Meta-Learning-Based Proactive Online Planning for UAVs Under Degraded Conditions

Esen Yel, *Member, IEEE*, Shijie Gao, *Graduate Student Member, IEEE*, and Nicola Bezzo, *Member, IEEE*

***Abstract*—Changes in model dynamics due to factors like actuator faults, platform aging, and unexpected disturbances can challenge an autonomous robot during real-world operations affecting its intended behavior and safety. Under such circumstances, it becomes critical to improve tracking performance, predict future states of the system, and replan to maintain safety and liveness conditions. In this letter, we propose a meta-learning-based framework to learn a model to predict the future system's states and their uncertainties under unforeseen and untrained conditions. Meta-learning is considered for this problem thanks to its ability to easily adapt to new tasks with a few data points gathered at runtime. We use the predictions from the meta-learned model to detect unsafe situations and proactively replan the system's trajectory when an unsafe situation is detected (e.g., a collision with an object). The proposed framework is applied and validated with both simulations and experiments on a faulty UAV performing an infrastructure inspection mission, demonstrating safety improvements.**

***Index Terms*—Planning under uncertainty, failure detection and recovery, aerial systems.**

I. Introduction

AUTONOMOUS mobile robots like aerial vehicles are rapidly becoming an integral part of our daily lives thanks to their widespread use in different applications from delivery to inspection. When operating in the real-world, numerous unpredictable challenges (e.g., component faults, external disturbances) can cause performance degradations, potentially leading to unsafe behaviors like collisions. For example, unpredicted wind fluctuations around buildings or bridges or motor failures will put an aerial vehicle at risk of collision while conducting inspection jobs. Since these uncertainties usually occur at runtime without apriori knowledge, it becomes challenging to take them into account during design time.

One way to cope with such unforeseen disturbances is to learn the system model and adapt the controller of the robot at runtime. However, given a well-developed robotic system, it is often the case that the controller is either hard to adapt or is not accessible by the user. Thus, one of the best options is to act on the planner which is by design available to change. By correcting the reference planned trajectory, it is often possible to make a robot with the original controller follow the original desired behavior. As we demonstrated in [1], this type of approach is effective in improving the performance of a robot dealing with an unforeseen component fault. However, we observe that depending on the fault, the tuning of the adapted planner, and the robot's physical limits, an undesired behavior may still occur, like a deviation from the desired path, possibly leading to unsafe situations.

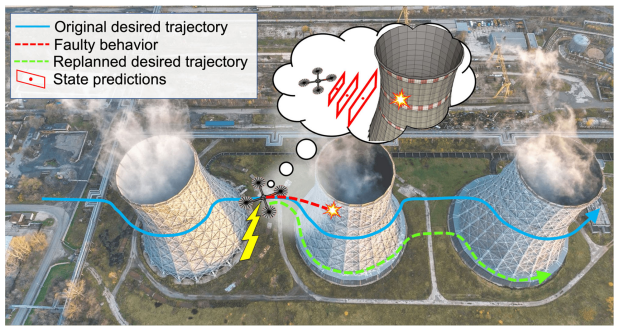


Fig. 1. Pictorial representation of a UAV experiencing a failure at runtime leading to potential collision.

With these premises, in this work, we introduce a novel safety monitoring technique to predict the future states of an autonomous robot under actuator noises and previously unforeseen actuator failures. Our framework predicts the future states and state uncertainties for the faulty robot and utilizes these predictions to monitor if the system will violate safety constraints (e.g., a collision with an obstacle). When an unsafe situation is detected, a safe trajectory is then replanned using a sampling-based approach for waypoint selection. Fig. 1 pictorially shows the problem space of this work in which a UAV is tasked to inspect a power plant. Due to unexpected failures, the UAV will deviate and collide with the plant. Our technique proactively monitors and predicts the regions the system may reach over a future horizon and will update and replan the trajectory when these regions intersect with obstacles.

The contribution of this work is four-fold: 1) we develop a technique to predict the states and regions that a robot under an unknown failure is going to reach within a time horizon; 2) we introduce a runtime monitoring and validation approach to update the prediction models at runtime to increase the prediction accuracy; 3) we propose a replanning approach to prevent unsafe situations at runtime, and 4) we validate with both simulations and experiments on quadrotor UAVs inspection missions.

Manuscript received 24 February 2022; accepted 26 June 2022. Date of publication 18 July 2022; date of current version 4 August 2022. This letter was recommended for publication by Associate Editor S. Zhang and Editor H. Kurniawati upon evaluation of the reviewers' comments. This work was supported by DARPA under Grant FA8750-18-C-0090. *(Esen Yel and Shijie Gao are co-first authors.) (Corresponding author: Shijie Gao.)*

Esen Yel was with the Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA 22904 USA. She is now with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: esenyel@virginia.edu).

Shijie Gao and Nicola Bezzo are with the Departments of Engineering Systems and Environment and Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904 USA (e-mail: sjgao@virginia.edu; nbezzo@virginia.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3191792, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3191792

The rest of the letter is organized as follows: related work about failure rejection and meta-learning is presented in Section II. We introduce the notation and assumptions of the proposed framework in Section III. The problem is then formally defined in Section IV, and we present our proposed meta-learning-based future state and uncertainty prediction approach in Section V. We validate our technique with simulations and experiments in Sections VI and finally draw conclusions and discuss future work in Section VII.

II. Related Work

For consistent performance under faulty behaviors, control techniques have been widely utilized to adapt the systems' control inputs according to the changes in the system dynamics and to alleviate the effects of faults. For example, for quadrotors with the complete loss of one or multiple actuators, specific controllers can be designed according to the failure to maintain stability and performance [2]–[5]. However, these techniques require explicit knowledge of the specific failures and how these changes affect the system's dynamical model to design resilient controllers. When such knowledge is not available, fault identification or adaptive control techniques can be leveraged. In [6], an Extended Kalman Filter (EKF)-based fault identification is used to decide if there are one or multiple rotor failures, and a control allocation is updated based on the failure using a nonlinear Model Predictive Control (MPC). In [7], a self-reconfiguration technique allows the system to decide on its configuration based on the actuator failure and its desired trajectory. Another well-known adaptive control technique called Model Reference Adaptive Control (MRAC) adapts the control variables of the system based on the difference between the observations and reference model output to improve tracking for systems with uncertainties and it has been used to compensate for failures [8], [9]. Recently, machine learning techniques such as Gaussian Processes (GP) [10] and deep neural networks (DNNs) [11] have been leveraged for the adaptive elements in MRAC frameworks. GP-based approaches have also been used to model the effects of changes in the system model (e.g., due to unknown payload mass) and to provide safe plans [12]. Robustness against faulty systems has also been achieved by using resilient distributed consensus [13], [14], and topology control [15] within the context of multi-agent navigation.

