Fault Tolerant Control combining Reinforcement Learning and Model-based Control *

Luke Bhan[1], Marcos Quinones-Grueiro[1] and Gautam Biswas[1]

***Abstract*— Fault tolerant control (FTC) focuses on developing algorithms to accommodate the impact of system faults while allowing the system to continuously operate in a degraded manner. Additionally, data-driven methods like reinforcement learning (RL) have shown excellent performance for complex continuous control tasks. However, faults affect the system dynamics which disrupt optimal policy guarantees as the Markov Property cannot be satisfied. In this work, we propose a scheme based on a combination of parameter estimation, RL, and model-based control to handle faults in a continuous control environment. We empirically demonstrate our approach on a complex octocopter trajectory-tracking task subject to single and multi motor faults. We show improved performance compared with nominal hierarchical PID control for large magnitude faults. Lastly, we demonstrate our approach's robustness against noisy parameter estimation.**

I. INTRODUCTION

Faults are defined as deviations of system properties or parameters that prevent the system from efficiently operating. Failures, on the other hand, represent a more drastic condition that completely prevents a system from functioning. Fault Tolerant Control (FTC) methods address the problem of improving system performance when operating in a degraded manner because of a fault(s). [1].

FTC approaches are broadly classified into active and passive solutions [2]. Active methods rely on a Fault Detection and Isolation (FDI) module that informs the controller about the characteristics of a fault and updates the control law accordingly. Passive methods involve predefined faults resulting in less computational complexity as FDI modules; however, they are constrained by the designers ability to predict fault instances. In addition, FTC techniques can be classified into model-based and data-driven methods depending on controller architecture [3]. Model-based controllers are designed based on physics representations of a system with parameters estimated from measurement data. In contrast, data-driven controllers learn directly from system data. As such, model-based controllers require assumptions about the fundamental behavior of the system while data-driven controllers only require a data set that accurately depicts the scenario and its operating conditions.

*This work was partially supported by NASA SWS Award number 80NSSC21M0087 (subaward 21-S06 to Vanderbilt University).

[1]Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA luke.bhan@vanderbilt.edu, marcos.quinones.grueiro@vanderbilt.edu, gautam.biswas@vanderbilt.edu

The performance of model-based FTC methods depends on having accurate and comprehensive first-principle models. Complex models require enhanced control methods compromising the robustness of the controller. Data-driven FTC methods allow developing complex control strategies but they are sample inefficient requiring numerous experiments to achieve a satisfactory performance. As such, deep reinforcement learning (DRL) approaches attempt to minimize this sample inefficiency and have become very popular for solving continuous control tasks in the last few years [4].

DRL applications to solve the FTC problem have been proposed in [5], [6]. For example, Wang et al.[6] presented a Deterministic Policy Gradient algorithm with an integral compensator for robust quadrotor control. However, as it will be shown in this paper, DRL methods for end-to-end control of systems lose convergence guarantees for FTC problems. As an alternative, recent works have combined model-based and DRL for FTC and robust control [7], [8], [9]. However, none of these works show capability to handle complex dynamic models nor multiple faults. Additionally, several model-based tuning methods have been proposed [10]; however, these methods involve underlying assumptions about the mechanics of the system and require substantial time to calculate which is inapplicable for rapidly changing dynamical systems. Meanwhile, our DRL agent learns from data which can be trained offline and then used online for fast acting adaptability. Additionally, since our DRL agent learns offline, we can leverage the experience from a wide range of faults compared to sole model-based tuning methods which must recompute the parameters for each fault after it occurs in online conditions. Therefore, the main contribution of this paper is a novel FTC architecture combining parameter estimation techniques with model-based control and DRL to solve the FTC problem. We consider an octocopter trajectory-tracking task subject to single and multiple motor faults with varying magnitude as a case study. Moreover, we generate the faults based on modifying the physical parameters of the system instead of manipulating the signals artificially which has been explored in the previous referenced literature. Finally, we make the model fully available so other researchers can work on the FTC problem for complex systems like octocopters.

