Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Deep Reinforcement Learning and Randomized Blending for Control under Novel Disturbances

Yves Sohège * **Gregory Provan** ** **Marcos Quiñones-Grueiro** ***
Gautam Biswas ****

* *Lero Centre for Software Research, UCC, Ireland (e-mail: yves.sohege@insight-centre.org).*
** *Computer Science Department, UCC, Ireland (e-mail: g.provan@cs.ucc.ie)*
*** *Universidad Tecnológica de la Habana José Antonio Echeverría (email: marcosqg88@gmail.com)*
**** *Vanderbilt University Nashville, Tennessee, USA (email: gautam.biswas@vanderbilt.edu)*

Abstract: Enabling autonomous vehicles to maneuver in novel scenarios is a key unsolved problem. A well-known approach, Weighted Multiple Model Adaptive Control (WMMAC), uses a set of pre-tuned controllers and combines their control actions using a weight vector. Although WMMAC offers an improvement to traditional switched control in terms of smooth control oscillations, it depends on accurate fault isolation and cannot deal with unknown disturbances. A recent approach avoids state estimation by randomly assigning the controller weighting vector; however, this approach uses a uniform distribution for control-weight sampling, which is sub-optimal compared to state-estimation methods. In this article, we propose a framework that uses deep reinforcement learning (DRL) to learn weighted control distributions that optimize the performance of the randomized approach for both known and unknown disturbances. We show that RL-based randomized blending dominates pure randomized blending, a switched FDI-based architecture and pre-tuned controllers on a quadcopter trajectory optimisation task in which we penalise deviations in both position and attitude.

Keywords: Design of fault tolerant/reliable systems; Fault accommodation and Reconfiguration strategies; Methods based on neural networks and/or fuzzy logic for FDI.

1. INTRODUCTION

Enabling agents to act autonomously is a significant challenge, with many unsolved tasks. One unsolved task is enabling a system to operate safely in novel scenarios, since it is impossible to pre-compute controllers for all disturbances (e.g., faults or external disturbances like wind), let alone scenarios that cannot be predicted during design-time.

We focus on systems for which real-time control (RTC) is important. In particular, we use a quadcopter (Özbek et al., 2016) as our running example. A quadcopter is a well-studied unmanned aerial vehicle that use four propellers to maneuver. These vehicles have fewer actuators than degrees of freedom, and hence are called under-actuated. Improper design of RTC can lead to crashing. Current state of the art quadcopter control utilizes a cascading PID-architecture (Mo and Farid, 2019).

Two approaches for RTC have been developed. The *control-theoretic approach* uses model-based methods to develop controllers, and relies on state estimation to compute a controller appropriate to a given state. The *AI-based approach* uses model-free methods, and relies on machine learning to estimate control parameters for given operating conditions.

Both approaches have strengths and weaknesses. The *control-theoretic approach* can generate controls with precise guarantees, but state estimation introduces latency into applying controls, and typically these model-based solutions require *a priori* knowledge of all potential states. The *AI-based approach* requires time to learn controls for novel scenarios, so cannot respond in real-time to such situations.

We propose an approach that is based on Weighted Multiple Model Adaptive Control (WMMAC), a state-of-the-art passive adaptive control technique that blends the outputs of a set of low-level controllers. WMMAC was first introduced in (Kuipers and Ioannou, 2010) as an improvement of discrete switching-based multiple model approaches. Blending avoids the control oscillations that are problematic in other switching methods, e.g., discrete switching (Hou et al., 1996) and sliding-mode control (Edwards and Spurgeon, 1998), since all control assignments consist of a blend of multiple controllers so switching is inherently less abrupt.

Most blending algorithms rely on (a) a priori controller specifications and (b) identification mechanisms (e.g., Kalman-filters) to estimate the probability of a known disturbance and adjust the blending proportions accordingly. Few blending algorithms for unknown disturbances (where identification is impossible) exist; for real-time controllers, delays or miss-identification of faults can be catastrophic.

⋆ This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

One recent WMMAC approach for stabilizing systems subject to novel disturbances uses a randomized blending (RB) distribution over all controllers without performing state estimation (Provan and Sohège, 2019). The weakness of this approach is that it uses a uniform distribution for controller sampling, which leads to sub-optimal control in comparison to an ideal optimal controller.