In addition to control approaches, machine learning techniques have also been widely used to improve the performance of UAVs under actuator faults or disturbances. In [16], the authors use MPC with active learning to learn the robot's new model under failure and provide necessary inputs. Reinforcement Learning (RL) techniques are also utilized to adjust the actuator control commands to compensate for component faults [17], [18]. Meta-learning approaches enable systems to speed up their learning process for new tasks with a small number of training samples from new tasks. This property makes meta-learning suitable for learning the models of uncertain systems at runtime for safe planning [19]. One of such techniques–Model-Agnostic Meta-Learning (MAML)–trains the model parameters explicitly to make them easy and fast to fine-tune for a new task [20]. MAML has been leveraged for fault-tolerant operations using MPC and RL in [21], [22]. An algorithm called Fast Adaptation through Meta-Learning Embeddings (FAMLE) is proposed in [23] that meta-learns multiple priors as opposed to a single prior in MAML, and picks the most likely prior to improve online learning efficiency. [24] introduces a concept of meta-active learning in which a Q-function is learned via meta-learning and used to find optimal actions to maximize the probability of staying in the safe region and promote information gain for systems with altered dynamics. In [25], meta-learning is utilized to model the system dynamics under external forces to be used with an adaptive control scheme to improve the tracking performance. All of these approaches assume that the user is given direct access to the controller or the actuator inputs. However, this assumption may not hold, especially when off-the-shelf robotic systems are used.

As mentioned in Section I, in our previous work [1] we have proposed a meta-learning-based approach to recover a faulty UAV undergoing an unforeseen fault, updating the reference trajectory at runtime without accessing the controller or control inputs. In this letter, we extend this work and the use of meta-learning to solve a reachability analysis problem to deal with situations in which even after replanning, the system may still deviate from the desired behavior possibly leading to unsafe states. In particular, with our proposed framework, future states and associated uncertainties for a faulty autonomous system considering future corrections (e.g., by leveraging our reference update method in [1]) are predicted at runtime without the need of retraining a learning component. These predictions are then utilized to monitor if the reference update method is insufficient to make the system follow its desired trajectory and remain safe, triggering a trajectory replanning procedure.

III. Preliminaries

In this work, we assume that once a robot is deployed to perform an operation, we do not have access to its controller or control inputs and we only have access to the high-level planner and reference trajectory generator. This consideration is made in order to increase realism since most robotic systems do not allow manipulations of the controller once a robot is deployed but instead, it is possible to change its goals locations and trajectory waypoints. We assume that the system is already applying corrective counter-measures both at design time and at runtime to alleviate the effects of faults and to follow a desired trajectory closely. However, under some disturbances/failures, its behavior may still become erratic as the corrective actions may not be able to compensate and bring the system back on its desired path.

A. Notations

In this letter we use $\boldsymbol{x}(k)$ to represent the state of the system at time k. $\boldsymbol{p}(k)$ and $\boldsymbol{v}(k)$ represent the position and velocity of the system respectively. The symbol $\tilde{\boldsymbol{x}}$ is used to represent the predicted state, and the symbol $\bar{\boldsymbol{x}}$ is used to represent the mean of sampled states. The notation $x(k : k_N)$ represents an array of values from time k to k_N: $x(k : k_N) = [x(k), x(k+1), \ldots, x(k_N)]^\mathsf{T}$ where $k_N > k$. The notation $x(k : \delta_N : k_N)$ represents an array of values from time k to k_N with $\delta_N \in \mathbb{Z}^+$ increments: $x(k : \delta_N : k_N) = [x(k), x(k+\delta_N), x(k+2\delta_N), \ldots, x(k_N)]^\mathsf{T}$ where $k_N > k$ and $\delta_N > 1$.

IV. Problem Formulation

The goal of this work is to design a technique that predicts the future states of a system under an unseen fault at runtime, uses these predictions to detect unsafe situations, and proactively

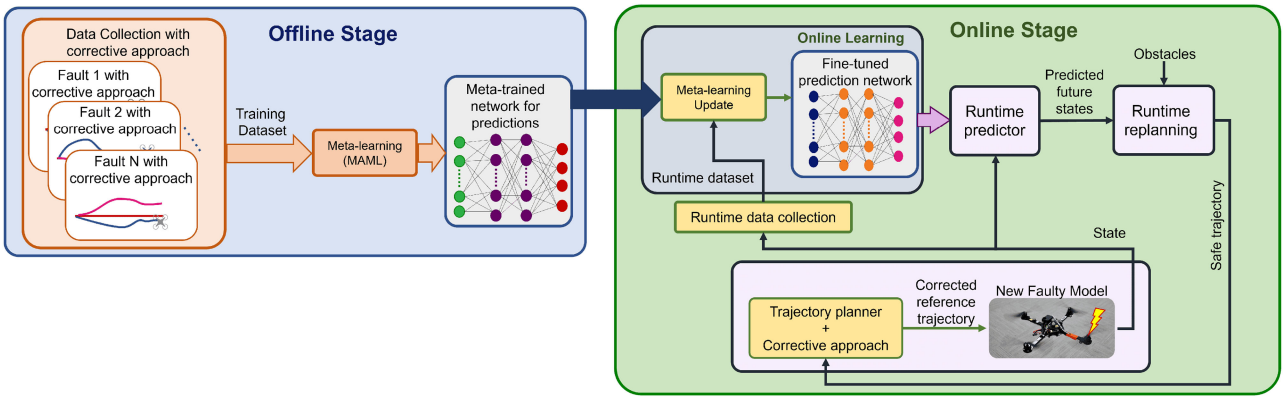


Fig. 2. Meta-learning-based future state prediction and replanning framework for systems under unknown faults.

replans to improve safety. The system might already be applying corrective actions to compensate for the experienced faults, however, these corrective measures may not be enough to prevent collisions. Formally, we define these problems as follows:

Problem 1: *Future State Prediction under Failure:* An autonomous system with a nominal dynamical model $f(\boldsymbol{x}, \boldsymbol{u})$ as a function of its states $\boldsymbol{x}$ and controller inputs $\boldsymbol{u}$ has an objective of following a predefined desired trajectory $\boldsymbol{x}_\tau$. Under actuator faults and noises, the system's model changes to $\boldsymbol{x}(k+1) = f'(\boldsymbol{x}(k), \tilde{\boldsymbol{u}}(k))$, where f' is unknown. The system has noisy inputs $\tilde{\boldsymbol{u}}(k)) = \boldsymbol{u}(k) + \boldsymbol{\eta}(k)$ which create state uncertainties, making the exact state prediction of the system challenging. With these premises, our goal is to design a predictor to map the future state predictions ($\tilde{\boldsymbol{x}}$) and state uncertainty predictions ($\tilde{\boldsymbol{\zeta}}$) as a function ($\tilde{h}$) of the history of states and reference trajectory: $[\tilde{\boldsymbol{x}}(k:k+H), \tilde{\boldsymbol{\zeta}}(k:k+H)] = \tilde{h}(\boldsymbol{x}(k-T:k-1), \boldsymbol{x}_\tau)$. H is the state prediction horizon and T is the size of the data history used to make predictions.

