Unknown Fault Tolerant Control using Deep Reinforcement Learning: A blended control approach

Yves Sohège[1 *], **Marcos Quiñones-Grueiro** [2], **Gregory Provan**[3]

[1,3]Insight-Centre for Data Analytics, University College Cork, Cork, Ireland

[2]Vanderbilt University, Tennessee, USA

yves.sohege@insight-centre.org, marcosqg88@gmail.com, g.provan@cs.ucc.ie

Abstract

It is impossible to pre-define a controller for every fault an autonomous system can experience as some faults are unknown at design time. Current fault tolerant control (FTC) architectures switch control to a pre-defined fault controller when a known fault is identified. *Blended control* implements a controller that is composed of multiple individual controllers instead of discretely switching between them. In this article we present a novel fault tolerant control architecture based on blended control that uses a high-level deep learning agent to learn the optimal blending proportions between low-level controllers for unknown faults. Faults are abstracted to the effect they have on the performance of a task while removing the inherent fault identification delays experienced by existing FTC architectures. The presented architecture is validated on a quadcopter trajectory tracking task and trained to tolerate abrupt rotor loss of effectiveness. We compare our approach against a switched architecture with the same underlying controllers and show its ability to learn unknown fault tolerance.

1 Introduction

Autonomous systems have been a major focus for the fault tolerant control community in recent years. For the application of autonomy to large scale systems they need to tolerate unknown faults. *Fault tolerance* can be defined as continuing mission performance under any fault conditions, known or unknown. Traditional switched fault tolerant control (FTC) systems rely on prior knowledge of the effects a fault has on the system dynamics for the fault detection and isolation unit (FDI) to identify the fault as well as a controller to operate under these conditions [Blanke *et al.*, 2016]. After identification Control is switched to the pre-defined fault controller. This fundamentally does not extend to unknown faults for two reasons:

1. Identification relies on prior knowledge of the effect a fault has on the system dynamics.
2. Fault controllers are designed to operate under known fault conditions only.

Identifying the effect a fault has on the system dynamics is complex and has an inherent time delay. For highly unstable systems, such as quadcopters, such a delay can be catastrophic. Blended Control is a variation of switching control that implements a control that is composed of multiple low-level controllers simultaneously instead of discretely switching between them.

In this article we present a novel hierarchical FTC framework based on blended control and a deep learning high-level controller that addresses the mentioned problems. The FDI unit and control switching function are replaced by a deep learning agent, specifically a deep deterministic policy gradient (*DDPG*) agent. This is an abstracted approach to learn the effect of a fault on the task performance rather than the system dynamics. Degrading task performance due to any faults is optimized through changing the blend weight vector which removes the need for prior knowledge of a fault and addresses problem 1. The low-level controllers are designed based on the *type* of reaction to a fault instead of for pre-defined fault conditions while providing similar control responses under nominal conditions. This allows for synthesis of new controllers for unknown faults by adapting the blend weight vector and addresses problem 2. The presented architecture provides a novel integration of existing controllers to deep learning FTC and is validated on a Quadcopter trajectory tracking task with abrupt rotor loss of effectiveness. We show the presented architecture is able to track a given trajectory closer than a switched architecture based on the same low-level controllers under rotor loss of effectiveness of 50% while also generating a improved control signal.

Our contributions are as follows:

- We present a novel hierarchical fault tolerant control architecture with the ability to learn unknown fault tolerance through blended control and a high-level deep learning agent.
- The architecture is validated on a quadcopter trajectory tracking task under unknown rotor loss of effectiveness faults. The trained controller shows robustness to the trained faults and exhibits less oscillations around the reference signal than the low-level controllers.

The remainder of this article is structured as follows: Sec-

*Contact Author

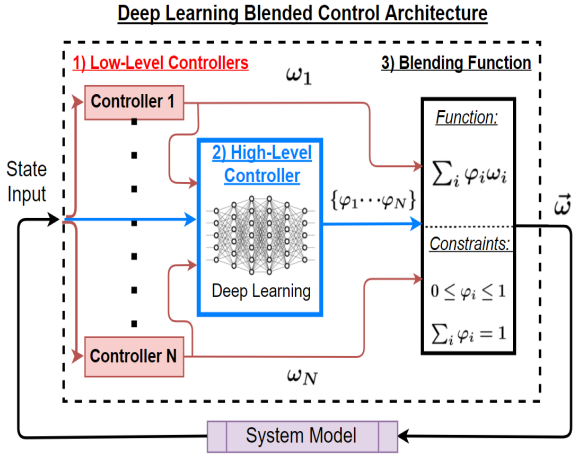

Figure 1: Deep Learning Blended Control architecture showing the 3 main parts: Low-level controllers, High-Level Controller and Blending Function.

