Transactions of the SDPS:
Journal of Integrated Design and Process Science
XX (XXXX) XX-XX
DOI 10.3233/JID-210017
http://www.sdpsnet.org

Remote Run-Time Failure Detection and Recovery Control For Quadcopters

Sajad Shahsavari[a,b*], Mohammed Rabah[a], Eero Immonen[a], Mohammad-Hashem Haghbayan[b] and Juha Plosila[b]

[a] *Computational Engineering and Analysis (COMEA), Turku University of Applied Sciences, Turku, Finland*
[b] *Department of Computing, University of Turku, Turku, Finland*

Abstract We propose an adaptive run-time failure recovery control system for quadcopter drones, based on remote real-time processing of measurement data streams. Particularly, the measured RPM values of the quadcopter motors are transmitted to a remote machine which hosts failure detection algorithms and performs recovery procedure. The proposed control system consists of three distinct parts: (1) A set of computationally simple PID controllers locally onboard the drone, (2) a set of computationally more demanding remotely hosted algorithms for real-time drone state detection, and (3) a digital twin co-execution software platform — the *ModelConductor-eXtended* — for two-way signal data exchange between the former two. The local on-board control system is responsible for maneuvering the drone in all conditions: path tracking under normal operation and safe landing in a failure state. The remote control system, on the other hand, is responsible for detecting the state of the drone and communicating the corresponding control commands and controller parameters to the drone in real time. The proposed control system concept is demonstrated via simulations in which a drone is represented by the widely studied *Quad-Sim* six degrees-of-freedom Simulink model. Results show that the trained failure detection binary classifier achieves a high level of performance with F1-score of 96.03%. Additionally, time analysis shows that the proposed remote control system, with average execution time of 0.49 milliseconds and total latency of 6.92 milliseconds in two-way data communication link, meets the real-time constraints of the problem. The potential practical applications for the presented approach are in drone operation in complex environments such as factories (indoor) or forests (outdoor).

Keywords: Digital twin, predictive fault management, failure recovery, quadcopter

1. Introduction

Over the past few decades, Unmanned Aerial Vehicles (UAVs), so-called *drones*, have been a subject of considerable interest for both academia and a spectrum of industries expanding way beyond the first military applications (Springer, 2013). Today, through advances in battery technology, wireless communication systems and automatic control, drones are able to deliver payloads over extended distances (Torabbeigi,

*Corresponding author. Email: sajad.shahsavari@turkuamk.fi.

Lim, & Kim, 2020), carry out autonomous surveillance (Hasan, Newaz, & Ahsan, 2018) and complex inspection tasks in both outdoor (e.g. Seo, Duque, and Wacker (2018)) and indoor (e.g. Shahmoradi, Talebi, Roghanchi, and Hassanalian (2020)) environments. As a practical example, in 2017, the European Maritime Safety Agency (EMSA) issued a contract to Martek, valued at 67M€, for the usage of drones in European waters to provide assistance with border control activities, pollution monitoring, search and rescue tasks, and detection of illegal activities (Howard, 2017). While at present drones are mostly used for *monitoring tasks*, in the future, they are also expected to increasingly participate in complex *actuation work* such as industrial maintenance and building construction (e.g. Da Silva and Eloy (2018); Goessens, Mueller, and Latteur (2018); Kim and Oh (2020)). This necessitates highly accurate and robust control of drone systems in complex environments and assignments.

Among the many drone design alternatives, the *quadcopter* model is widely used in practice. A quadcopter drone utilizes four motors directly connected to four fixed-pitch propellers in a symmetrical X-shaped plane arrangement, making it both low-cost and easy to control. However, being absent of redundant propulsion equipment, quadcopter drones are also prone to damage in collision situations (Alabsi & Fields, 2019). To protect the (potentially expensive) on-board equipment, practical drone control should seamlessly extend to such failure situations.