Problem 2: *Safe Replanning:* Design an online policy to monitor the safety of the future states of the system and to replan the trajectory when an unsafe situation is detected to ensure that the following safety condition will be satisfied by the future predicted states:

$$R_{\boldsymbol{p}}|_k^{k+H} \cap \mathcal{O} = \emptyset \tag{1}$$

where $\mathcal{O}$ is the set of obstacle positions and $R_{\boldsymbol{p}}|_k^{k+H}$ is the union of future position sets that the system is predicted to reach over a time horizon H with time increments of δ_H:

$$R_{\boldsymbol{p}}|_k^{k+H} = R_{\boldsymbol{p}}(k) \cup R_{\boldsymbol{p}}(k+\delta_H) \cdots \cup R_{\boldsymbol{p}}(k+H) \tag{2}$$

The set of positions that the system is predicted to reach at time k is computed as follows:

$$R_{\boldsymbol{p}}(k) = \cup \left\{ \boldsymbol{p} \text{ s.t. } \|\boldsymbol{p} - \tilde{\boldsymbol{p}}(k)\| \leq \tilde{\boldsymbol{\zeta}}_{\boldsymbol{p}}(k) \right\} \tag{3}$$

where $\tilde{\boldsymbol{p}}$ is the predicted position and $\tilde{\boldsymbol{\zeta}}_{\boldsymbol{p}}$ is the predicted position uncertainty.

V. Meta-Learning-based Predictions and Proactive Replanning

Our framework consists of offline and online stages as depicted in Fig. 2. During offline and online stages, the system applies corrective measures to compensate for the faults. During the offline stage, the robot under various faults follows a set of trajectories. A meta-network is trained offline to predict future states and their uncertainties based on the collected training data. At runtime, the robot experiences a new, unforeseen fault. With a few data collected at runtime, the meta-network is fine-tuned to make predictions over a finite horizon about the future states and state uncertainties of the new faulty system considering corrective countermeasures. These predictions are used within a runtime replanning approach to find a safe trajectory if the original desired trajectory is deemed unsafe with the fault that the system is experiencing. The next section will explain how the offline training for state and uncertainty predictions is performed.

A. Offline Training for State and Uncertainty Predictions

To train a model offline for state and uncertainty predictions, we first collect training data using a UAV with various faults following a rich set of trajectories. Then, we use meta-learning to train a network which is easy to fine-tune at runtime using a small number of data [20].

1) Data Collection: During the offline stage, a dataset is created using the data collected from a UAV with actuator fault from a discrete fault set $\mathcal{F}$ while it is following different trajectories applying fault-tolerant corrective measures.

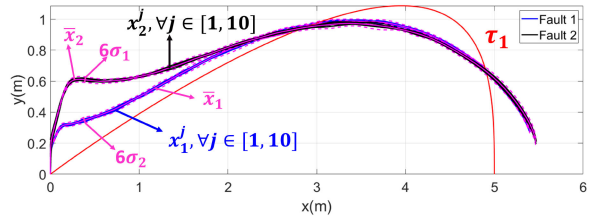
Specifically, we consider a system with faulty dynamics and actuator noise modeled as follows:

$$\boldsymbol{x}(k+1) = f'\left(\boldsymbol{x}(k), \boldsymbol{u}(k) + \boldsymbol{\eta}(k)\right) \tag{4}$$

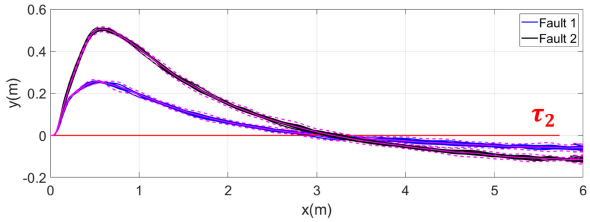
The actuator noise $\boldsymbol{\eta}(k) \sim \mathcal{N}(\boldsymbol{\mu}_\eta, \boldsymbol{\sigma}_\eta)$ is sampled from a normal distribution with mean $\boldsymbol{\mu}_\eta$ and standard deviation $\boldsymbol{\sigma}_\eta$. Each trajectory is run N times and for each sampled run, the mean of the actuator noise is sampled from a normal distribution to capture the behavior of the system under various uncertainties: $\boldsymbol{\mu}_\eta \sim \mathcal{N}(\bar{\boldsymbol{\mu}}, \boldsymbol{\sigma}_\mu)$.

For each trajectory $\tau \subset \mathcal{T}$, we compute the mean and standard deviation of the N sampled paths for each fault $\mathcal{F}_i \subset \mathcal{F}$ and for each discrete sample times $k \in [0, T_\tau]$.

$$\bar{\boldsymbol{x}}_i(k) = \frac{\sum_{j=1}^{N} \boldsymbol{x}_i^j(k)}{N}, \boldsymbol{\sigma}_i(k) = \sqrt{\frac{\sum_{j=1}^{N} |\boldsymbol{x}_i^j(k) - \bar{\boldsymbol{x}}_i(k)|^2}{N-1}}$$
$$\forall k \in [0, T_\tau], \forall i \in \{1, \ldots, |\mathcal{F}|\} \tag{5}$$



(a) Desired trajectory 1 followed by two different faulty systems.



(b) Desired trajectory 2 followed by two different faulty systems.

Fig. 3. Sample desired trajectories with two different faulty systems.

In Fig. 3 we show two different sample desired trajectories ($\boldsymbol{\tau_1}$ and $\boldsymbol{\tau_2}$) that are followed by a UAV under two different faults applying the reference update procedure described in [1] to reduce their tracking error. The first faulty UAV has reduced thrust in one actuator: $\mathcal{F}_1 \rightarrow T_1' = T_1 - 0.25\,\text{N}$ and the second faulty system has more degradation on the same actuator: $\mathcal{F}_2 \rightarrow T_1' = T_1 - 0.5\,\text{N}$. Roll and pitch angles of both systems are limited to $|\phi| \leq \frac{\pi}{18}, |\theta| \leq \frac{\pi}{18}$ respectively. Each trajectory is run $N = 10$ times with different actuator noise sampled as explained above. Blue curves in Fig. 3(a) show these N sampled paths for the first fault and the magenta curve in the middle shows the mean of these samples ($\bar{\boldsymbol{x}}_1$). Dashed magenta curves show the uncertainty around the mean of the samples with $\pm 3\boldsymbol{\sigma}_i$. Fig. 3(b) shows the paths of the same faulty UAVs following another training trajectory ($\boldsymbol{\tau_2}$). During training, we create 100 training trajectories using minimum-jerk trajectory generation [26] with different final positions, initial and final velocities. Determining the training data size for training an accurate meta-model is an open problem and beyond the scope of this work. This framework leaves the choice of training data size to the user. However, as for most learning components, it benefits from training data that covers a wide variety of faults.