The structure of the paper is the following. In Section II, the preliminaries of the different methods we use are explained. In Section III, we present our approach to FTC. The case study and fault scenarios considered are detailed

in Section IV. The experiments and results are presented in Section V, and finally, conclusions and directions for future works are given in Section VI.

II. PRELIMINARIES

A. Reinforcement Learning

Reinforcement learning (RL) aims to solve an optimal control problem through neural network-based methods. The control law is refined through the continuous interactions of a learning agent with an environment [11]. The control problem is formalized through the following definition,

Definition 1 (Markov Decision Process): A Markov decision process is defined by a four tuple: $M = fS; A; T; Rg;$ where S represents the set of possible states in the environment. The transition function $T : S \quad A \quad S \; ! \; [0; 1]$ defines the probability of reaching state s' at $t + 1$ given that action $a \; 2 \; A$ was chosen in state $s \; 2 \; S$ at *decision epoch* t, $T = p(s'js; a) = P r fs_{t+1} = s'js_t = s; a_t = ag$. The reward function $R : S \quad A \; ! \; <$ estimates the immediate reward $R \quad r(s; a)$ obtained from choosing action a in state s. The objective of the agent is to find an optimal policy * that maximizes the following criteria $8s \; 2 \; S$:

$$V^{*}(s) = \max_{\in} E\left[\sum_{t=0} {}^{t} R(s_t; a_t) js_0 = s; a_t = (s) \right]; \tag{1}$$

where $V \; : S \; ! \; R$ is called value function and it is defined as

$$V \; (s) = E\left[\sum_{t=0} {}^{t} R(s_t; a_t) js_0 = s \right]; \; 8s \; 2 \; S; \tag{2}$$

where $0 <$ 1 is called the discount factor, and it determines the weight assigned to future rewards. The agent's objective is to find the policy that maximizes the expected sum of reward. Obtaining a policy with optimality guarantees requires the following two conditions to be satisfied

1) $jR \quad r(s; a)j \quad C < 1; 8a \; 2 \; A; s \; 2 \; S$
2) T and R do not change over time.

Systems subjects to faults undergo changes that cause their dynamic model, represented by the transition function T, to change over time [4]. Therefore, learning direct control with DRL for fault tolerance is not theoretically feasible. Given this, we propose a fault adaptive control scheme that avoids using DRL for direct control, but combines model based and DRL in the next section.

B. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm based on the Natural Policy Gradient method [12]. PPO learns a policy (s) using a neural network whose input is the state vector and the output is the mean and standard deviation of the best possible action in that state. A second neural network, called the value network, keeps track of the values associated with the states under this policy. This is subsequently used to estimate the advantage of a certain action compared to alternatives in that state. This network is trained using the Temporal Difference (TD) error [11]. PPO has demonstrated outstanding performance compared to other gradient-based policy learning algorithms for a number of complex stochastic environment benchmarks, such as those provided in the *Mujoco* platform [13]. Moreover, PPO guarantees monotone improvement of the policy over multiple learning iterations and as such, we select PPO as our DRL algorithm.

III. PROPOSED APPROACH

We propose the combination of model-based control schemes with DRL as a solution to the FTC problem presented in Figure 1. Model-based control methods like Proportional Integral Derivative (PID) Controllers remain dominant in real world industry applications thanks to their simple structure, ease of implementation, and wide variety of tuning methods [14]. Nonetheless, traditional tuning methods for PID control do not account simultaneously for multiple input-output systems and multiple PID controllers. Carlucho et al. [15] proposed to use DRL for PID parameter tuning to tackle previously mentioned problems in robotic tasks. However, adaptation is required for control systems which undergo faults. We propose to extend the scheme proposed in [15] to accommodate faults assuming they are not catastrophic- the system can continue to operate in a degraded manner and performance can be recovered to some extent by updating the PID parameters.