The questions of interest in this research are: (1) how to recognize the drone operation mode remotely in real-time, (2) how to control of the drone when one of its propellers has failed to provide the required upward thrust, and (3) how to perform adaptive modification to the on-board controllers of the drone to fit the corresponding operation state. We present a proposal solution in order to detect the failure in the drone's motor operation fast enough and act to safely land it. More specifically, our paper proposes an adaptive run-time damage recovery control methodology for quadcopter drones in a simulation-based framework. We consider situations in which the propulsive power of one of the four motors is lost — due to overheating (Dai, Quan, Ren, & Cai, 2018), collision (Rydell, Tulldahl, Bilock, Axelsson, & Köhler, 2020) or a similar incident in practice — and the drone must be gracefully landed by using the remaining three motors only. The upshot in this work is that, to reduce equipment weight and potentially also power consumption, and thus maximize the drone's operating range, a significant part of the overall control system is hosted *outside of the drone*.

The proposed control system consists of three components: (1) Onboard PID controllers: a set of computationally simple PID controllers locally onboard the drone, responsible for maneuvering the drone in all conditions *subject to externally given controller parameters and control objectives*: Path tracking under normal operation and safe landing in a failure state. (2) Remote state detection and failure recovery control system: a set of computationally more demanding remotely hosted algorithms, responsible for detecting the state of the drone (failure or normal) based on real-time RPM measurements, and communicating the corresponding controller parameters and control objectives to the drone's onboard controllers. (3) Digital twin co-execution software platform — *ModelConductor-eXtended (MCX)* (Shahsavari, Immonen, Rabah, Haghbayan, & Plosila, 2021) — for real-time two-way signal data exchange between the former two.

The proposed drone state detection method employs the *ChangeFinder* algorithm (Takeuchi & Yamanishi, 2006) and a logistic regression classifier, trained offline, to rapidly identify the drone state from monitored motor speeds. This task is non-trivial for two reasons. First, the motor speeds vary considerably also during normal operation, due to maneuvering. Second, the failure detection is time-critical and has to be very fast in order to maintain control and avoid crash landing. In this article, the proposed drone control system concept is demonstrated by numerical simulations in which the drone is represented by the widely studied *Quad-Sim* six degrees-of-freedom Simulink model (Hartman, Landis, Mehrer, Moreno, & Kim, 2014). This model is first used offline for designing controllers for both normal and failure mode operation, and then executed online on *MCX* with a failure event time unknown to the remote system.

In summary, the main contributions of this work are as follows:

- We propose a modification of the *Quad-Sim* simulation model to also represent a motor failure situation, and a control system is designed accordingly to recover from such a situation.
- We propose a novel fast fault detection and recovery technique, based on the *ChangeFinder* algorithm and logistic regression, capable of failure detection and re-configuration of the on-board control system at run-time.
- We describe new development of a two-way data stream management facility in the open-source *ModelConductor-eXtended* (MCX) framework in order to enable bidirectional data exchange between the drone and the remote control system.
- We demonstrate the feasibility of the proposed remote run-time failure control for quadcopter drones in a simulation case study.

The remainder of this article is organized as follows. In Section 2, we describe the relation of the present work to drone control, remote run-time control, digital twins and predictive fault management. Then, an overall system architecture is explained in Secion 3. In Section 4, we introduce the *Quad-Sim* quadcopter model and use it for designing controllers for normal (Subsection 4.2.1) and failure (Subsection 4.2.2) situations. The failure detection algorithm in the remote control system is explained in Section 5. Afterward, in Section 6, the *MCX* platform is presented as a solution for hosting the algorithms and arranging data streams in the remote control problem. A simulation case study is reported in Section 7. Finally, in Section 8, the article is concluded and directions for future work are discussed.

2. Related Work

2.1. Drone control

Various types of control systems have been developed for stable trajectory tracking in the normal operation of drones. Different control strategies such as PID, self-tunning fuzzy PID, neural network-based and hybrid controllers have been studied in Emran and Najjaran (2018) for quadrotors with respect to system underactuation, model uncertainty and fault management.