2) Meta-Network Training: The purpose of meta-learning is to train an easily adaptable model to predict the future positions and position uncertainties of a faulty system. We denote this learning model as h which takes the history of the system's observed states, history of the desired states and future desired states as inputs and returns the future states and state uncertainties as an output. A training input $\boldsymbol{\chi}_h^i$ and a training output $\boldsymbol{\gamma}_h^i$ are constructed as follows:

$$\boldsymbol{\chi}_h^i(k) = \begin{bmatrix} \boldsymbol{\xi}_{\boldsymbol{x}}^i(k) \\ \boldsymbol{\xi}_{\boldsymbol{\tau}}^i(k) \\ \boldsymbol{\xi}_h^i(k) \end{bmatrix} \qquad \boldsymbol{\gamma}_h^i(k) = \begin{bmatrix} \boldsymbol{\xi}_{\boldsymbol{\gamma}}^i(k) \\ \boldsymbol{\zeta}_{\boldsymbol{\gamma}}^i(k) \end{bmatrix} \tag{6}$$

where $\boldsymbol{\xi}_{\boldsymbol{x}}^i(k)$, $\boldsymbol{\xi}_{\boldsymbol{\tau}}^i(k)$ and $\boldsymbol{\xi}_h^i(k)$ represents the vectors related to the history of the system's states, history of the desired states and future desired states respectively and $\boldsymbol{\xi}_{\boldsymbol{\gamma}}^i(k)$ and $\boldsymbol{\zeta}_{\boldsymbol{\gamma}}^i(k)$ are the vectors related to the future state and state uncertainty predictions respectively. For the UAV application considered in this letter, the input vectors for a quadrotor with the fault $\mathcal{F}_i \subset \mathcal{F}$ are constructed as follows:

$$\boldsymbol{\xi}_{\boldsymbol{x}}^i(k) = \begin{bmatrix} \bar{x}_i(k-T+1:k) - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ \bar{y}_i(k-T+1:k) - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ \bar{v}_i^x(k-T+1:k) \\ \bar{v}_i^y(k-T+1:k) \end{bmatrix}$$

$$\boldsymbol{\xi}_{\boldsymbol{\tau}}^i(k) = \begin{bmatrix} x_\tau(k-T+1:k) - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ y_\tau(k-T+1:k) - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}(k-T+1:k) \\ v_{y,\tau}(k-T+1:k) \end{bmatrix}$$

$$\boldsymbol{\xi}_h^i(k) = \begin{bmatrix} x_\tau(k+\delta_H:\delta_H:k+H) - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ y_\tau(k+\delta_H:\delta_H:k+H) - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}(k+\delta_H:\delta_H:k+H) \\ v_{y,\tau}(k+\delta_H:\delta_H:k+H) \end{bmatrix}$$

$$\forall \tau \subset \mathcal{T}, \qquad \forall k \in \{1,\ldots,T(\tau)\} \tag{7}$$

where $\bar{\boldsymbol{p}}_i(k) = [\bar{x}_i(k), \bar{y}_i(k)]$ and $\bar{\boldsymbol{v}}_i(k) = [\bar{v}_i^x(k), \bar{v}_i^y(k)]$ are the position and velocity components of the mean state $\bar{\boldsymbol{x}}_i(k)$ respectively. For this application, we use the positions and velocities in $x-y$ plane as part of the observed and desired states, however, it should be noted that, depending on the application, higher order states such as acceleration or jerk values could also be added into the network input. Similarly, the output vectors are constructed as follows:

$$\boldsymbol{\xi}_{\boldsymbol{\gamma}}^i(k) = \begin{bmatrix} \bar{x}_i(k+\delta_H:\delta_H:k+H) - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ \bar{y}_i(k+\delta_H:\delta_H:k+H) - \bar{y}_i(k-T)\vec{\mathbf{1}} \end{bmatrix}$$

$$\boldsymbol{\zeta}_{\boldsymbol{\gamma}}^i(k) = \begin{bmatrix} 3 \cdot \max(\sigma_i^x(k:k+H)) \\ 3 \cdot \max(\sigma_i^y(k:k+H)) \end{bmatrix} \tag{8}$$

where $\sigma_i^x(k)$ and $\sigma_i^y(k)$ are the x and y position components of the standard deviation $\boldsymbol{\sigma}_i(k)$ respectively.

The dataset for meta-learning training $\mathcal{D}_i^H$ for fault $\mathcal{F}_i$ contains the training input matrix $\boldsymbol{X}_h^i$ and output matrix $\boldsymbol{Y}_h^i$, with the columns $\boldsymbol{\chi}_h^i$ and $\boldsymbol{\gamma}_h^i$ respectively. The training dataset for meta-learning contains the dataset for each fault: $\mathcal{D}_i^H \subset \mathcal{D}^H$.

The purpose of meta-learning is to learn a model represented by a parameterized function $h_{\boldsymbol{\phi}}$ that maps the model input to the output. We use MAML [20] as a meta-learning algorithm to train the network. During the offline training, the model parameters vector $\boldsymbol{\phi}$ are meta-optimized according to 1 in [20]:

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \beta \nabla_{\boldsymbol{\phi}} \sum_{\mathcal{F}_i \subset \mathcal{F}} \mathcal{L}_{\mathcal{F}_i}(h_{\phi - \alpha \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\mathcal{F}_i}(h_{\boldsymbol{\phi}})}) \tag{9}$$

where α is the learning step size, β is the meta step size, and $\mathcal{F}_i \subset \mathcal{F}$ indicates the sample batch of faults among the training data. For more details, we refer readers to [20].

This meta-optimization allows the parameters to be quickly fine-tuned with a few data at runtime. The loss function used during this training is given as follows:

$$\mathcal{L}_{\mathcal{F}_i}(h_{\boldsymbol{\psi}}) = \sum_{\boldsymbol{\chi}_h^i, \boldsymbol{\gamma}_h^i \in \mathcal{D}_i^H} \| h_\psi\left(\boldsymbol{\chi}_h^i\right) - \boldsymbol{\gamma}_h^i \|_2^2 \tag{10}$$

where $\boldsymbol{\chi}_h^i$ and $\boldsymbol{\gamma}_h^i$ are given in (6).