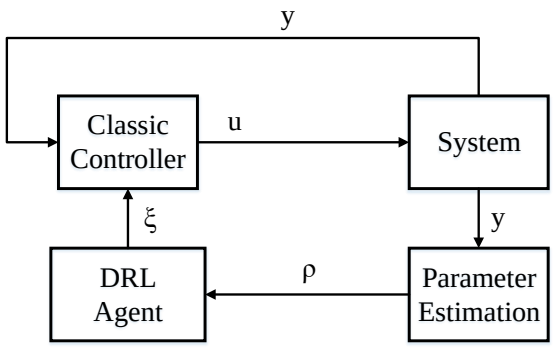

Fig. 1. Fault Adaptive Control framework

The core of the proposed approach relies on the combination of parameter estimation techniques with DRL. We propose to update the PID controller when the value of the parameter(s) associated with faults affect the control performance. The measurements obtained from the system $y \; 2 \; <^m$ are used to estimate fault-related parameters 2 $<^n$ through widely studied estimation techniques like the Unscented Kalman Filter and the Particle Filter [16]. The estimated parameters are then used as inputs to the DRL agent ($s =$) and the action of the agent consists in a new set of parameters for the controller(s) ($a =$). We demonstrate

the feasibility of the proposed approach in a complex control task presented in the next section.

IV. CASE STUDY

We consider an octocopter dynamics model based on Newton-Euler equations of motion for a rigid body [17].

The octocopter's cascade control scheme is shown in Figure 2. This control approach allows for stabilization of the position and orientation of the octocopter with respect to a trajectory. A set of three PID controllers adjust the vehicle attitude, and a different set of three PID controllers adjust the position variables, together forming nested feedback loops. The reference trajectory is defined in terms of position and

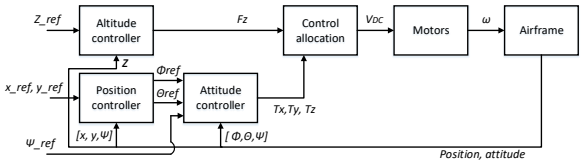

Fig. 2. Cascade Control scheme for the octocopter

yaw angle $[x_t; y_t; z_t; z_t;\ _t]$. The Altitude PID controller generates the required force in the z direction. The position PD controllers estimate, based on the current position of the vehicle and yaw angle, the reference for the pitch ( ) and roll ( ) angles. The attitude PD controllers generate the required torque in each direction. The control allocation block transforms the torques and forces into a reference voltage for each motor of the octocopter. Finally, each motor generates angular velocity according to brushless dc motor dynamics and we cap the input voltage to 11.1V to represent a realistic motor scenario. [18] More details of the octocopter modeling and control allocation can be found in [19]. Additionally, we publicly publish our Open AI Gym Environment for exploration of the Octorotor dynamics case study and to encourage future research in this sector of fault tolerant control - Octorotor Gym Github.

TABLE I
OCTOCOPTER PARAMETERS

Parameter	Value
Mass	2
Inertia Coefficient X	0.0429 kgm^2
Inertia Coefficient Y	0.0429 kgm^2
Inertia Coefficient Z	0.0748 kgm^2
Length	1m
Rotor Thrust Constant	$8.54858 * 10^{-6}\ Ns^2/rad^2$
Drag Constant	$1.3678 * 10^{-7}\ Nms^2/rad^2$
Nominal Motor Resistance	0.2371 Ω
Electrical Motor constant	0.0107 Vs/rad
Mechanical Motor Constant	0.0107 Nm/A

A. Fault scenarios

Typically, the degradation of the components of the octocopter increases monotonically from mission to mission. Motors are susceptible to mechanical degradation in the form of bearing wear, and electrical degradation in the form of contact corrosion and insulation deterioration [20]. Instead of generating faults through the manipulation of control signals as has been done in previous works, we take a more realistic simulation approach and generate the faults by modifying the value of the motor parameters. We consider the following simplified model for each of the eight motors

$$\dot{!} = \frac{1}{J_m}(K_e i_c \quad T_{load} \quad D_f ! \quad T_f); \tag{3}$$