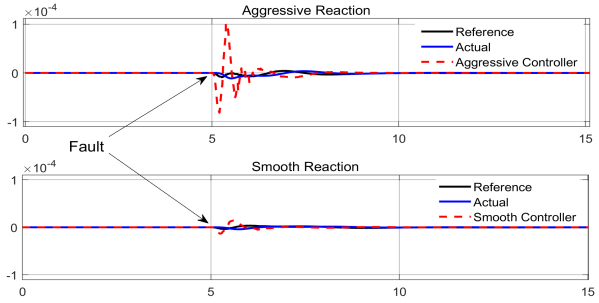

Figure 2: Quadcopter attitude controller outputs for trajectory tracking under rotor fault. Y-Axis represents Reference and Actual position as well as controller response in red, Aggressive (Top) and Smooth (Bottom).

tion 2 gives a detailed overview of the presented architecture. We compare our architecture to current state-of-the-art FTC architectures and applications of deep learning for control in Section 3. We discuss deep reinforcement learning and its application to control in Section 4. The implementation and training on a quadcopter simulation is given in Section 5 followed by experimental validation in Section 6.

2 Deep Reinforcement Learning Blended Control

We will firstly give a detailed overview of the presented architecture, referred to as Deep Reinforcement Learning Blended Control (**DRLBC**), to improve the overall clarity of the article. An architecture diagram of DRLBC can be seen in Figure 1. The framework can be broken down into three parts which will be discussed in detail:

1. Low-Level Controllers
2. High-Level Controller
3. Blending Function

2.1 Low-Level Controllers

In hierarchical control architectures the low-level controllers generate the control signals that are directly applied to the systems actuators (motors, valves, etc). Several types of well known controllers exist for system control such as Proportional Integral Derivative (PID), Model-Preditive Controllers (MPC) or Linear Quadratic Regulators (LQR) to name a few. In this article we will restrict the low-level controllers to PID controllers which are an industry standard way of controlling automatic systems but the presented framework works with any low-level controllers. Traditionally an optimal controller is designed offline for a system model under predefined operating conditions. Fault tolerance is achieved through redundant controllers designed for known fault conditions. The number of faults a system is tolerant for depends on the number of low-level controllers predefined at design time. In this article, the set of low-level controllers are not designed for specific operating conditions but based on the *type* of reaction to a disturbance or fault and are denoted $\Omega = \{\omega_1, \cdots, \omega_N\}$. Figure 2 shows the reaction of two differently tuned PID controllers to a fault. Gain parameters for these can be found in Table 1. Higher gain parameters cause the controller to have a more *aggressive* response to a fault and performs better for aggressive flight maneuvers (top) while smaller gain parameters have a *smoother* response which results in more precise control but slower stabilization after faults (bottom). The blue line representing the actual state variable that is manipulated stabilizes on the black reference line for both controllers. The controllers reactions shown in Figure 2 are used in the quadcopter simulation experiments and will be elaborated on further in Section 5.1.

2.2 High-Level Controller

In traditional switched architectures the high-level controller contains the FDI unit which identifies the effect of faults on the system. The high-level controller selects which of the low-level controllers is active at any time and provides fault tolerance capabilities by switching control to the predefined controller once a known fault is identified. In the presented architecture the high-level controller is implemented with a deep neural network instead of a set of state observers that identify faults. In this article we will use a Deep Deterministic Policy Gradient network [Lillicrap *et al.*, 2015] which will be described in more detail in Section 4. The low-level controller outputs, Ω , and a subset of state variables representing the performance on a task are used as an input for the high-level controller. The objective is to learn the optimal blend of low-level controllers to maximise the systems performance on a task. Faults that impact the performance of the system can be mitigated generically in real time without having to define specific fault observers. The output of the high-level controller is defined as the blend weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$ that specifies the weight of each low-level controllers in the blended control signal $\vec{\omega}$ applied to the system.

2.3 Blending Function

The blending function takes as inputs the low-level controller outputs $\{\omega_1, \cdots, \omega_N\}$ and the blend weight vector $\{\varphi_1, \cdots, \varphi_N\}$ and outputs a blended control signal $\vec{\omega}$. Formally blended control can be defined as follow:

Definition 1 (Blended Control) *Given a collection of controllers $\Omega = \{\omega_1, \cdots, \omega_N\}$ and a blend weight vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_N\}$, blended control is a weighted combination $\vec{\omega} = \sum_i \varphi_i \omega_i$ such that: (1) $\forall_i, \omega_i \in \Omega, 0 \leq \varphi_i \leq 1$, and (2) $\sum_i \varphi_i = 1$*

The two constraints imposed on the blend weight vector ensure that $\vec{\omega}$ is bound by the low-level controller outputs. Further, if the low-level controllers output the same control signal, blending to any proportions will have no effect. In Figure 1 these constraints are shown in the blending function block for clarity but can be imposed on the high-level controller outputs for a simpler implementation.