B. Online Meta-Network Update

At runtime, the UAV may experience a new fault which is not included in the training set and may apply the same corrective measures as during the training stage. While the system moves under this new fault, we collect K_p consecutive data from its state sensors to update the offline meta-trained model for future state predictions. The inputs and outputs of the online learning data set are constructed in the same way as in (6) and (7):

$$\boldsymbol{\chi}_h^*(k) = \begin{bmatrix} \boldsymbol{\xi}_{\boldsymbol{x}}^*(k) \\ \boldsymbol{\xi}_{\boldsymbol{\tau}}^*(k) \\ \boldsymbol{\xi}_h^*(k) \end{bmatrix} \qquad \boldsymbol{\gamma}_h^*(k) = \begin{bmatrix} \boldsymbol{\xi}_{\boldsymbol{\gamma}}^*(k) \\ \boldsymbol{\zeta}_{\boldsymbol{\gamma}}^*(k) \end{bmatrix} \tag{11}$$

where:

$$\boldsymbol{\xi}_{\boldsymbol{x}}^*(k) = \begin{bmatrix} x^*(k-T+1:k) - x^*(k-T)\vec{\mathbf{1}} \\ y^*(k-T+1:k) - y^*(k-T)\vec{\mathbf{1}} \\ v_x^*(k-T+1:k) \\ v_y^*(k-T+1:k) \end{bmatrix}$$

$$\boldsymbol{\xi}_{\boldsymbol{\tau}}^i(k) = \begin{bmatrix} x_\tau^*(k-T+1:k) - x^*(k-T)\vec{\mathbf{1}} \\ y_\tau^*(k-T+1:k) - y^*(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}^*(k-T+1:k) \\ v_{y,\tau}^*(k-T+1:k) \end{bmatrix}$$

$$\boldsymbol{\xi}_h^i(k) = \begin{bmatrix} x_\tau^*(k+\delta_H:\delta_H:k+H) - x^*(k-T)\vec{\mathbf{1}} \\ y_\tau^*(k+\delta_H:\delta_H:k+H) - y^*(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}^*(k+\delta_H:\delta_H:k+H) \\ v_{y,\tau}^*(k+\delta_H:\delta_H:k+H) \end{bmatrix} \tag{12}$$

for $k \in \{T+1, \ldots, T+K_p\}$ with $\boldsymbol{p}^* = [x^*, y^*]$ and $\boldsymbol{v}^* = [v_x^*, v_x^*]$ the position and velocity of the UAV with an unknown fault at runtime. $\boldsymbol{p}_\tau^* = [x_\tau^*, y_\tau^*]$ and $\boldsymbol{v}_\tau^* = [v_{x,\tau}^*, v_{y,\tau}^*]$ are desired trajectory positions and velocities respectively. The output of the online learning dataset consists of the following vectors:

$$\boldsymbol{\xi}_{\boldsymbol{\gamma}}^i(k) = \begin{bmatrix} x^*(k+\delta_H:\delta_H:k+H) - x^*(k-T)\vec{\mathbf{1}} \\ y^*(k+\delta_H:\delta_H:k+H) - y^*(k-T)\vec{\mathbf{1}} \end{bmatrix}$$

$$\boldsymbol{\zeta}_{\boldsymbol{\gamma}}^i(k) = [\,\sigma_x\ \sigma_y\,]^\mathsf{T} \tag{13}$$

where σ_x and σ_y are assigned uncertainties in x and y directions respectively and they are initially set to a value larger than the mean of the observed uncertainties during training. By using the data collected at runtime, the meta-learned model parameters $\boldsymbol{\phi}$ are updated to $\boldsymbol{\phi}^*$ using only a few stochastic gradient descent updates. The updated model $h_{\boldsymbol{\phi}^*}$ is then used to make predictions for the future states of the system for the same horizon considered in training. These predictions are used to replan trajectories if unsafe situations are detected, as explained in the next section.

1) Runtime Validation: After the initial meta-network update at runtime, the runtime inputs are compared to the training inputs to assess if a further meta-network update is necessary or not. The distance between the runtime input and training inputs with training faults is calculated as:

$$d_{\mathcal{F}}^i(k) = \min_{\boldsymbol{\chi}_h^i \in \mathrm{col}(\boldsymbol{X}_h^i)} \|\boldsymbol{\chi}_h^*(k) - \boldsymbol{\chi}_h^i\| \quad \forall i \in \{1, \ldots, |\mathcal{F}|\}$$
$$\forall k \in \{T+K_p, \ldots, T(\tau)\} \tag{14}$$

If the minimum distance between the observed test input and the training inputs is larger than a given threshold, the system re-tunes its meta-trained network:

$$s_H(k) = 1 \qquad \text{if } \min_{i \in \{1,\ldots,|\mathcal{F}|\}} \left(d_{\mathcal{F}}^i(k)\right) > \lambda_H \tag{15}$$

where s_H is a binary variable that enables re-updating the meta-trained network at runtime using the last K_p runtime training inputs and λ_H is a user-defined threshold.

We also constantly monitor the observed state to check if it is outside of the predicted reachable region. If so, the network is re-tuned:

$$s_H(k) = 1 \qquad \text{if } \boldsymbol{p}(k) \not\subset \tilde{R}_{\boldsymbol{p}}(k) \tag{16}$$

where $\tilde{R}_{\boldsymbol{p}}(k)$ is a region where the system is predicted to reach at time k:

$$\tilde{R}_{\boldsymbol{p}}(k) = \cup\{\boldsymbol{p} \text{ s.t. } \|\boldsymbol{p} - \tilde{\boldsymbol{p}}(k)\| \le \tilde{\boldsymbol{\zeta}}(k)\} \tag{17}$$

C. Runtime Replanning for Safety

After updating the meta-trained network, the future states and state uncertainties of the system are predicted using the fine-tuned network:

$$\begin{bmatrix} \tilde{x}(k+\delta_H:\delta_H:k+H) \\ \tilde{y}(k+\delta_H:\delta_H:k+H) \\ \tilde{\sigma}_x(k+\delta_H:\delta_H:k+H) \\ \tilde{\sigma}_y(k+\delta_H:\delta_H:k+H) \end{bmatrix} = \boldsymbol{h}_\phi^*\left(\boldsymbol{\chi}_h^*(k)\right) + \begin{bmatrix} x^*(k-T)\vec{\mathbf{1}} \\ y^*(k-T)\vec{\mathbf{1}} \\ 0 \\ 0 \end{bmatrix}$$
$$\forall k \ge T + K_p \tag{18}$$

At runtime, the predicted set based on the updated meta-trained model are used to proactively detect unsafe situations. Given an environment with a set of static obstacles $\boldsymbol{\mathcal{O}}$, the regions that the system is predicted to reach, which are computed as in (17), are checked for collision:

$$s^*(k+t) = \begin{cases} 0 & \text{if } \tilde{R}_{\boldsymbol{p}}(k+t) \cap \boldsymbol{\mathcal{O}} \neq \emptyset \\ 1 & \text{otherwise} \end{cases}$$
$$\forall t \in \{\delta_H, 2\delta_H, \ldots, H\} \tag{19}$$

At time k, if it is detected that $s^*(k+t) = 0$ (i.e., reachable regions intersect with obstacles for $t \in \{\delta_H, 2\delta_H, \ldots, H\}$), the trajectory is replanned. To this end, here we use a sampling-based replanning method in which a waypoint around the original unsafe desired trajectory point is generated and tested for safety until a safe waypoint is found as outlined in Algorithm 1.