$$i_c = \frac{1}{R_e q}(v_{DC} \quad K_e ! _i); \tag{4}$$

where $R_e q = \frac{2}{3} \sum_{j=1}^{3} R_j$ is the equivalent electric resistance of the coils, K_e is the back electromotive force constant, $!$ is the angular velocity, T_f is the static friction torque, D_f is the viscous damping coefficient, J_m is the inertia along the z axis, v_{DC} is the input voltage control signal, i_c is the current demanded, and T_{load} represents the torque load generated by the propellers. An increase in winding resistance ($R_e q$) results in the loss of effectiveness of the motor. Therefore, through the modification of this parameter we generate faulty behaviors of the octocopter in the trajectory-tracking task. In this work, we considered single motor faults ranging between 3 and 8 times the nominal value of the resistance as well as dual motor faults ranging between 2 and 4 times the nominal value of the resistance. Additionally, we refer to the motors as labeled in figure 3 for defining the motor's position in our experiments.

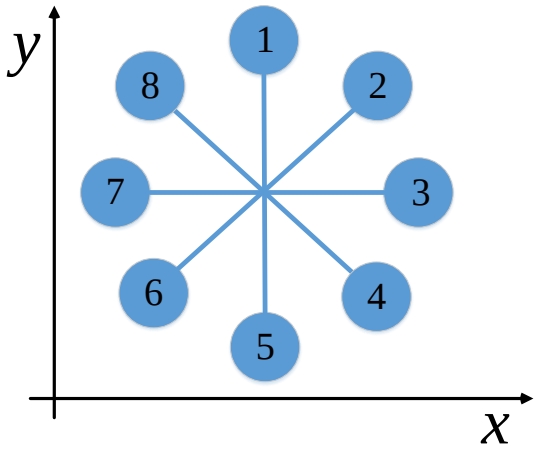

Fig. 3. Octocopter motor fault configuration

V. EXPERIMENTS AND RESULTS

A. Experimental design

In this work, we considered training the DRL agent to learn how to adapt the parameters of PD position controllers. This implies a four-dimensional action space $a = f K_p^x; K_d^x; K_p^y; K_d^y g$. Additionally, the state space consists of the position vector, velocity vector, Euler angles, and

their angular velocities $s = fx; y; z; \dot{x}; \dot{y}; \dot{z};\ ;\ ;\ ; -; -; -g$. The only information received by the agent is the motor resistance estimated and the reward function is defined by $R = (10 \quad error)$=10 where $error$ is the Euclidean distance calculated between the position of the octocopter and the reference trajectory. We defined 10 meters as the maximum deviation allowed from the reference trajectory and we re-scale the reward between 0-1 as suggested for continuous control tasks[21]. Other error functions were considered such as $R_t = max(0; 1 \quad jj\mathbf{x} \quad \mathbf{x}_{ref}jj) \quad C\ jj\ jj \quad C_!jj!jj$ where $\mathbf{x}$ is the position vector, $!$ is the angular velocity and is the euler angle vector and C and $C_!$ are constants that penalize the octorotor for spinning. However, we found that this error function did not perform well as the constants were hard to tune and the octorotor was not robust to noise.

We considered a change in the reference from $(x = 0; y = 0)$ to $(x = 5; y = 5)$ as the trajectory tracking task for simplification purposes and assuming that the resulting architecture will scale well as long as the changes in the reference are smaller than the one experience during training. Different fault magnitudes must be experienced by the agent to learn how to adapt the position controller parameters. However, we noticed that randomly selecting the fault magnitude for each episode resulted in no convergence. We thus defined a curriculum learning approach where we first expose the agent to the lower bound of the fault magnitude until it converges and then we generate for each episode with probability of 0.5 a fault with maximum magnitude. In this way, we avoid the catastrophic forgetting problem for the agent.