3 Running Example & Related Work

We will use the quadcopter with abrupt rotor loss of effectiveness (LOE) as a running example for the remainder of this article. 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 fewer actuators than degrees of freedom, and hence are called under-actuated: only four actuators (propellers) are 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. Hierarchical PID-based control is a standard way to control quadcopters. The dynamical 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.

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

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})(1 - \varsigma)\boldsymbol{u}(t), \tag{1}$$

where $\boldsymbol{x} = [x\ \dot{x}\ y\ \dot{y}\ z\ \dot{z}\ \phi\ \dot{\phi}\ \theta\ \dot{\theta}\ \psi\ \dot{\psi}]^T$ is the state vector, control input $\boldsymbol{u}(t) = [U_1\ U_2\ U_3\ U_4]^T = \varrho(\upsilon_1\ \ \upsilon_2\ \ \upsilon_3\ \ \upsilon_4)$, 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.

Quadcopters have been shown to be able to maintain flight even after the complete loss of one or more rotors [Mueller and D'Andrea, 2014]. Several applications of deep learning for quadcopters exist which mostly focus on learning the direct control mapping of state space to motor commands [Hwangbo *et al.*, 2017; Greatwood and Richards, 2019; Koch *et al.*, 2019]. This usually requires large amount of training data and complex fine tuning of the reward functions to accurately represent the desired behaviour.

The application of deep learning for FTC of a quadcopter has been achieved by learning a complementary controller that adjusts the nominal controller output during a rotor fault [Fei *et al.*, 2019]. The success of this approach compared to other adaptive control strategies is attributed to the continuous output of the neural network and removal of the FDI unit to identify when a correction to the nominal controller is needed.

	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
Aggressive	12	5	5
Smooth	5	1.1	3

Table 1: PID parameters for aggressive and smooth reactions to faults from a quadcopter simulation.

A large amount of work has been done on FTC through hierarchical control architectures [Blanke *et al.*, 2016; Lunze, 2016]. An extension to the traditional switched architectures is blended control. Blended control has been proven to be safe as the applied control signal will always be bound by the underlying controllers and will at worst perform like a switched architecture [Kuipers and Ioannou, 2010]. Blended control has been successfully applied for FTC of a quadcopter with partial rotor failure in [Büyükkabasakal *et al.*, 2017] but has received little success outside of this due to the complexity of generating adequate blending weights.

3.1 Comparison to DRLBC

We contrast the presented DRLBC architecture to current state of the art approaches. The deep learning algorithm is applied to an abstracted high-level task while relying on existing control mechanism to control the vehicle. This reduces the overall complexity of the learning task as nominal system control is already established which usually takes a large number of learning iterations and fine tuning an accurate reward function to achieve. The direct mapping of state space to motor commands is a highly complex function. Every additional input to the deep learning algorithm increases the size of the space that is explored which is a reason for the long convergence times experienced by the application of deep learning to control tasks. DRLBC only uses a subset of the state space variables as an input which reduces this space drastically. If the underlying controllers provide safe responses with varying performance on a task, DRLBC guarantees a safe exploration space for the agent during training as any blend weight vector used will create a control signal bound by the safe controller outputs.

The presented architecture is able to generically identify degrading performance since no fault specific observers are used in the high-level controller. Faults have an effect on the performance an autonomous system is able to achieve on its task. For example abrupt rotor LOE on a quadcopter will cause overall instability and reduce the trajectory tracking accuracy the quadcopter is able to achieve. By choosing inputs for the high-level controller that represent the performance on a given task the deep learning agent is able to identify when system performance degrades and learns to optimize this through the selection of an adequate blend weight vector. For a trajectory tracking task such a performance measure could be the current trajectory tracking error as it captures the overall task the system is executing. To be able to distinguish the effects different faults have on the overall system

we extend the set of high-level controller inputs with a subset of the state variables that represent the changing conditions of the system. The angular state of a quadcopter is heavily effected by rotor faults. This would be an adequate input for the high-level controller to learn the effect a rotor fault has on the system performance without specific observers monitoring these states. This abstraction allows the system to learn how to maintain control during unknown faults or even when the fault is not identifiable from any state variables.

With specifically tuned fault controllers blended control is mostly limited to the correction of partial faults. The calculations of the correct blending weights for partial fault control is complex as the FDI observers need to be able to identify the magnitude of the fault and then calculate the adequate blend weights. DRLBC is not limited to partial fault tolerance as the low-level controllers are not tuned to be fault specific. Blending low-level controllers based on the type of control that is required allows for the deep learning agent to synthesise a new controller when system performance degrades due to any fault. The complexity of defining the blending weights is left to the deep learning agent. Theoretically this framework can learn fault tolerance on-line given an adequate training environment but this is beyond the scope of this article.