Algorithm 1: Trajectory Replanning.

$\tau \leftarrow$ Initialize the desired trajectory
$s^*(k+t) \leftarrow$ Assess safety based on (19)
while $s^*(k+t) = 0$ **do**
  $d_s \sim \mathcal{U}_{[0,\bar{d}_s]} \leftarrow$ Sample update distance
  $\psi_s \sim \mathcal{U}_{[-\pi,\pi]} \leftarrow$ Sample update direction
  $\boldsymbol{p}_w = \boldsymbol{p}_\tau(k+t) + d_s[cos(\psi); sin(\psi)] \leftarrow$ Sample a waypoint
  $\tau \leftarrow$ Replan trajectory with $\boldsymbol{p}_w$
  $\tilde{R}_{\boldsymbol{p}}(k) \leftarrow$ Predict the reachable region with τ
  $s^*(k+t) \leftarrow$ Assess the safety of τ based on (19)
end while
return τ

The replanning algorithm is applied until the desired trajectory reaches the goal location. It should be noted that as the main

TABLE I
FAULT TYPES USED DURING SIMULATIONS

Training fault	Fault type
$\mathcal{F}_1$	66% thrust on motor 1 and 136% on motor 3
$\mathcal{F}_2$	43% thrust on motor 1 and 160% on motor 3
$\mathcal{F}_3$	21% thrust on motor 1 and 184% on motor 3
Test fault	**Fault type**
${\mathcal{F}_1}^*$	32% thrust on motor 1 and 173% on motor 3

objective of this work is not the replanning algorithm itself, a user can consider a different replanning approach based on the application.

VI. SIMULATIONS AND EXPERIMENTS

The proposed meta-learning-based predictive and proactive replanning framework was validated on a faulty UAV infrastructure inspection task. For the following simulations and experiments, we consider a quadrotor UAV that is undergoing a faulty behavior and is equipped with a fault-tolerant corrective method which may or may not be well-tuned to recover the system against all possible failures/disturbances that can occur at runtime. Thus, the vehicle can deviate from its desired trajectory when facing a new fault beyond the corrective method's capabilities. In our simulations and experiments, we chose to use our meta-learning-based trajectory update method presented in [1] as a fault-tolerant corrective method. This technique has shown trajectory tracking improvements under degraded conditions. The corrective method is purposely poorly tuned to show not only the issue more clearly but also the effectiveness of the proposed approach. It is worth noting that we choose to use this method because it does not require access to the controller or controller inputs, however, our predictive and proactive planning framework can be used with other corrective/adaptive techniques as well (e.g., robust and adaptive controllers).

A. Simulations

In these simulations, the quadrotor is modeled with a 12-dimensional state vector [26] and the fault is modeled as a thrust change on randomly selected motors. Specifically, the fault is simulated by both reducing the thrust on one of the motors and increasing the thrust on the opposite motor of the quadrotor. The details of the faults used during training and testing are shown in Table I. Note that the faults caused by motors 1 and 3 are amplified effects of the fault on the single motor. Considering the quadrotors are symmetric by nature, faults presented on 2 or 4 can be treated similarly. In addition to the faults, we also consider a system with limited roll and pitch angles: $|\phi| \leq \tau_\phi, |\theta| \leq \tau_\theta$ in order to accentuate the issue and for ease of demonstration. For training, we used two different angle limits: $\tau_\phi = \tau_\theta \in \{\frac{\pi}{18}, \frac{\pi}{12}\}$ and for testing we considered a different angle limit: $\tau_\phi = \tau_\theta = \frac{\pi}{16}$.

During the offline stage of collecting training data, the UAV is tasked to follow different minimum-jerk trajectories [26] under different faults. Specifically, the training trajectories are shaped as trapezoidal paths of different lengths and angles. The collected data are used for meta-training the reference trajectory neural network and the state and uncertainty prediction neural network.

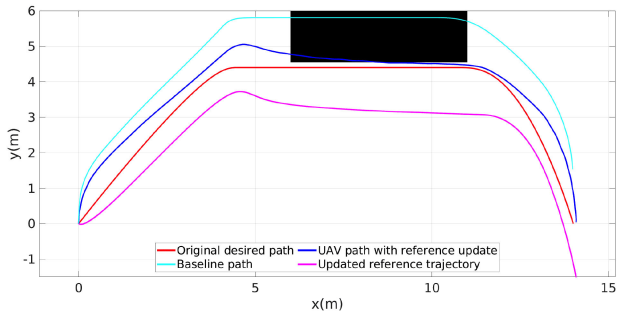


Fig. 4. UAV path under a test fault without the meta-learning prediction and replanning approach.

During training, the control loop runs at 40 Hz. We use $T = 10$ past data to predict the future states, and we set the future horizon for the predictions $H = 50$ steps which is equivalent to $1.25\,\mathrm{s}$ ahead. The state and the uncertainty predictions are given at five future times spaced $\delta^h = 10$ time steps apart. The prediction network contains five hidden layers with 100 nodes and is meta-trained by using a Tensorflow Keras implementation of MAML [20].

During the online testing stage, the UAV is tasked to follow a desired path to inspect a structure while undergoing an unexpected failure. In Fig. 4 we show a baseline case that will be used to compare our framework. Without correction, the UAV collides with the black obstacle (cyan-colored baseline path) while following the desired trajectory (red-colored line). In the same figure, we show also the case in which the UAV only applies corrective reference updates according to [1] (blue line) while trying to follow the desired path without the proposed predictive and proactive replanning technique. The UAV adapts the trained model to deal with the new fault by using prior data obtained during the flight and then uses the adapted model to update the reference trajectory (magenta line in the figure). It should be noted that, as a proof of concept, we used a poorly tuned correction which leads to collisions, to demonstrate the prediction and compensation capabilities of our approach next.