This article proposes a stable, real-time controller for novel disturbances that does not require on-line fault detection and diagnosis. We extend randomized blending (RB) control with reinforcement learning (RL) (Sutton and Barto, 1998) to design an agent that learns how to adapt the blending distributions depending on the system state and environmental conditions. We use deep reinforcement learning to teach an agent how to parameterize the RB distribution depending on the scenarios encountered. This approach improves on traditional RL methods for real-time control (e.g., (Hwangbo et al., 2017; Koch et al., 2019; Shi et al., 2018)) as the RB guarantees stability for the exploration phase of novel scenarios, whereas no stability guarantees exist for any existing RL-based controller in novel scenarios. Further, the randomization inherent in control execution provides an excellent basis for RL exploration, as it provides good coverage of the control space under the novel situation.

Our contributions are as follows.

- We present a novel real-time control system based on WMMAC with randomized blending and deep reinforcement learning. The agent is trained to learn the optimal randomized blending distribution offline for known faults given a static controller set. We show the trained framework is able to maintain control for unknown disturbances by shifting the randomized blending distributions accordingly in real time.
- We demonstrate our approach using a trajectory-following task for a quadcopter, by comparing its path- and attitude-deviation performance against that of two hand-tuned controllers, a (non-learning) randomized approach, and a traditional switching controller. We show that a neural-network based high-level controller trained for two fault conditions, abrupt rotor faults and attitude sensor noise, outperforms the other architectures under unknown disturbance conditions including wind-gusts, position noise and a combination of all known and unknown faults.
- The proposed fault-tolerant control scheme does not require an online fault-detection step.

The paper layout is as follows: Section 2 outlines various relevant control architectures and deep reinforcement learning. We introduce the generic Deep Reinforcement Learning Randomized Blended Control (DRLRBC) architecture in section 3 and the general quadcopter model in section 4. Section 5 gives the implementation and training of DRLRBC on a quadcopter followed by experimental evaluation and conclusion in section 6 and 7 respectively.

2. BACKGROUND & RELATED WORK

This section discusses relevant background and control architectures explored in this article. We introduce manually-tuned controllers and then some hierarchical control frameworks. We are interested in comparing different high-level frameworks that use the same low-level controllers.

(1) Manual-tuning is widely done but balancing the system behaviour for different operating scenarios during the tuning process is extremely complex. Sub-optimal system controllers that exhibit unwanted behaviour during specific operating conditions are common. In this article two manually tuned controllers are designed to exhibit a specific behaviour, referred to as *C1* and *C2*. The low-level controller set for any high-level control architecture compared in this article will always consist of *C1* and *C2*.

(2) Discrete Switching is a hierarchical control architecture based on a set of low-level system controllers tuned for specific operating conditions. A large amount of work has been done on hierarchical control architectures (Blanke et al., 2016; Lunze, 2016). A high-level controller identifies changes in operating conditions such as faults and discretely switches control between the system controllers using a switching function. A drawback of this architecture for novel disturbance scenarios is that no pre-defined controller exists as controller design requires *a priori* knowledge of the operating conditions. By definition the performance of the pre-defined controllers is inherently unknown for novel scenarios, but one controller must maintain system control which leads to unknown performance.

(3) Weighted Multiple-Model Adaptive Control (WMMAC) uses a high-level controller to blend the inputs of a set of low-level system controllers, each of which is defined for a specific operating condition. Formally:

Definition 1. (**WMMAC**). Given a collection of controllers $\Omega = \{\omega_1, \cdots, \omega_m\}$ and a control distribution vector $\boldsymbol{\varphi} = \{\varphi_1, \cdots, \varphi_m\}$, the blended control signal is given by a weighted combination $\boldsymbol{\omega} = \sum_i \varphi_i \omega_i$ such that: (1) $\forall_i, \omega_i \in \Omega, 0 \leq \varphi_i \leq 1$, and (2) $\sum_i \varphi_i = 1$

Constraints (1) and (2) ensure that the blended control signal is bound *between* the low-level controller outputs. In (Zhang, 2015) the stability of WMMAC is proven for a discrete time stochastic plant and (Kersting and Buss, 2018) discusses a systematic distribution of controllers for MMAC. A major drawback of the WMMAC approach is that computing blend weights relies on FDI techniques, which are dependent on *a priori* knowledge about the disturbances. Further, detection and isolation may be inaccurate and takes inherently takes time, which can be problematic for RTC.