Considering the drone landing, authors in Demirhan and Premachandra (2020) proposed a PID altitude controller to smoothly control the drone toward the detected symbol in the landing spot which its position has been automatically recognized using image processing. The ground effect in landing procedure and utilization of fuzzy logic controllers have been studied in Talha, Asghar, Rohan, Rabah, and Kim (2019).

Recent studies include vast investigation on improving of safety-critical quadcopter systems by developing fault-tolerant controllers. In Sun, Cioffi, De Visser, and Scaramuzza (2021), authors developed an onboard state estimation to continue the path tracking of quadcopter despite the failure of one of its motors. Their proposed state estimation method is based on combining gyroscopic data with range sensory data with event camera information. Major distinction between the present article and aforementioned research is that the state estimation in our work is performed remotely with no need to place additional resources for executing heavy computational tasks.

Adjusting the center of mass of the quadcopter has also been studied as a proposal solution to keep hovering or safely land the quadcopter in case of failure of one motor (Ambroziak et al., 2019; Muhamad, Nasir, Kari, & Ali, 2017). In Ambroziak et al. (2019), authors proposed an onboard failure detection method based on gyroscopic (IMU) measurements in addition to the re-configuration procedure of the PID control system and shifting the plant center of mass, if necessary, in case of partial or complete failure in a single rotor of the quadcopter. However, such methods are able to successfully control the drone in experimental environment, additional mechanical equipment is needed for altering the mass structure of the drone during the flight.

The *Quad-Sim* model that is used in the current study has been validated in our previous articles. In Rabah, Rohan, Talha, Nam, and Kim (2018), a comparative study between different types of controllers has been presented to test the performance of the quadcopter while tracking rectangular and triangular paths. Furthermore, the simulated controllers have been deployed on a physical quadcopter system that used PX4 flight controller interfaced with the open-source ArduPilot framework to follow a moving target object (Rabah, Rohan, Han, & Kim, 2018; Rabah, Rohan, Mohamed, & Kim, 2019). The tests showed that the developed controllers which were originally designed in the simulation model were working fine for the experimental platform as well. Moreover, an autonomous landing algorithm has been developed in Rabah, Rohan, Haghbayan, Plosila, and Kim (2020) to overcome the ground effect. All simulation studies have been carried out in the *Quad-Sim* model to test the developed algorithms. Results showed that the developed algorithms were able to autonomously land the quadcopter smoothly.

In the current article, widely studied PID controllers have been used to control the drone in the normal operation state to follow the trajectory and land it in the failure state, by only modifying their gain parameters. The failure recovery strategy consists of remote re-configuration of the onboard PID controllers and also gradually decreasing the drone altitude to ensure a safe landing.

2.2. Remote run-time control, data synchronization and digital twins

For remote control, one should arrange a robust two-way data exchange to and from the system to be controlled. In real-time applications, especially those employing model-based control (as in digital twins Colombo, Karnouskos, Kaynak, Shi, and Yin (2017), defined here as accurate numerical simulation models operating alongside their corresponding physical systems) and safety-critical elements, data synchronization between the two communication ends should also be addressed. Specifically, this requires *co-execution* of the remote control system with the target to be controlled. These problems are not new, but, perhaps surprisingly, they have been mostly considered *separately* from each other in previous research.

As for the data communication problem, there are open-source and commercial software platforms to provide data exchange facilities in a cyber-physical system such as Eclipse (2020) and Microsoft (2020). As for digital twins, we refer the reader to the recent review article Cimino, Negri, and Fumagalli (2019).

To *simultaneously* address co-execution of physical and digital assets and manage asynchronous data streams, Aho and Immonen (2020) and subsequently Shahsavari et al. (2021) have developed an open-source platform, *ModelConductor-eXtended (MCX)*, which is capable of establishing a data exchange channel between physical device and simulation-based digital twin[1]. The MQTT architecture and experimental examples have been presented Shahsavari et al. (2021), and in the present article, we describe two-way data communication with real-time control functionality in *MCX*.

Data-driven digital twins have previously been used for optimizing control in process systems (see e.g. He, Chen, Dong, Sun, and Shen (2019) and the references therein). In the present article, we carry out remote run-time control on a simple data-based digital twin representing inference of the prevailing state of the drone deduced from incoming quadcopter motor speed data streams.