Given the same simulation setup, we validate our prediction and replanning technique while the UAV is applying the same reference trajectory update in [1]. Fig. 5 shows that the UAV predicts a potential collision within the predicting horizon ($1.25\,\mathrm{s}$) and replans its desired path to avoid the obstacle. In the zoomed-in window, we show the instance in which the UAV predicts and detects the collision and keeps predicting and checking the safety of the replanned paths. The proposed replanned desired paths and the predictions for checking the safety are color coded. Given the first proposed replan trajectory which is shown as the dashed brown line, the framework predicts the future positions as well as the uncertainties of the vehicle which are indicated as a series of brown rectangles in the zoomed-in window. The first proposed replan is considered as unsafe since the predicted positions collide with the obstacle and thus it is abandoned. Another replanned trajectory orange-colored is considered but yet deemed unsafe. Finally, a safe replanned desired trajectory (green dashed line) is validated by the predictions of the framework (green rectangles) and the vehicle can overcome the unsafe situation. As the goal of our proposed framework is to recover the faulty vehicle from the unsafe operations, the path replanning prioritize getting the UAV safely to the desired destination over staying close to the initial desired path.

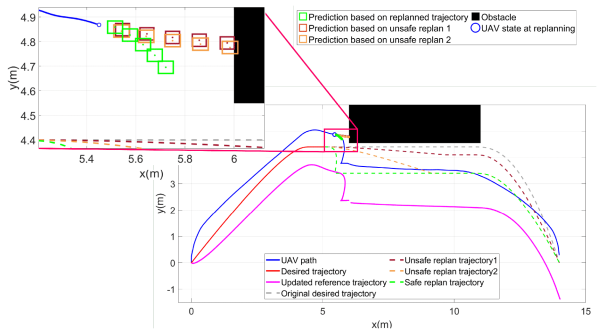
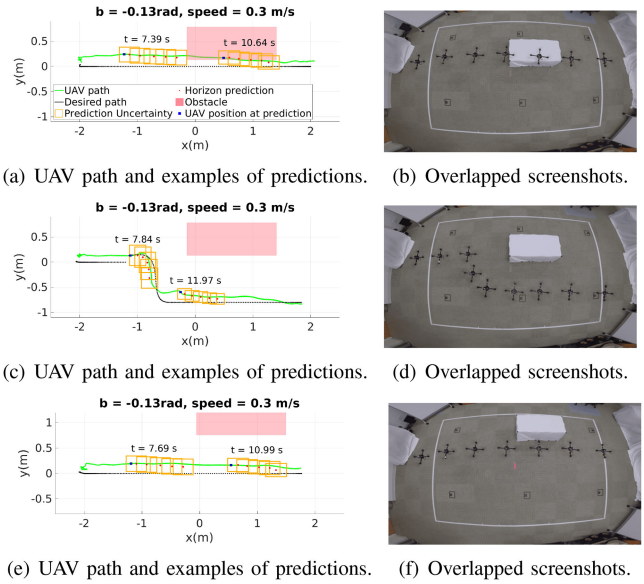


Fig. 5. UAV path under a test fault with the proposed meta-learning-based prediction and replanning approach.



(a) UAV path and examples of predictions. (b) Overlapped screenshots.

(c) UAV path and examples of predictions. (d) Overlapped screenshots.

(e) UAV path and examples of predictions. (f) Overlapped screenshots.

Fig. 6. The UAV with an unknown fault is tasked to fly from the start point to a destination in 3 different cases. Case 1 (a) and (b) shows the UAV detects a collision while replanning is disabled. Case 2 (c) and (d) shows the UAV detects a potential collision and avoids the obstacle by replanning. Case 3 (e) and (f) shows the obstacle does not threaten the safety of the UAV and no replanning is thus triggered.

B. Experiments

The proposed meta-learning-based prediction and proactive re-planning approach was also validated with experiments by using an Asctec Hummingbird quadrotor UAV inside a controlled laboratory space. Similar to the simulations, the UAV uses a baseline PID controller which is designed for the nominal quadrotor (without a fault) to control its position and attitude. To create the faulty behavior on motors 1 and 3, a fault is injected into the system by adding a bias to the commanded pitch angle before it is fed to the attitude controller. A poorly tuned meta-learning based reference trajectory update approach [1] is used to compensate for the deviation caused by the fault during tracking. The experiments are implemented in ROS and a Vicon motion capture system is used to monitor and track the state of the UAV.

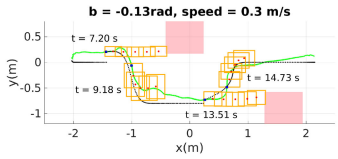
To train a model for future state prediction, we collected training data for the UAV following different trajectories considering various speeds and angled paths. We ran each of the training trajectories with 5 different faults by directly adding biases $b \in \{-0.06, -0.09, -0.12, -0.15, -0.18\}$rad to the pitch command and three runs for each case. As the magnitude of the faults increases, the quadrotor deviates more toward the positive y-direction. A meta-learning model was trained with the collected data to predict the states as well as the uncertainty of the vehicle at different time frames $\{+0.4, +0.8, +1.2, +1.6, +2\}$s in the future. The same architecture of the neural networks in the simulation was used for the experiments.

During the online stage, the vehicle was tasked to follow a desired trajectory at a maximum velocity of $3\,\mathrm{m/s}$ while a bias $b = 0.13$rad – which is different from the ones in the training set – was applied to the vehicle. While the UAV was tracking the desired path, it used the offline meta-trained model to predict the positions and uncertainties at 5 different time frames in the future with the maximum predicting horizon of $2\,\mathrm{s}$. Fig. 6 shows the results of the UAV taking a desired straight path from the start point to the destination. The predictions are shown as the orange bounding boxes in the figures. Any overlapping area between the predicted boxes and the obstacles is considered a potential collision and thus triggers the re-planning procedure.

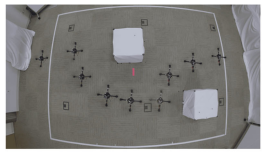
Fig. 6(a) and 6(b) show the results from the first experiment in which the re-planning procedure was disabled. To demonstrate the correctness of the prediction and show where the UAV would have reached without re-planning, avoiding damaging the vehicle, we set its height above the obstacle. As a result, the UAV detected a collision at time $7.39\,\mathrm{s}$ when it was $0.98\,\mathrm{m}$ away from the obstacle and flew through the obstacle because replanning was disabled in this test. In the second set of experiments shown in Fig. 6(c) and 6(d), we enabled the replanning module. While the UAV is flying, at time $7.84\,\mathrm{s}$, the predictions by the meta-learned model indicated a collision risk, which resulted in triggering the replanning behavior. The UAV randomly sampled a waypoint at $(-0.41, 0.8)\mathrm{m}$ and generated a new desired trajectory to drive around the obstacle. Given the new desired trajectory, the UAV confirmed that the new path is safe within the prediction horizon based on the meta-trained model predictions. We note that as Experiment 1 and Experiment 2 are two separate experiment instances, the UAV detects the collision at different times. In Experiment 2, it took less than $0.05\,\mathrm{s}$ to find a safe solution after detecting the collision, which demonstrates that our approach was fast enough to be performed at runtime. It is also noteworthy that how to pick the waypoints for re-planning is not the contribution of this work; we rather focus on predicting the states of the vehicle with an unknown fault.