4 Deep Reinforcement Learning (DRL)

The standard setup for reinforcement learning is a decision maker called agent that interacts with an unknown environment E in discrete time steps for achieving a goal. The information exchanged between the agent and its environment is reduced to three signals: one signal to represent the choices made by the agent $a_t \in \Re^N$ (the actions), one signal to represent the basis on which the choices are made $x_t \in \Re^M$ (the observations), and one scalar signal that represents the agent's goal $r_t \in \Re$ (the reward) [Sutton and Barto, 1998]. Here, we assumed the environment is partially-observed ($x_t = s_t$ where s_t is the state vector).

The agent makes a decision based on a policy π that maps the states to a probability distribution over the actions $\pi : S \rightarrow P(A)$. The environment is modeled as a Markov decision process defined by a four tuple: $\{S, A, T, R\}$. The transition function $T : S \times A \times S \rightarrow [0, 1]$ allows to estimate the probability of reaching state s' at $t + 1$ given that action $a \in A$ was chosen in state $s \in S$ at time t, $p(s'|s, a) = Pr\{s_{t+1} = s'|s_t = s, a_t = a\}$. The reward function estimates the immediate reward $R \sim r(s, a)$ obtained from choosing action a in state s.

The goal of the agent is to learn a policy that maximizes the future discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ over a time period T without explicit knowledge about the shape of the reward or the dynamics of the environment. Therefore, solving a reinforcement learning problem means, roughly, finding the policy function that maximizes the expected reward over the long run. One approach to find the best policy is to derive it from the so-called action-value function that approximates the expected reward for any state and action pair. Hence, the optimal action-value function Q must be learned

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{r_{i \geq t}, s_{i>t}, a_{i>t} \sim \pi}[R_t | s_t, a_t] \qquad (2)$$

Deep neural networks have been successfully used as function approximators for learning the optimal action-value function. The Deep Q Network (DQN) algorithm was first proposed to deal with continuous state spaces [Mnih *et al.*, 2015]. However, DQN can only handle low-dimensional action spaces mainly because of the curse of dimensionality. Then, Lillicrap et al. (2015) proposed an off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces [Lillicrap *et al.*, 2015]. Their algorithm, called Deep Deterministic Policy Gradient (DDPG), promotes stability and efficiency by training the network off-policy with samples from a replay buffer and using a target Q network to give consistent targets during temporal difference backups.

Learning a set of deep neural networks for direct fault-tolerant control of a quadrotor is a complex problem. The two main challenges are the design of an appropriate reward function and the time-consuming exploration process required given the high-dimensional action and observation space required. We leave a deeper investigation into these challenges for future work. The goal of using reinforcement learning in this work is to design an agent that maps from an observation vector to the optimal blended weight vector depending on the system state.

5 Quadcopter Implementation and Training

Hierarchical PID-based control is a standard way to achieve quadcopter trajectory tracking. A trajectory is a temporally-indexed set of coordinates in 2D or 3D, denoted $\boldsymbol{\zeta}(k)$. We denote the reference (desired) trajectory as $\boldsymbol{\zeta}^R(k)$, and the executed trajectory as $\tilde{\boldsymbol{\zeta}}(k)$. The goal of a trajectory tracking task can be defined as minimizing the Trajectory Loss.

Definition 2 (Total Trajectory Loss) *We can represent the total trajectory loss as a difference function between reference and executed trajectories, i.e.,* $\mathcal{L}_{0:T} = \sum_{k=0}^{T} \parallel \boldsymbol{\zeta}^R(k) - \tilde{\boldsymbol{\zeta}}(k) \parallel$ *for a trajectory over time points* $k = 0, \cdots, T$.

For simplicity we will focus on 2D (x, y) trajectory tracking. Figure 3 shows the full architecture diagram implemented on the quadcopter which will be discussed similarly to Section 2, omitting the blending function details as they do not change.