(4) Randomized Blended Control (RBC) is a randomized version of WMMAC. RBC avoids the estimation phase of WMMAC, and uses randomization to estimate a blend distribution by sampling uniformly over the space of all low-level controller weights. (Provan and Sohège, 2019) show that, given unknown disturbances, RBC can stabilize a system, and that its performance for known scenarios converges to the performance of the optimal low-level controller.

Control of multiple-model systems using mixing has been shown to be stable for situations in which the (fixed) unknown parameter set of the plant is assumed to lie in the convex hull of the control parameters of the multiple models (Han and Narendra, 2011). This has recently been extended to the case of systems with an unknown varying parameter set (Narendra and Esfandiari, 2019). Using these notions, (Provan and Sohège, 2019) also show stability of the randomized approach, bounded by the convex hull of the available controllers.

(5) RL-based control. We now introduce our approach, which extends the *Non-Learning (NL)* RBC with an architecture based

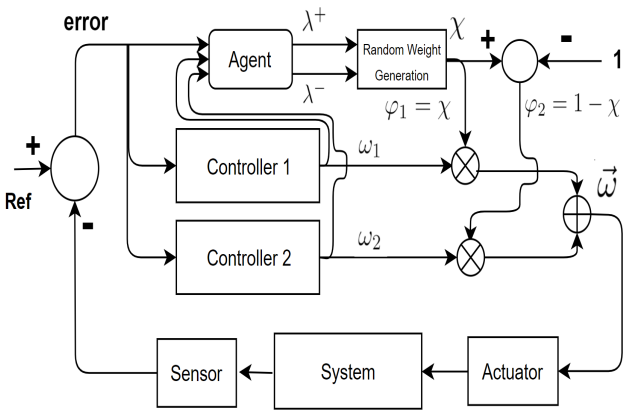

Fig. 1. The DRLRBC architecture (with two controllers).

on Deep Reinforcement-Learning (DRL), which we call Deep Reinforcement-Learning RBC (DRLRBC). DRL (Sutton and Barto, 2018) has successfully been able to learn accurate controllers for complex systems such as quadcopters and cars (Sallab et al., 2017; Hwangbo et al., 2017). Further, although optimal control and RL have been developed in different communties, they are both capable of solving the same optimal control task (Powell, 2019). One weakness of RL-based control approaches is the training-time required to learn control of the system for each of several operating conditions, and the complexity of defining an accurate reward function. Currently, no approach exists that can provide stability guarantees under novel disturbances. This article extends the RBC approach with deep RL in order to learn a controller weight distribution that is tailored to the observed environmental/fault conditions. To simplify the training process, we use for our training data a restriction of the entire state space of the quadcopter, namely attitude loss and low-level controller outputs.

3. DEEP REINFORCEMENT LEARNING RANDOMIZED BLENDED CONTROL

We will give a generic overview of the presented architecture, referred to as Deep Reinforcement Learning Randomized Blended Control (**DRLRBC**). An architecture diagram showing a two controller example can be seen in Figure 1. We will discuss the architecture in four parts: (1) Low-Level Controllers, (2) Performance Estimate, (3) Blending Function and (4) High-Level Controller.

3.1 Low-Level Controllers

In this article, we will restrict the low-level controllers to PID controllers, which are an industry-standard way of controlling automatic systems (Özbek et al., 2016). Fault tolerant control architectures tune low-level controllers for different operating conditions to achieve fault tolerance under those conditions (Lunze, 2016). Although MPC and LQR controllers have been applied to quadcopters, they are typically designed around a fixed operating point and require extensive modelling effort (Mo and Farid, 2019). The investigation into these controllers is planned future work and beyond the scope of this article. The set of low-level controllers outputs is denoted $\Omega = \{\omega_1, \cdots, \omega_N\}$. Figure 1 shows an example architecture with two controllers.