In the third experiment, the obstacle was set at a different position than the previous two experiments. As shown in Fig. 6(e) and 6(f), the UAV constantly monitored the predictions and did not need to perform re-planning since no potential collision was detected.

We also tested our approach in scenarios with more obstacles. As shown in Fig. 7, we sampled some examples of the predictions while the UAV navigates through the cluttered environment. At times $7.20\,\mathrm{s}$ and $13.51\,\mathrm{s}$ the UAV predicted and detected potential collisions. We also showed two other predictions of future states and uncertainties at times $9.18\,\mathrm{s}$ and $14.73\,\mathrm{s}$ to further demonstrate the output of the proposed framework. Due to the space constraints of our lab, we did not perform tests with more obstacles and failures; however, we believe that the presented results are representative enough to demonstrate the effectiveness of the proposed approach.



(a) UAV path and examples of predictions.



(b) Overlapped screenshots.

Fig. 7. The UAV detects the collisions and avoids the obstacles multiple times. The legend for this figure is the same of Fig. 6(a).

VII. Conclusions and Future Work

In this work, we have presented a framework to predict future reachable states and their uncertainties of a system with an unknown and unforeseen fault that causes degraded behavior like deviations and possible collisions. To this end we have designed a method that leverages meta-learning to train an easily adaptable model at runtime to make predictions about the future states and state uncertainties of the faulty system. At runtime, we use these predictions to detect potential collisions in the environment, and proactively replan the trajectory to avoid the predicted collisions. We validated the applicability of our approach with both simulations and experiments on a quadrotor UAV with an actuator fault for infrastructure inspection case studies.

In our current formulation, we are only considering static faults and static obstacles, but we plan to extend the proposed technique to consider time-varying faults and mobile objects like other actors in the environment. Additionally, we are exploring ways to incorporate different replanning techniques to provide guarantees about finding a safe planning solution within a fixed time duration.

References

[1] E. Yel and N. Bezzo, "A meta-learning-based trajectory tracking framework for uavs under degraded conditions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 6884–6890.

[2] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 45–52.

[3] S. Sun, L. Sijbers, X. Wang, and C. de Visser, "High-speed flight of quadrotor despite loss of single rotor," *IEEE Robot. Automat. Lett.*, vol. 3, no. 4, pp. 3201–3207, Oct. 2018.

[4] S. Sun, X. Wang, Q. Chu, and C. de Visser, "Incremental nonlinear fault-tolerant control of a quadrotor with complete loss of two opposing rotors," *IEEE Trans. Robot.*, vol. 37, no. 1, pp. 116–130, Feb. 2021.

[5] Z. Hou, P. Lu, and Z. Tu, "Nonsingular terminal sliding mode control for a quadrotor uav with a total rotor failure," *Aerosp. Sci. Technol.*, vol. 98, 2020, Art. no. 105716.

[6] D. Tzoumanikas, Q. Yan, and S. Leutenegger, "Nonlinear mpc with motor failure identification and recovery for safe and aggressive multicopter flight," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 8538–8544.

[7] N. Gandhi, D. Saldaña, V. Kumar, and L. T. X. Phan, "Self-reconfiguration in response to faults in modular aerial systems," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2522–2529, Apr. 2020.

[8] Y. Liu, G. Tao, and S. M. Joshi, "Modeling and model reference adaptive control of aircraft with asymmetric damage," *J. Guid., Control, Dyn.*, vol. 33, no. 5, pp. 1500–1517, 2010. [Online]. Available: https://doi.org/10.2514/1.47996

[9] Y. Liu and G. Tao, "Multivariable mrac for aircraft with abrupt damages," in *Proc. IEEE Amer. Control Conf.*, 2008, pp. 2981–2986.

[10] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "A bayesian nonparametric approach to adaptive control using Gaussian processes," in *Proc. IEEE Conf. Decis. Control*, 2013, pp. 874–879.

[11] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. IEEE Conf. Decis. Control*, 2019, pp. 4601–4608.

[12] E. Yel and N. Bezzo, "Gp-based runtime planning, learning, and recovery for safe UAV operations under unforeseen disturbances," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 2173–2180.

[13] K. Saulnier, D. Saldaña, A. Prorok, G. J. Pappas, and V. Kumar, "Resilient flocking for mobile robot teams," *IEEE Robot. Automat. Lett.*, vol. 2, no. 2, pp. 1039–1046, Apr. 2017.

[14] H. Zhang and S. Sundaram, "Robustness of information diffusion algorithms to locally bounded adversaries," in *Proc. IEEE Amer. Control Conf.*, 2012, pp. 5855–5861.

[15] F. Zhang and W. Chen, "Self-healing for mobile robot networks with motion synchronization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 3107–3112.

[16] M. L. Schrum and M. C. Gombolay, "When your robot breaks: Active learning during plant failure," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 438–445, Apr. 2020.

[17] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting UAVs with reinforcement learning-assisted flight control under cyber-physical attacks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 7358–7364.

[18] S. R. Ahmadzadeh, P. Kormushev, and D. G. Caldwell, "Multi-objective reinforcement learning for AUV thruster failure recovery," in *Proc. IEEE Symp. Adaptive Dyn. Program. Reinforcement Learn.*, 2014, pp. 1–8.

[19] T. Lew, A. Sharma, J. Harrison, A. Bylard, and M. Pavone, "Safe active dynamics learning and control: A sequential exploration–exploitation framework," *IEEE Trans. Robot.*, pp. 1–20, 2022.

[20] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[21] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HyztsoC5Y7

[22] I. Ahmed, M. Quinones-Grueiro, and G. Biswas, "Complementary meta-reinforcement learning for fault-adaptive control," in *Proc. Annu. Conf. PHM Soc.*, vol. 12, no. 1, 2020, p. 8.

[23] R. Kaushik, T. Anne, and J.-B. Mouret, "Fast online adaptation in robotics through meta-learning embeddings of simulated priors," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5269–5276.

[24] M. L. Schrum, M. Connolly, E. Cole, M. Ghetiya, R. Gross, and M. C. Gombolay, "Meta-active learning in probabilistically-safe optimization," 2020, *arXiv:2007.03742*.

[25] S.M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," in *Proc. Robot. Sci. Syst.*, Jul. 2021.

[26] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.