5.1 Low-Level Controllers

A PID controller for each x and y axis generates the desired Roll (ϕ) and Pitch (θ) reference angular state needed for the quadcopters to execute the desired trajectory. The low-level roll and pitch controllers translate the desired angular states into motor throttle commands which are applied to the vehicle. We disregard ψ, the rotational state of the quadcopter, as it is not relevant for the 2D position. By changing the gain parameters of the roll and pitch PID controllers we change how the quadcopter responds to a divergence of the reference trajectory. To achieve a blended control architecture at least one extra controller is needed for each of the angular states for which the control is to be blended. As previously mentioned an *aggressive* and a *smooth* controller, referred to as C1 and C2 respectively, are used and for simplicity both roll

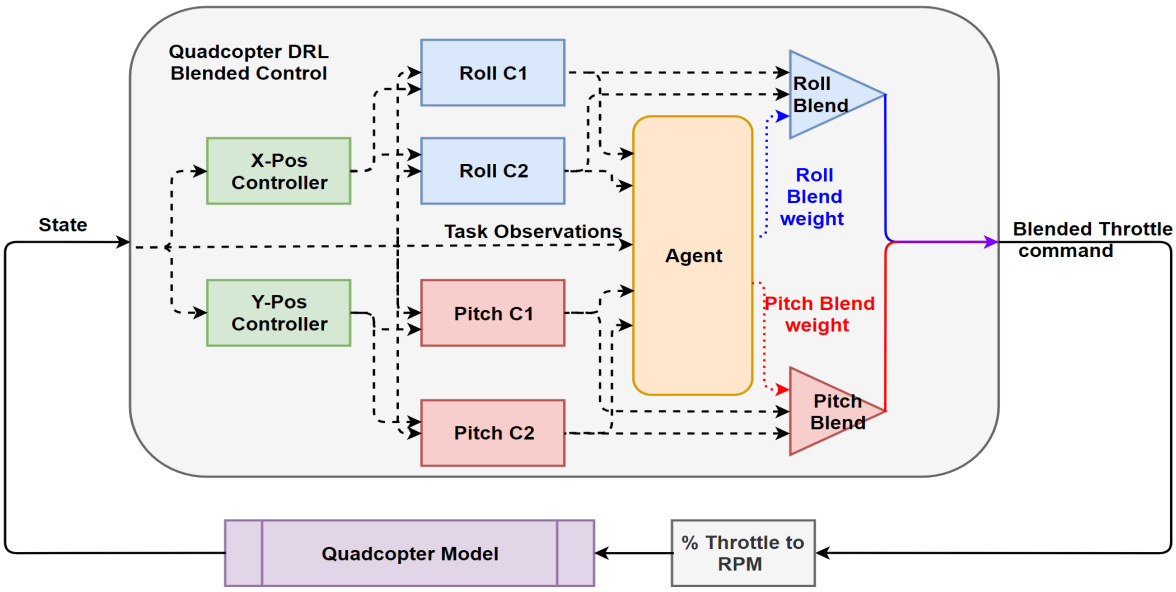

Figure 3: Quadcopter Deep Reinforcement Learning Blended Control architecture.

and pitch use the same controller tuning. Table 1 show the detailed gain parameters used and Figure 2 shows the difference in response to an abrupt change in position due to some unknown fault.

5.2 High-Level Controller

A standard actor-critic DDPG network structure is used to generate the blend weight vector. Both actor and critic take in the same observation vector which we define as:

$$[\phi \ \theta \ \delta_X \ \delta_Y \ C1_\phi \ C2_\phi \ C1_\theta \ C2_\theta]$$

where ϕ and θ are the current angular states, δ_X and δ_Y represent the current trajectory tracking error and the remainder the low-level controller outputs of C1 and C2 for ϕ and θ respectively.

Since there are only two controllers for each control axis being blended and the second blended control constraint from Definition 1 enforces $\sum_i \varphi_i = 1$, the actor output can be defined simply as : [φ_ϕ φ_θ]. The full blend weight vector $\boldsymbol{\varphi}$ can then be calculated as :

$$[\varphi_\phi \ (1-\varphi_\phi) \ \varphi_\theta \ (1-\varphi_\theta)]$$

This reduces the neural network output to one variable per control axis compared to current state of the art approaches learning the direct control mapping of *all* control signals needed to control the system. We give a brief overview of the network architecture but details are omitted due to space restrictions. The standard DDPG network was used without special modifications. The actor network is defined by three fully connected layers separated by ReLU (Rectified Linear Unit) layers. Finally a hyperbolic tangent layer with output size 2 is used to naturally enforce the blended control constraints, bounding φ_ϕ and φ_θ between 0 and 1. The critic network has two paths, one for the observation vector and the other for the actor output which are joined after two and one fully connected layer respectively for each path. All fully connected layer contains 32 neurons in this implementation.

5.3 Training Details

For training we use a simple sinusoidal reference path of 10 meters for the X and Y position executed over 15 seconds. We define the performance of the quadcopter on the trajectory tracking task as the **average trajectory loss** over the 15s simulation. The performance of C1 and C2 under nominal conditions is 48.88cm and 48.86cm respectively. The quadcopter was trained over 3000 episodes and we define other relevant training parameters used in Table 2.

