agent to outperform the uniform randomized approach, since this would show that the agent has learned an improved probability distribution.

5.1 Experimental Design

To validate these hypotheses, we ran experiments on a test trajectory, a 5-meter diamond, which is significantly more difficult than the straight line training trajectories. Faults at two separate time points are investigated, $T1 = 1800$ and $T2 = 2300$. We test faults of magnitude 0%, 10%, 20% and 30% , where 0% indicates nominal control, on all rotors at each time point. As a performance metric we use the **total time outside the safety bound** so the smaller the better. Figures 3 shows the summarized experimental results.

5.2 Parameter Spaces

Data-driven approaches typically use all of the parameters available as observation and generate the four motor commands as the action. The minimum observation vector needed is the full state vector of size 15 and most approaches use additional parameters (Hwangbo *et al.*, 2017). The hybrid approach only requires a small subset of parameters that are specific to the task since flight control is already established. We use an observation space of 9 parameters and an action space of 2 parameters creating a smaller problem compared to the purely data-driven approach.

5.3 Trajectory Following Results

Hybrid Approach

We will now discuss the results presented in Figure 3. There are four groups of bars for each chart representing one fault level. Each bar represents a different controller C1, C2, RBC and the trained hybrid approach, on that fault level. Each coloured stack in a bar represents which rotor experienced the fault.

The *left-most bar group* in the chart represent nominal conditions (0%). As expected, the bars in this group are identical, as no fault occurs. This also shows that the smooth

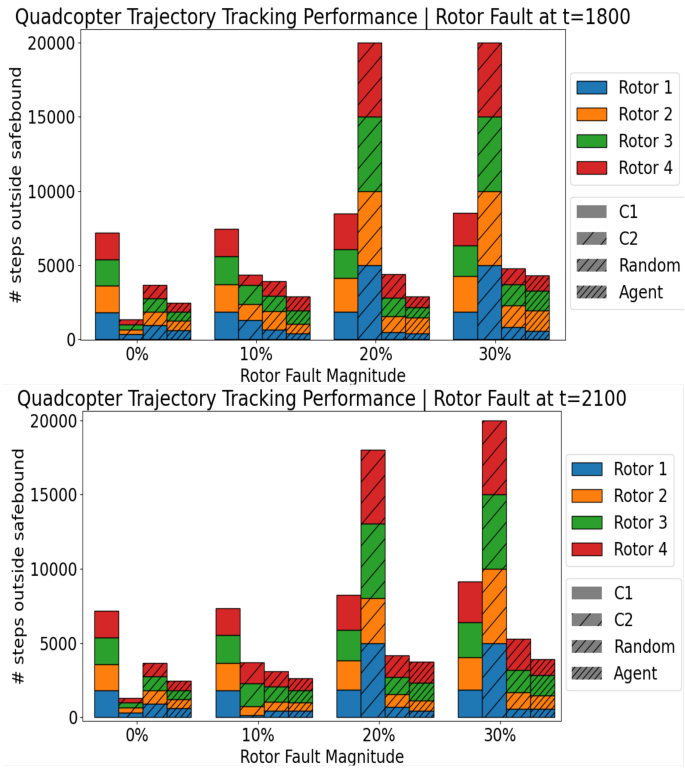

Figure 3: Experimental results showing the amount of time a quadcopter was outside of the safety bound for rotor faults of varying magnitudes. Lower is better.

controller (C2) is much better under nominal conditions as the high-gain controller naturally overshoots the waypoints. This makes C1 undesirable as a controller under nominal conditions. However as the fault magnitudes increase we see that C2's performance drastically degrades. This is because the controller cannot respond aggressively enough to correct the effect of the higher rotor fault. The high-gain controller is much more robust to rotor faults as the naturally aggressive reactions can mitigate rotor faults quickly and performance is less affected.

The *third bar* in each bar group represents random blended control, which samples from a uniform probability distribution over the controllers. Uniform randomized blended control outperforms both C1 and C2 on fault magnitudes of 10% or higher. This shows how randomization is able to utilize benefits from both controllers.

The *fourth (right-most) bar* in each group represents our approach which significantly outperforms all other controllers under all fault levels. The agent is able to very quickly shift control to the aggressive controller to mitigate the fault and then continue using the nominal controller when the system stabilizes. This allows our approach to provide good control under faults that would otherwise crash the system. Under nominal conditions our approach performs comparably to the better-performing individual controller. This shows that the agent was successfully able to learn how to parameterize the underlying probability distribution used for RBC, and thus provides an exciting new way to integrate model-based low-level control frameworks with high level data-driven controllers.