3.2 Performance Estimate

We define $\delta = [\mathcal{P}, \mathcal{E}]$, where $\mathcal{P}$ indicates overall system performance and $\mathcal{E}$ the current control effectiveness. These measures are system- and task-dependent and give the agent a real-time measure of how well the task is being achieved.

3.3 Blending Function

In developing DRLRBC we must make additional modifications to RBC. We adapt the RBC input parameters to allow an external input Λ, referred to as the *Randomization Bounds Vector (RBV)*, to control the range from which φ_i is sampled. Λ is a set of tuples $[\lambda_i^-, \lambda_i^+]$, one tuple for each low-level controller $\omega_i \in \Omega$, indicating the range from which the randomized blend weight φ_i is sampled. More precisely :

Definition 2. (Bounded Randomized Blending). Given a collection of controller outputs $\Omega = \{\omega_1, \cdots, \omega_m\}$ and Randomization Bounds Vector $\Lambda = \{[\lambda_1^-, \lambda_1^+], \cdots, [\lambda_m^-, \lambda_m^+]\}$, the control distribution vector $\varphi = \{\varphi_1, \cdots, \varphi_m\}$ is randomly sampled such that for the weighted combination $\boldsymbol{\omega} = \sum_i \varphi_i \omega_i$ the following constraints hold: (1) $\forall_i, \omega_i \in \Omega, \lambda_i^- \leq \varphi_i \leq \lambda_i^+$, (2) $\sum_i \varphi_i = 1$,and (3) $0 \leq \lambda_i \leq 1$

By introducing a bound on the range from which random blend weights are sampled from, the space of combinations of controllers can be explored while still relying on the stability guarantees provided by RBC under novel disturbances.

3.4 High-Level Controller

In the presented architecture the high-level controller is implemented using a deep neural network as they are known to be excellent function approximators. Neural networks have continuous observation and action spaces which remove the inherent FDI delay experienced by traditional switched systems.

The observation vector for DRLRBC can generically be defined as: [δ , Ω]. The agent's action output is the Randomization Bounds Vector, Λ. So the goal of the agent is to learn the mapping: $[\delta, \Omega] \rightarrow \Lambda$ that optimizes the performance of the system on a given task. A more detailed example of observation vector and action space will be given in section 5.

3.5 Architecture comparison

We contrast the presented DRLRBC architecture to current state of the art approaches. The deep learning algorithm is applied to an abstracted high-level task while relying on existing low-level system 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. DRLRBC uses a measure of task performance to direct how influential each controller is in the applied signal, which is a much smaller problem to learn.

WMMAC enables FTC with respect to a pre-defined set of (partial) faults, but at the expense of (a) tuning a controller for

each fault and (b) using FDI to isolate the fault magnitudes prior to controller allocation. In contrast, our approach does not require *a priori* knowledge of any faults or FDI for fault isolation; our randomized controller assignment (as tuned to actual faults via RL) handles the FTC.

The framework we propose aims to estimate the optimal blended control signal given a static pre-defined controller set under situations it was not trained for. Since we are calculating a convex combination of controller outputs, the overall range of the blended signal is limited to between the low-level controller outputs.

4. QUADCOPTERS

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 and wind disturbances 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 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. The wind is generated as a drag force that acts on the body frame ε.

Quadcopters have been shown to be able to maintain flight even after the complete loss of one or more rotors under specific conditions (Mueller and D'Andrea, 2014). The standard way to achieve trajectory tracking is by employing a cascading PID controller structure, which is the approach we consider in this article. Position controllers generate the required attitude reference to execute the trajectory. Roll and Pitch attitude controllers generate the required motor commands to attain the attitude. Blended control is only applied on the attitude controllers, not on the position controllers.

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 significant training data and complex fine-tuning of the reward functions to achieve 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.

[1] The MATLAB simulation codebase necessary to run the experiments in this article is available under github.com/YvesSohege/IFAC20-Simulation.

The task of the Quadcopter in this work is focused on **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 Total Trajectory Loss:

Definition 3. (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 $k = 0, \cdots, T$.