2.3. Predictive fault management

The quadcopter drone failure recovery control research reported in this article addresses aspects of *predictive fault management*, in which the goal is to monitor the state of a physical device for abnormalities and take necessary measures to avoid or minimize damage to the equipment. Such measures can be fully automatic (as is the case in the present work) or, simply a notification to maintenance workers to take action Tajima et al. (2008). In this context, abnormality detection is based on some representation, or model, of system operation which measurement data from the physical system is compared to (Werner, Zimmermann,

[1] https://github.com/COMEA-TUAS/mcx-public

& Lentes, 2019). The system model can be based on fundamental physics (e.g. Aivaliotis, Georgoulias, Arkouli, and Makris (2019); Almasi (2016); Alvarez, Gutierrezzea, Bilb, Napolitano, and Fravolini (2019); de Azevedo, Araújo, and Bouchonneau (2016); Rehman et al. (2013)), data (e.g. Furukawa and Deng (2020); Susto, Schirru, Pampuri, McLoone, and Beghi (2014)), state machines (Hooman & Hendriks, 2007) or combinations thereof.

Physics-based models typically rely on computational fluid dynamics (CFD), finite-element method (FEM) or similar, which are computationally so expensive that they cannot be used in real-time applications such as the quadcopter motor failure recovery control considered in the present article. On the other hand, state machine models are capable of real-time performance, but require clearly defined states which are not *a priori* available here, because the quadcopter motor speeds also vary in normal flight due to changing drone attitude in trajectory tracking. Consequently, in the present work, we utilize data-based predictions: The well-known data stream change point detection algorithm *ChangeFinder* (Takeuchi & Yamanishi, 2006) is combined with a simple binary logistic regression classifier, which is trained to be able to deduce the drone state in previously unseen operating patterns. This approach is similar to that of Furukawa and Deng (2020) who used machine learning to detect faults in a tank system. However, they reported fault detection times reaching over 10 seconds, which is not feasible for the present drone application. The proposed fault identification method, on the other hand, is capable of fault detection in the order of tens of milliseconds.

Additionally, neural network-based classifiers such as Long Short Term Memory (LSTM) sequence classifier models, one dimensional Convolutional Neural Network (CNN) models or a combination of them have also been used for fault detection (Fu, Sun, Yu, & Liu, 2019; Sadhu, Zonouz, & Pompili, 2020). The main disadvantage of using such deep neural network models is their time and computational complexity, and this conflicts with the inherent real-time requirement of recovering a faulty flying quadcopter. If the models are to be executed on-board the drone, they will also require some embedded AI computing resource (such as Jetson TX2) to be installed on the drone. Here, we aim for computational simplicity in the learning model by using *ChangeFinder* in which internal parameters are adjusted on-line, leading to a fast state classification with no need to heavy tensor or matrix operations. Also, this on-line adjusting of parameters in *ChangeFinder* will help the model to adapt reasonably to the patterns in new environments. While for transferring neural network-based models to new environments, usually some re-training to new data set is needed.

3. Overall System Architecture

The overall system architecture of the proposed method is shown in Figure 1. The proposed recovery technique consists of two parts: (1) the local *Quad-Sim* model (described in detail in Section 4) and (2) the remote control system (described in detail in Section 5), both executed in real-time on the *MCX* platform (described in detail in Section 6). The first part is responsible for rapid measurement of motor RPMs and applying the transferred signals to the on-board PID controllers. Moreover, this part is responsible for sending the measurement to the stationary remote processing unit and receiving the appropriate control parameters from it. The second part is the remote control system that is responsible for analysing the received streaming data of RPM measurements, through a measurement queue, and processing this in the failure detection unit. Based on the analyzed data, if a fault is detected, the remote control system informs the drone about the fault by sending the predicted action mode signal as well as new controller parameters and a new path reference target point. Using the observation and action queues facilitates asynchronous data streams in two-way communication.