Parameter	Value
Discount factor	0.99
Initial learning rate of the critic	0.01
Initial learning rate of the actor	0.025
Batch size	5
Replay buffer size	5
Training steps of an episode	150
Number of episodes	3000

Table 2: Training parameters used

Rotor Fault Generation

We use abrupt rotor loss of effectiveness as the fault to learn which has briefly been defined in the quadcopter model in Equation 1. For the purposes of training we extend the definition of the rotor faults to a triplet:

$$[\varsigma \ \gamma \ t]$$

where ς defines fault magnitude, γ discretely selects the rotor and t the time of occurrence. We define the sampling

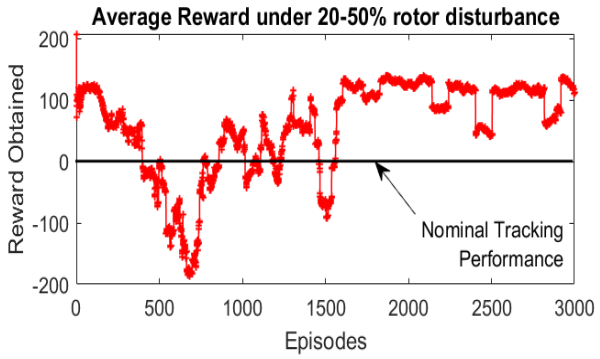

Figure 4: Average reward obtained over 3000 episodes shown against nominal tracking performance.

interval $0.2 < \varsigma < 0.5$ indicating a loss of angular rotor velocity of 20-50%. Each fault parameter is sampled randomly to provide a varied set of training data and the probability of a fault occurring at time t is set to 5%. Each fault is applied for 0.1s which is one time step.

Reward function

Nominal control is already established through the low-level controllers and the objective of the agent is to learn to tolerate any fault by maintaining nominal task performance. The total trajectory loss over the training path under nominal conditions, defined $\mathcal{L}^N_{0:T}$, makes for a natural baseline to compare the performance achieved under fault conditions against.

Definition 3 (Trajectory Tracking Reward) *Given total trajectory loss under faults $\mathcal{L}^F_{0:T}$, the obtained reward is defined as:*

$$\mathcal{R}_{0:T} = \mathcal{L}^N_{0:T} - \mathcal{L}^F_{0:T}$$

for time points $k = 0, \cdots, T$.

where $\mathcal{L}^N_{0:T}$ is the average total trajectory loss of C1 and C2 under nominal conditions. The reward function is defined independently of the faults applied to the system which make it extensible for any fault that effects the trajectory, for example wind disturbances, but this is beyond the scope of this article. The reward function allows for a positive and negative rewards indicating improved or degraded system performance which training should maximise.

5.4 Training Results

Figure 4 shows the average reward obtained by the agent over 3000 episodes calculated over 100 training episodes. This figure shows that the reward stabilizes to a positive value indicating that the agent performed better than nominal control. We partially attribute this to the way the reward is calculated. During rotor faults the quadcopter can actually move closer to the reference trajectory due to the steady state error experienced during tracking. This has a positive effect on the overall trajectory loss. Additional factors for this are explored during the experiments section. Since the average reward is strictly positive after 1500 episodes we reason the agent has learned to stabilize the fault correctly without having a dedicated controller predefined for the rotor fault condition.

6 Experiments

We compare the trained blended control framework against a traditional switched architecture based on the same low-level controllers. We set C2 as the nominal controller as it performs slightly better under nominal operating conditions and smooth control is more desirable. After experimentation we found that C1 was more robust to rotor faults which make it an adequate fault controller in a switched architecture. A switch is triggered when the trajectory deviation is more than 2m, since the steady state error is around 1.8m this gives a small margin of error for deviations before identifying a fault.

We design two experiments on a 10m diamond path over 60s. We compare the average tracking performance of C1 and C2 on their own as well as the trained DRLBC framework and the switched control architecture. The cumulative tracking error is dependant on duration of flight and the shape of the path. We hence use the average tracking error as the performance measure reported in this article as it gives a better indication of general performance. The first experiment tests the performance under nominal operating conditions while the second compares performance under 50% rotor faults at every 5 second interval. The rotor selection is kept random and results shown are averaged over 10 independent runs to account for this.

6.1 Experiment 1: Nominal Control

Table 3 shows the average trajectory loss for the 4 tested controllers. DRLBC performs *between* the performance of C1 and C2 which is exactly as expected. The switched architecture performs exactly as C2 since no fault switch to C1 is triggered. This shows if all low-level controllers produce a similar output signal, blending to any proportions will have little effect and nominal performance can be maintained even with a constantly changing blend weight vector.