Data-Driven Approach

We implemented a learning-based controller where a neural network generates four motor throttle commands, which are applied directly. The full state vector and target location are set as the observation vector. Even when we used a simpler learned trajectory, a simple hover at a given altitude, and increased the training length to 50 million steps ($5\times$ as much as our approach, or 48 hours CPU-time), *the agent was unable to converge*. There are several possible reasons for this: (1) Parameter fine-tuning is needed such as the learn rate, neural network size, etc.; (2) Reward shaping - a well known open problem for complex control tasks; (3) More training time/better hardware is needed.

6 Conclusion

We compared a purely data-driven FTC approach with a neural-symbolic approach that uses an architectural integration of data-driven and model-based FTC. The architecture uses models for low-level controllers, and learns a supervisory controller for switching weights across multiple controllers. We demonstrate our approach on learning trajectory following for a quadcopter that follows a safe region even through it experiences faults in its rotors. We empirically show that our hybrid learning approach converges to safely follow given trajectories, whereas a purely data-driven approach requires significantly more training to converge than the hybrid approach (if it converges at all). The parameter-space for the data-driven approach is significantly larger than that of the hybrid approach, resulting in this increased training challenge. Further, the hybrid approach inherits the stability guarantees of the model-based component, whereas there are no similar guarantees for the data-driven approach.

References

Philip Becker-Ehmck, Maximilian Karl, Jan Peters, and Patrick van der Smagt. Learning to fly via deep model-based reinforcement learning. 2020.

Tarek R. Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kuehnberger, and Luis C. Lamb. Neural-symbolic learning and reasoning: A survey and interpretation, 2017.

Mogens Blanke, Michel Kinnaert, Jan Lunze, and Marcel Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer, 2016.

Andreas Bunte, Benno Stein, and Oliver Niggemann. Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2727–2735, 2019.

Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Fusing physics-based and deep learning models for prognostics. *arXiv preprint arXiv:2003.00732*, 2020.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, and Gleave. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

Raban Iten, Tony Metger, Henrik Wilming, Lídia del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical Review Letters*, 124(1), Jan 2020.

William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for UAV attitude control. 2019.

Jan Lunze. From fault diagnosis to reconfigurable control: A unified concept. In *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 413–421. IEEE, 2016.

Yuan Luo, Ya Xiao, Long Cheng, Guojun Peng, and Danfeng Daphne Yao. Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities. *arXiv preprint arXiv:2003.13213*, 2020.

Abhijit Majumdar. *Python Quadcopter Simulation*, 2018.

Hendrik Poulsen Nautrup, Tony Metger, Raban Iten, Sofiene Jerbi, Lea M. Trenkwalder, Henrik Wilming, Hans J. Briegel, and Renato Renner. Operationally meaningful representations of physical systems in neural networks, 2020.

Necdet Sinan Özbek, Mert Önkol, and Mehmet Önder Efe. Feedback control strategies for quadrotor-type aerial robots: a survey. *Transactions of the Institute of Measurement and Control*, 38(5):529–554, 2016.

Gregory Provan and Yves Sohège. Fault-tolerant control for unseen faults using randomized methods. In *IEEE SysToL*, 2019.

Rahul Rai and Chandan K Sahu. Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. *IEEE Access*, 8:71050–71073, 2020.

Yves Sohège, Gregory Provan, Marcos Quinones-Grueiro, and Gautam Biswas. Deep reinforcement learning and randomized blending for control under novel disturbances. In *IFAC World Congress*, 2020.

Jinjiang Wang, Junfei Zhuang, Lixiang Duan, and Weidong Cheng. A multi-scale convolution neural network for featureless fault diagnosis. In *2016 International Symposium on Flexible Automation (ISFA)*, pages 65–70. IEEE, 2016.

N Fathiah Waziralilah, Aminudin Abu, MH Lim, Lee Kee Quen, and Ahmed Elfakharany. A review on convolutional neural network in bearing fault diagnosis. In *MATEC Web of Conferences*, volume 255, page 06002. EDP Sciences, 2019.

Dunzhu Xia, Limei Cheng, and Yanhong Yao. A robust inner and outer loop control method for trajectory tracking of a quadrotor. *Sensors*, 17(9):2147, 2017.

Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. Deep structured energy based models for anomaly detection, 2016.