Given the Total Trajectory Loss over the T time-steps, the current trajectory loss at time t can be defined as $\mathcal{L}_t$. Similarly, we introduce a second performance metric to measure the attitude tracking error, $\mathcal{A}_{0:T}$, as a loss function between reference attitude $\boldsymbol{\Delta}^R(k)$ and actual attitude $\tilde{\boldsymbol{\Delta}}(k)$ which is omitted due to space constraints. For simplicity we will focus on 2D (x, y) trajectory tracking but the approach extends to 3D trajectories with minimal changes. We do not apply blending on the rotational ψ controller of the quadcopter as this does not effect a 2D trajectory.

5. QUADCOPTER IMPLEMENTATION AND TRAINING

In this section we will describe the implementation of the DRLRBC architecture on a Quadcopter MATLAB simulation as well as the training details of the agent. [2]

5.1 Low-Level Controllers

In this article, we will use two controllers for the roll and pitch attitude of the quadcopter. Each axis will have a C1 and a C2 controller with different tuning which can be seen in Table 1. Since a quadcopter is symmetric we can use the same tuning for both axis. By changing the PID gain parameters, we change how the quadcopter responds to different situations. The controllers were hand-tuned to nominal operating conditions under which they perform identically in terms of trajectory loss. This is highlighted experimentally in Table 3 Exp1, in Section 6. However C2 has a higher proportional gain which allows it to respond more aggressively to abrupt disturbances than C1. We also purposely add slight oscillations around the reference when tuning C2 by slightly lowering the derivative gain.

	$\mathcal{P}$	$\mathcal{I}$	$\mathcal{D}$
C1 (Smooth)	2	1.2	1.3
C2 (Aggressive)	4	1.5	1.2

Table 1. PID parameters for low-level controller tuning used.

5.2 Performance Estimation

We use the current x and y trajectory error, denoted δ_x and δ_y respectively, as a measure of how well the system is performing overall on its task. Blended control is being applied on two axis of control and each needs a measure of effectiveness. For this we select the current roll and pitch attitude error, denoted δ_ϕ and δ_θ respectively. We can hence define the full performance estimation output for the Quadcopter as $\delta = [\delta_x\ \ \delta_y\ \ \delta_\phi\ \ \delta_\theta]$. These metrics allow the agent to judge how good control is for the current scenario and adapt it to maximize the performance without explicit knowledge of the operating conditions.

[2] Real-world flights are beyond the scope of this article.

5.3 Blending Function

Since this article focuses on blended control of a controller **pair** a simplification can be made to Equation 2. By relying on Constraint (2) of Equation 2 which forces the sum of both controller weighting to be 1, after randomly generating the first weight, the second can simply be obtained by subtraction from 1. This simplification is indicated using χ in Figure 1 and reduces the size of Λ by half.

5.4 High-Level Controller

An actor-critic DDPG network structure is used to generate the randomized bounds vector Λ. Both actor and critic take in the same observation vector which we define in full detail as $[\delta_x \ \delta_y \ \delta_\phi \ \delta_\theta \ \omega_1^\phi \ \omega_2^\phi \ \omega_1^\theta \ \omega_2^\theta]$, where ω_1^ϕ indicates C1 for the roll axis, ω_2^θ is C2 for pitch and so forth.

Through the additional simplification introduced to the Blending Function the agents action output can simply be defined as $[\lambda_\phi^- \ \lambda_\phi^+ \ \lambda_\theta^- \ \lambda_\theta^+]$, indicating the lower and upper bounds of the randomized blending ranges for ϕ and θ controllers respectively. We give a brief overview of the neural network architecture. 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 4 is used to naturally enforce the blended control constraints, bounding the agent action space 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.

Training Conditions We use random rotor loss of effectiveness (LOE) and noise on the attitude sensors of the quadcopter as training conditions. All rotor faults in this article are of magnitude 10%. To vary the training conditions we set the time a rotor fault can occur randomly, with every time-step having a probability of spontaneously losing angular rotor velocity of 10% for one time-step. This has an impact on both attitude and trajectory. The noise is modelled as random white noise applied to roll and pitch state, ϕ and θ respectively, and is defined simply by its magnitude.