We demonstrate the approach on two different experiments. For both experiments, we use PPO as defined previously with the following parameters:

TABLE II
PPO PARAMETERS

Parameter	Value
Optimizer	ADAM
Batch Size	64
Discount Factor	0.99
Initial Learning Rate	0.0003
Time Steps of an Episode	2000
Total Number of Episodes	3000

In the first experiment, we explore a single fault on motor 3 as defined in figure 3. We initially begin training the DRL agent on a 3x fault (3 times the nominal value of the parameter) and after convergence is reached, we then introduce a larger 8x where at the beginning of the episode, a choice of the fault is made between 3x and 8x with equal probability. We then test our approach by comparing the trajectory between the nominal PD controller and the DRL scheme when the DRL controller is given the exact magnitude of the 8x fault. Furthermore, we then evaluate the robustness of our controller by introducing variance in the estimated parameter such that the value given to the controller is biased. We sample the parameter from a normal distribution with mean value equal to the true value and a deviation of 0.5x the nominal resistance.

For our second experiment, we explore a dual motor fault on motor 5 and 6 as defined in figure 3. However, our controller is now given two values for the estimated motor resistances instead of the one above. For training, we set both motors to an equal 2x fault and allow the DRL agent to learn until convergence is reached. Then, following the same method as proposed above, we introduce a large fault of 4x in both motors such that a choice is made between the smaller and larger fault at the beginning of each episode. Following this, we then test our approach by comparing the trajectory where both motors have a 4x fault. However, to explore the robustness of our approach, we then explore setting motor 6 to a 4x fault and allow the fault of motor 5 to vary. We then introduce variance in the state estimator of both motors such that parameters fed are normally distributed with mean equal to the true value and a standard deviation of 0.5x the nominal resistance. This allows us to explore the effects of large state estimation errors in both of our motors. Furthermore, it also allows us to explore the realistic case of different fault magnitudes on the two different motors. In this exploration, it's worth noting that we are considering multiplicative faults where changes in the motor dynamics may affect the dynamics of the controllers and control allocation requirements. This results in a more complex scenario to handle and showcases the value of using a DRL agent for online adaptation. Furthermore, it is worth highlighting that we do not consider additive faults as they have been widely studied subject [22] and most likely will not need a DRL-based tuning approach.

B. Single-motor faults

For our single-motor fault experiment, we can see in figure 4 that our reward function clearly converges by step 1000 and when the larger fault is introduced, we have a dip in performance, but then converge again by the time training ends. As such, this demonstrates that our scheme first minimizes the error with a small fault, and then builds on it's initial learning by minimizing the error of both the small and larger fault. From figures 5 and 6, we can see that our method outperforms the nominal PD controller in terms of a 8x fault for a single episode. In the X-direction, it is clear that our controller does not overshoot and converges closer to the reference value while the nominal PD controller significantly overcompensates and contains more error in it's convergence. Meanwhile, in the Y-direction, both controllers perform similarly due to the positioning of the motor fault. Furthermore, in 7, we can see that while the median error of smaller faults are outperformed by the nominal PD, when we move to larger faults, our approach can still accurately achieve convergence with a significantly lower error when compared to the nominal PD controller. As such, a parameter estimator can be used to measure the active states of each motor and when a large fault is detected, our controller can be invoked to ensure the stability of the octocopter. Additionally, figure 7 demonstrates that our controller is robust to large

instability as only a marginal set of outliers ever surpass the PD controller's error in 7x, 7.25x, and 8x faults.

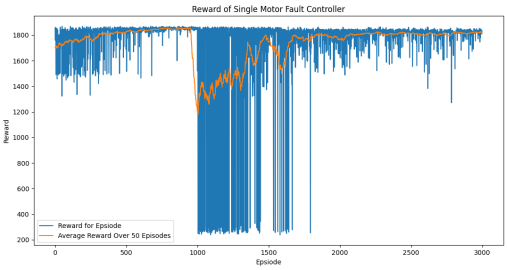

Fig. 4. Reward Function for Training the DRL Agent on a Single Motor 3 Fault

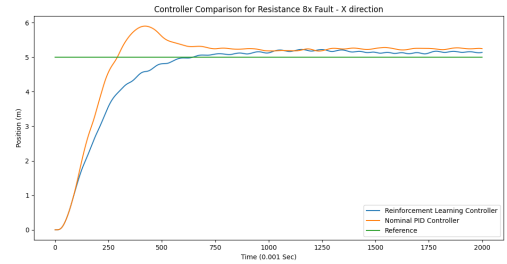

Fig. 5. Comparison of X-trajectory Between PD Control and Hybrid Scheme for 8x Fault on Motor 3