Controller	Average Tracking Error
C1	48.88cm
C2	48.86cm
DRLBC	48.87cm
Switched Control	48.86cm

Table 3: Experiment 1: Average Tracking Accuracy under nominal conditions.

6.2 Experiment 2: Rotor Faults of 50% at 5s intervals

Experimental results are shown in Table 4. We firstly investigate the performance of C1 and C2 under rotor faults. The

Controller	Average Tracking Error
C1	50.1cm
C2	60.33cm
DRLBC	50.3cm
Switched Control	55.58cm

Table 4: Experiment 2: Average Tracking error under 50% rotor loss of efficiency.

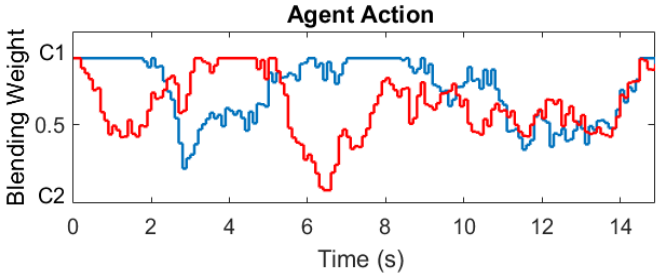

Figure 5: Agent actions under rotor faults for blending Roll (Blue) and Pitch (Red).

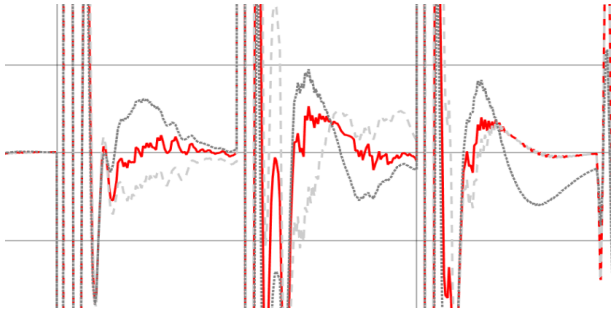

Figure 6: Grey: Low-level controller responses and Red: Blended signal through DRL. The blended signal is able to stabilize around the set-point quicker.

aggressive C1 controller performs better (50.1cm avg. error) as it reacts to faults with a higher response due to the higher PID gain parameters. This means C1 is more robust to rotor faults and validates the selection as a fault controller for the switched architecture. C2 performs with greater deviations (60.33cm avg. error) which is expected since the controller reactions to faults are smoother, hence taking longer to stabilize and increasing the overall error.

The switched architecture was able to utilize some of the benefits of the aggressive controller after the deviation had crossed a threshold and a fault was identified. With 55.58cm tracking accuracy it performs close to halfway between the low-level controller performances. Although this is an improvement to C2 the delay in the FDI unit still has major implications for the overall executed trajectory. The most important part in successfully stabilizing a rotor fault is the speed of reaction due to the highly unstable dynamics of a quadcopter.

The blended architecture performs comparable to C1. Given that the agent is bound between the low-level controllers the optimal result that could have been attained from the training is 50.1cm, C1s performance. Figure 5 shows the blend weight vector applied during a sample run of this experiment. Most of the time both controllers are used to control the system to some degree. It is interesting to point out that the agent did not converge to only use C1 as one would expect since it is the better performing of the underlying controllers. We attribute this to an improved blended controller output compared to the individual PID controllers. Figure 6 shows the over and undershooting responses of the C1 and C2 PID controllers as they stabilize on the reference signal after a rotor fault. The blended control output, shown in red, stabilizes faster and smoother than the low-level controllers can on their own which has a positive effect on the overall tracking performance.

The output of C1 and C2 are both not optimal to stabilize the system but the agent is able to use them to synthesise an improved response that is more robust to oscillations around the set-point and stabilizes quicker. This also plays a factor in the largely positive reward seen in Figure 4 as this shows the agent has successfully learned to produce a less oscillating response signal than the underlying controllers while still being able to utilize the more aggressive responses from C1 to stabilize rotor faults.

To highlight the performance difference we provide Figure 7 which shows the executed trajectories from a sample run of this experiment for DRLBC (Blue) and switched (Red) architectures. The effect of the rotor faults can be clearly seen as abrupt deviations from the X and Y reference paths (Black) at 5 second intervals.