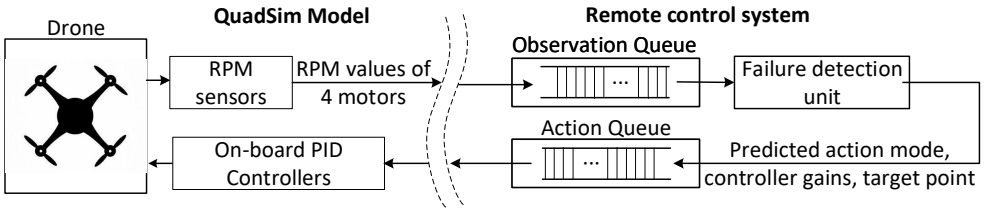

Fig. 1. Overview of the System Containing Drone's Simulation and Run-time Failure Detection Models Linked and Executed Together Using *MCX* Framework

4. Quadcopter Simulation Model and the On-Board Control System

A quadcopter is a type of Unmanned Aerial Vehicles (UAVs), which is lifted and propelled by four motors providing the upward thrust in addition to the direction control of the Quadcopter. The motors located diagonally opposite to each other rotate in the same direction while the adjacent motors rotate in the opposite direction.

Generally, a quadcopter has six degrees of freedom (6-DOF) meaning that six variables are needed to express its position and orientation in space: $(x, y, z, roll, pitch, yaw)$. The quadcopter is an Underactuated Mechanical System (UMS) with four actuator inputs to control six degrees of freedom. This underactuation and its nonlinear dynamical model add extra challenges to the robust failure-tolerant controller design.

In this work, the open-source MATLAB/SIMULINK-based quadcopter simulation model *Quad-Sim* is used to study the behavior of the quadcopter while tracking a trajectory. The original simulation model for normal behavior is available on MATLAB file exchange (Hartman et al., 2014). The model has been extended for simulating the operation of quadcopter in the failure mode of operation as well, with one (or more) of its motors failing to provide expected upward thrust.

4.1. Simulation model description

Figure 2 shows the *Quad-Sim* model composition. The **Path command** block is responsible for generating a time series database that determines a path. The **Position controller** block calculates the desired angles (roll, pitch) and altitude that are needed for following the path by using a PD controller. The desired quadcopter angles and altitude are used as a reference to the **Attitude/Altitude controller** block which controls the state of the quadcopter (roll, pitch, yaw, and altitude) using PID controllers. The main function of this block is to minimize the error between the desired angles/altitude and their current values (angle/altitude corrections) to maintain the stability of the flying quadcopter. The **Quadcopter control mixing** block takes angle/altitude corrections and computes the corresponding throttle percentage for each motor. The **Disturbances** block is designed to mimic the effect of external forces and disturbances which affect the forces applied to the quadcopter in x and y directions (F_x, F_y). Finally, the **Quadcopter dynamics** block outputs the suitable motors' RPMs that are required based on the state space equations describing the quadcopter dynamics (Luukkonen, 2011).

4.2. Quadcopter controller design

Equation 1 presents the general formula of the PID controller, where the error $e(t)$ is the difference between desired and current angles/altitude values, $y(t)$ is the output of the controller (here angles and altitude correction) and K_p, K_i and K_d are the proportional, integral, and derivative gains respectively. The PD controller is a type of PID controller that uses the same formula as in Equation 1 with $K_i = 0$.

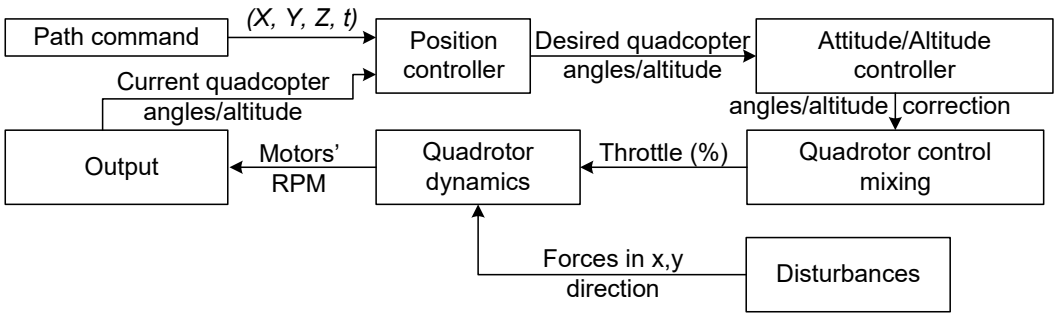

Fig. 2. Quadcopter Simulation Blocks.