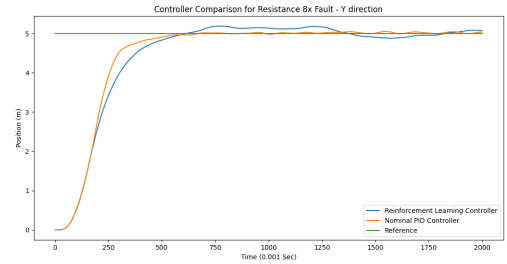

Fig. 6. Comparison of Y-trajectory Between PD Control and Hybrid Scheme for 8x Fault on Motor 3

C. Multiple-Motor faults

Considering a multi-motor fault carries larger impact on the octocopter's performance, we only explore multi-motor faults up to 4x. Similar to above, we can see in figure 8 that our controller first learns to handle the smaller dual fault and then struggles initially when the larger fault is introduced, but ultimately converges to minimize the larger and small fault errors. As such, in a single episode's trajectory, we can see in figure 9 that both the nominal controller and the reinforcement learning scheme converge to the correct X position. However, we can see that the nominal controller performs poorly as it initially becomes unstable and then overcompensates for the fault while our

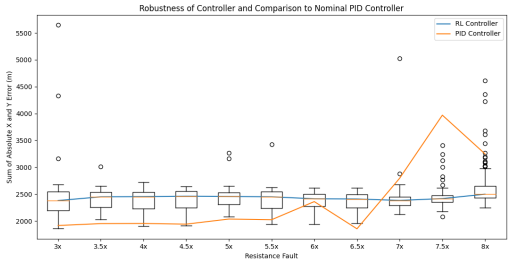

Fig. 7. Robustness of Hybrid Scheme for Single Motor 3 Fault. The robustness is shown over a box-plot where the circles are outliers and the edges of the box represent the first and third quartiles of the data.

approach accurately compensates to ensure a much faster convergence. Furthermore, in the Y-direction show by figure 10, we can see that the reinforcement learning approach slightly overcompensates, but still converges equally fast as the nominal controller. We can see that the approach shown is not perfect in the Y-direction, but the improvements in the X-direction significantly outweigh the marginal overshoot in the Y-direction. Similar to figure 7, we can also see in 11 that the hybrid based control outperforms the sole PD controller at large faults when the PD controller begins to deteriorate in stability. However, in this case, we set a single motor to a 4x fault and vary the fault of the second motor. Despite the variance of the second motor's fault, we see that the controller is robust to parameter estimation noise as even the outliers in the dual motor fault experiment have smaller error than that of the nominal controller. Additionally, this demonstrates that our approach is viable for different magnitude faults in each motor as figure 11 demonstrates an equal or better performance then the PD control scheme at almost every single fault magnitude.

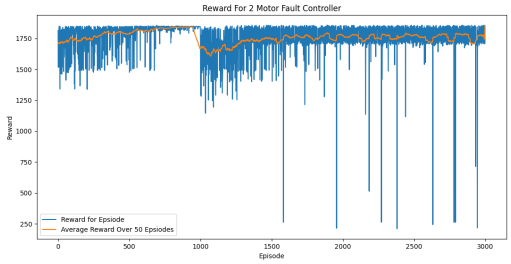

Fig. 8. Reward Function for Training the DRL Agent on a Dual Motor 5 and 6 Faults

VI. CONCLUSIONS

In this paper, we presented a FTC architecture combining parameter estimation, DRL, and model-based control techniques. We tested the approach with an octocopter considering a cascade control scheme subject to single and multiple motor faults with different magnitude. The parameters of the position controllers in the hierarchical control scheme are updated according to the fault magnitude estimated through the