7 Conclusion

In this article we presented a novel fault tolerant control architecture with the ability to learn unknown fault tolerance. This was achieved through the reliance on existing low-level control mechanism and an abstract application of deep learning to the high-level control task. Using blended control, this architecture exploits a new way to integrate deep learning algorithms to well known hierarchical control architectures. Low-level controllers are designed based on the type of response they provide under fault conditions rather than for specific fault conditions. The FDI unit and it's associated delays are replaced with a DDPG agent that generically identifies degrading performance on a task in real time and learns to optimize for the new conditions. This architecture was implemented on a quadcopter trajectory tracking task under rotor loss of effectiveness faults for which no identification or pre-defined optimal control mechanism exists. We validated the effectiveness of the approach through training and experimentation showing the blended controller is able to synthesize an improved control signal that handles the unknown fault as well as improve oscillations around the reference value. The size of the learning problem is greatly reduced compared to current state of the art approaches to learn direct control of state space to control signals. We showed the presented approach can track a trajectory under rotor faults more accurately than a switched architecture with the same low-level controllers.

Future work includes the training for several faults simultaneously and different application domains. More complex blending functions or different neural network designs pose a large frontier for exploration for the research community which can provide more new ways for control systems to learn to adapt to unknown faults and drive the application of autonomy to large scale systems.

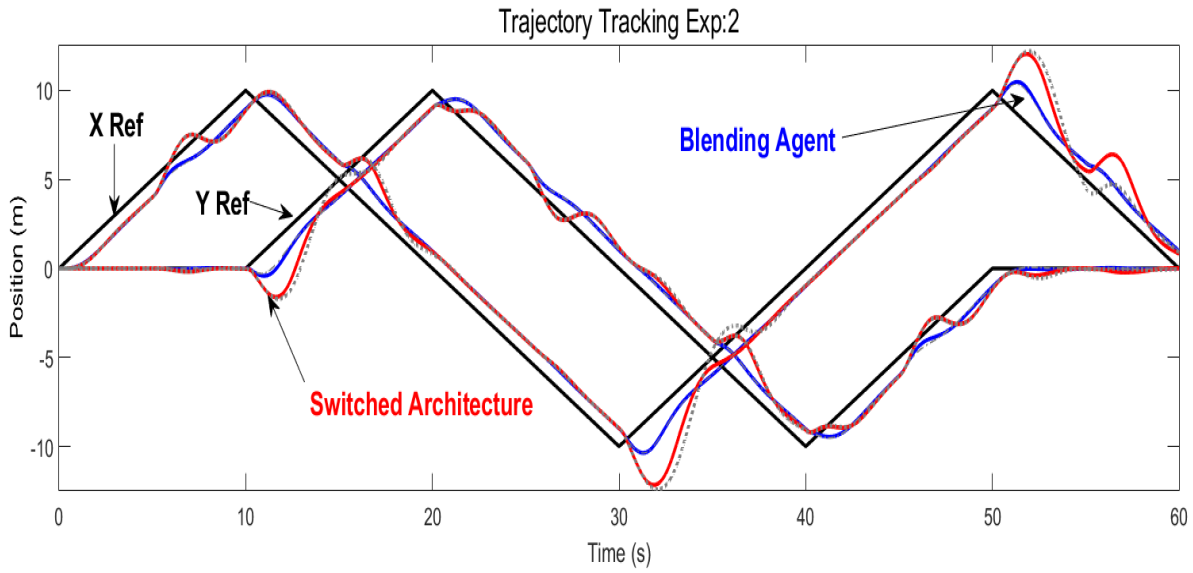

Figure 7: Trajectory under rotor failure for different control architectures. Red: Switched Architecture, Blue: Blended DRL Architecture. Black: X and Y Reference.

References

[Blanke *et al.*, 2016] Mogens Blanke, Michel Kinnaert, Jan Lunze, and Marcel Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer, third edition, 2016.

[Büyükkabasakal *et al.*, 2017] Kemal Büyükkabasakal, Barış Fidan, and Aydoğan Savran. Mixing adaptive fault tolerant control of quadrotor UAV. *Asian Journal of Control*, 19(4):1441–1454, 2017.

[Fei *et al.*, 2019] Fan Fei, Zhan Tu, Yilun Yang, Xiangyu Zhang+, Dongyan Xu+, and Xinyan Deng. Learn to Recover: Reinforcement Learning-Assisted Fault Tolerant Control for Quadrotor UAVs. 2019.

[Greatwood and Richards, 2019] Colin Greatwood and Arthur G. Richards. Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control. *Autonomous Robots*, pages 1–13, 2019.

[Hwangbo *et al.*, 2017] 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.

[Koch *et al.*, 2019] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for UAV attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):22, 2019.

[Kuipers and Ioannou, 2010] Matthew Kuipers and Petros Ioannou. Multiple model adaptive control with mixing. *IEEE Transactions on Automatic Control*, 55(8):1822–1836, 2010.

[Lillicrap *et al.*, 2015] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[Lunze, 2016] 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.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Mueller and D'Andrea, 2014] Mark W. Mueller and Raffaello D'Andrea. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 45–52. IEEE, 2014.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[Özbek *et al.*, 2016] 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.