Reward function We define the reward function in terms of Total Trajectory and Attitude Loss. The agents goal is to maximize its reward function so we define the reward obtained by the agent as:

$$\mathcal{R}(t) = -(|\delta_x(t)| + |\delta_y(t)|) - (|\delta_\phi(t)| + |\delta_\theta(t)|)$$

which represents the sum of trajectory and attitude loss at time t. We negate this to allow the agent to maximize the reward. This reward function allows the agent to learn how to improve the trajectory and attitude tracking performance regardless of operating conditions. For training we use a straight line path of 30 meters executed over 30 seconds. The quadcopter was trained over 3000 episodes and average reward is calculated over 50 episodes. The agent converged after around 1500 episodes. We define other relevant training parameters used in Table 2 and refer the reader to the github repository provided in section 4 for further details.

Parameter	Value
Discount factor	0.99
Initial learning rate of the critic	0.01
Initial learning rate of the actor	0.025
Batch size	32
Replay buffer size	10000
Training steps of an episode	300
Number of episodes	3000

Table 2. DDPG Training parameters used

6. EXPERIMENTS

We empirically compare the simulated performance of the presented control systems on a number of scenarios, including known and novel disturbances. We do not consider faults that would cause catastrophic failure. Since the agent was trained on abrupt rotor loss of effectiveness and attitude noise, we classify these as the known disturbances. In addition wind gusts and position noise are tested and we classify these as novel disturbances. We compare the two baseline controllers, C1 and C2, with three high-level control architectures which utilize C1 and C2 to improve control, namely DRLRBC, Non-Learning RBC and Switching. We define the switching condition for the switched architecture as a trajectory deviation of 10% above nominal tracking error. C1 is selected as the nominal controller and C2 as the fault controller. We use a diamond shape path with diagonal length of 10 meter starting and ending in the centre over 60 seconds with a sample time of 0.1s. Since all disturbances are modelled with randomness we present average results taken over ten runs on each experiment.

The full list of experimental conditions can be found in Table 4. We test nominal conditions, three known disturbances (rotor loss of effectiveness, attitude noise and both) and three unknown disturbances (position noise, wind gusts and all faults). Magnitude of disturbances (rotor and wind) is given followed by disturbance trigger probability at any time step. E.g: *10% error (30%)* indicates every time-step has a 30% chance of triggering a 10% fault.

6.1 Experimental Results

We evaluate the performance on each experiment using the average Total Trajectory Loss and Total Attitude Loss of each control system over 10 independent runs. We present the complete set of results in Table 3. On average across all experiments the presented architecture is able to outperform all other control systems in terms of attitude and trajectory tracking accuracy.

Known Disturbances The training conditions included rotor faults and attitude noise. DRLRBC is able to significantly outperform all other control systems for trajectory accuracy for all three experiments showing the agent successfully learned to improve its performance under known disturbance conditions. Under heavy attitude noise C1 is able to track the attitude slightly more accurately than the DRLRBC architecture but at the expense of trajectory loss.

Unknown Disturbances Positional noise and wind gusts are used to test the control systems under unknown scenarios. DRLRBC performs second best across all 3 experimental conditions in terms of trajectory accuracy showing a comparable performance. In terms of attitude tracking DRLRBC is able to perform best under wind gusts and perform comparable to C1 and NL-RBC under the other two scenarios.

	Total Attitude Loss (rad)					Total Trajectory Loss (m)				
Exp #	DRLRBC	C1	C2	NL-RBC	Switched	DRLRBC	C1	C2	NL-RBC	Switched
Exp1	**0.68**	1.23	0.78	0.88	1.23	48.98	**48.79**	**48.79**	**48.79**	**48.79**
Exp2	**30.43**	42.08	43.71	35.1	41.08	**50.88**	98.51	59.31	55.71	56.20
Exp3	**19.98**	43.74	44.62	25.59	30.96	**50.60**	127.42	60.77	57.98	65.38
Exp4	33.16	**30.62**	42.07	34.39	34.90	**55.17**	57.87	56.36	56.21	63.98
Exp5	**10.98**	17.5	14.30	12.62	12.35	52.07	64.70	**51.67**	53.42	53.72
Exp6	16.73	**15.03**	23.01	16.29	20.89	57.91	57.97	**57.15**	58.39	57.67
Exp7	29.14	36.07	42.14	**27.43**	28.17	55.85	79.28	60.49	**55.66**	56.89
Average	**20.16**	26.62	30.09	21.76	24.23	**53.07**	76.36	56.36	55.16	58.66

Table 3. Experiment list showing Total Attitude Loss (in radians) and Total Trajectory Loss (in meters) for the compared control systems. Best performing in bold text.