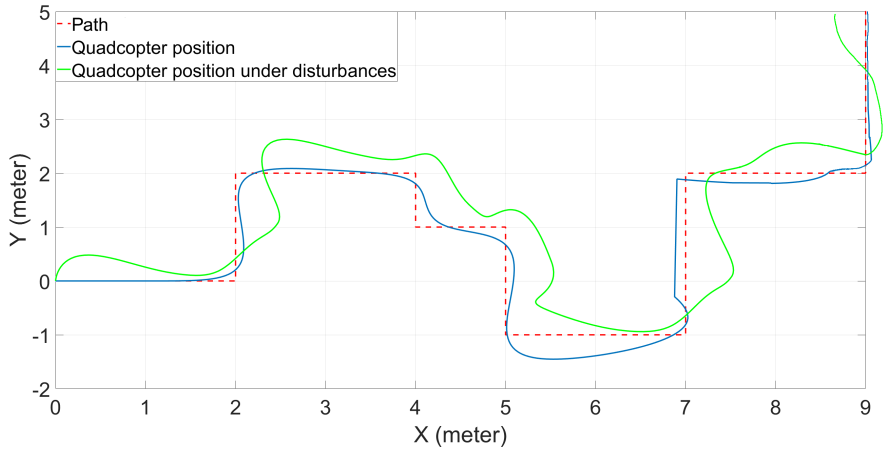

Fig. 3. Simulation Results of the Proposed Controller in 2D Space.

$$y(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

There are four PID controllers in the **Attitude/Altitude controller** and two PD controllers in the **Position controller** block. For controller design, i.e. specifying the gain parameters of all the controllers, it is assumed that the quadcopter is following the path shown in Figures 3. During the tracking process, an external force of a sawtooth wave, with an amplitude of 3 N and frequency of 0.5 Hz affects the x-direction, while a sin wave with an amplitude of 2 N and frequency of 0.5 Hz affects the y-direction. The purpose of adding these forces, is to test the performance of the PID controllers under disturbances. Obtained motor RPMs are shown in Figure 5. In this simulated data, two different modes have been considered; normal and failure mode, where each mode has distinct PID and PD gains.

4.2.1. Normal mode

During the first 21 seconds of the path tracking shown in Figures 3 and 4, the quadcopter is in normal mode. The PID and PD gains used in this mode are the default gains of the *Quad-Sim* model shown in Table 1 and Table 2. Despite the applied F_x and F_y disturbances, the quadcopter was able to maintain a stable RPM