Exp #	Att. Noise	Pos. Noise*	Rotor LOE	Wind Gusts*
Exp1	-	-	-	-
Exp2	0.02	-	10% error (10%)	-
Exp3	-	-	10% error (30%)	-
Exp4	0.05	-	-	-
Exp5	-	-	-	0-10m/s (30%)
Exp6	-	0.05	-	-
Exp7	0.02	0.02	10% error (10%)	0-10m/s (10%)

Table 4. Experimental Disturbance Details.

7. CONCLUSION

In this article, we presented a novel hierarchical control architecture based on weighted multiple model adaptive control, deep reinforcement learning and randomized blending. The presented architecture is tested on a quadcopter trajectory tracking simulation, and trained under rotor loss of effectiveness and attitude noise. We compare the presented architecture to a non-learning randomized approach, a standard switched architecture, and the underlying controllers working individually. We showed that, averaged across all experiments, the presented architecture outperforms all other baselines in terms of both trajectory tracking and attitude tracking under *known* and *unknown* disturbances. This extends the field of fault tolerant control by providing a novel way to apply deep reinforcement learning to high-level control tasks. In future work, we plan to explore additional learning mechanisms, such as unsupervised learning and a larger set of low level controllers.

REFERENCES

Blanke, M., Kinnaert, M., Lunze, J., and Staroswiecki, M. (2016). *Diagnosis and Fault-Tolerant Control*. Springer.

Edwards, C. and Spurgeon, S. (1998). *Sliding mode control: theory and applications*. CRC Press.

Fei, F., Tu, Z., Yang, Y., Zhang, X., Xu, D., and Deng, X. (2019). Learn to Recover: Reinforcement Learning-Assisted Fault Tolerant Control for Quadrotor UAVs.

Greatwood, C. and Richards, A.G. (2019). Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control. *Autonomous Robots*, 1–13.

Han, Z. and Narendra, K.S. (2011). New concepts in adaptive control using multiple models. *IEEE Transactions on Automatic Control*, 57(1), 78–89.

Hou, L., Michel, A.N., and Ye, H. (1996). Stability analysis of switched systems. In *Decision and Control, 1996., Proceedings of the 35th IEEE Conference on*, volume 2, 1208–1212.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4), 2096–2103.

Kersting, S. and Buss, M. (2018). How to systematically distribute candidate models and robust controllers in multiple-model adaptive control: A coverage control approach. 63(4), 1075–1089.

Koch, W., Mancuso, R., West, R., and Bestavros, A. (2019). Reinforcement learning for UAV attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2), 22.

Kuipers, M. and Ioannou, P. (2010). Multiple model adaptive control with mixing. *IEEE Transactions on Automatic Control*, 55(8), 1822–1836.

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

Mo, H. and Farid, G. (2019). Nonlinear and adaptive intelligent control techniques for quadrotor UAV a survey. 21(2), 989–1008.

Mueller, M.W. and D'Andrea, R. (2014). 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)*, 45–52. IEEE.

Narendra, K.S. and Esfandiari, K. (2019). Adaptive identification and control of linear periodic systems using second-level adaptation. *International Journal of Adaptive Control and Signal Processing*, 33(6), 956–971.

Özbek, N.S., Önkol, M., and Efe, M.Ö. (2016). Feedback control strategies for quadrotor-type aerial robots: a survey. *Transactions of the Institute of Measurement and Control*, 38(5), 529–554.

Powell, W.B. (2019). From reinforcement learning to optimal control: A unified framework for sequential decisions.

Provan, G. and Sohège, Y. (2019). Fault-tolerant control for unseen faults using randomized methods. In *IEEE SysToL*.

Sallab, A.E., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19), 70–76.

Shi, Q., Lam, H.K., Xiao, B., and Tsai, S.H. (2018). Adaptive PID controller based on Q-learning algorithm. *CAAI Transactions on Intelligence Technology*, 3(4), 235–244.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: an introduction*. The MIT Press.

Zhang, W. (2015). Further results on stable weighted multiple model adaptive control: Discrete-time stochastic plant. 29(12), 1497–1514.