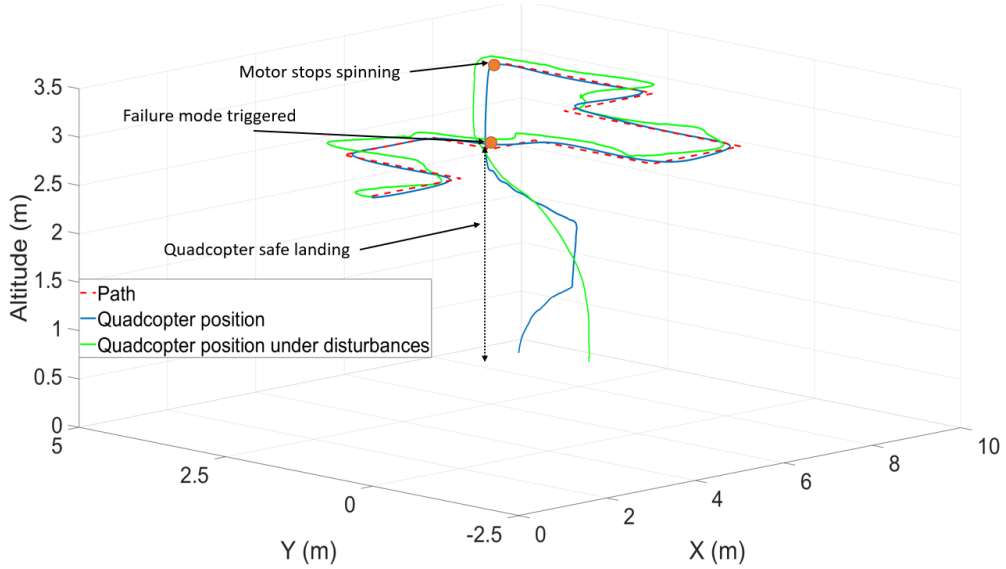

Fig. 4. Simulation Results of the Proposed Controller in 3D Space.

among its motors (as shown in the Figure 5), and thus smooth and stable flight over the path in the normal mode.

Table 1. *Quad-Sim* Attitude/Altitude Controller Default Gains

Controller	K_p	K_i	K_d
Roll	2	1.1	1.2
Pitch	2	1.1	1.2
Yaw	4	0.5	3.5
Altitude	2	1.1	3.3

Table 2. *Quad-Sim* Position Controller Default Gains

Controller	K_p	K_d
Roll	0.32	0.1
Pitch	0.32	0.1

4.2.2. Failure mode

The failure mode occurs at $t = 21$ seconds as one motor fails and its RPM drops to zero. This total failure of the motor might be due to collision, electrical overload, overheating, or power supply issues. This will lead to disruption in RPM value of the other three motors and cause imbalance orientation of the drone, and will eventually result to the crash of the plant. By changing the parameter gains of the PID and PD controllers the plant could be smoothly landed to avoid any additional loss.

As demonstrated in Figure 4, once the quadcopter reaches the final point $(9,5)$ at $t = 21$ seconds, one of the motors will stop spinning causing the Quadcopter to fall down from 3 meters to 2 meters. Once the

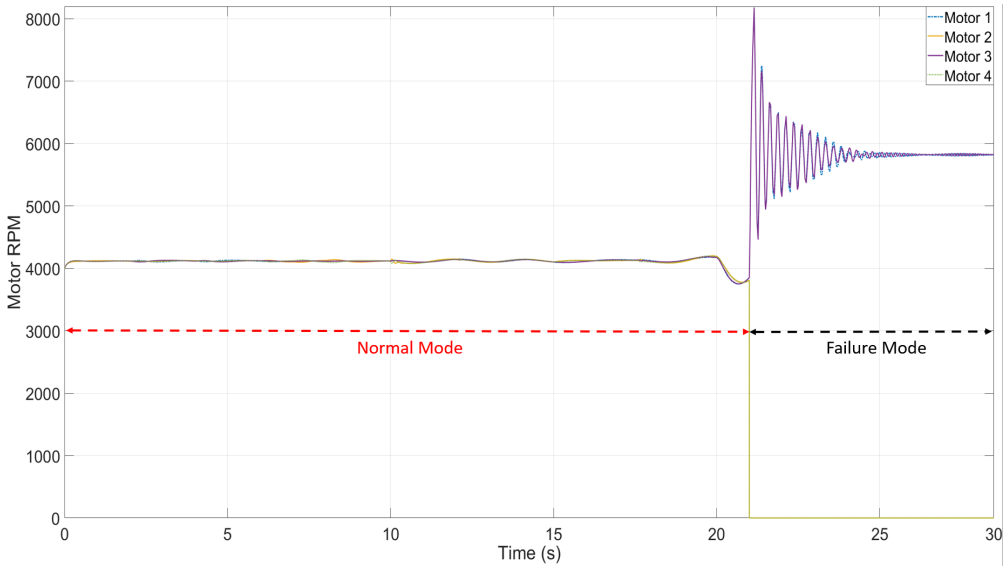

Fig. 5. Simulated Motors RPM During Tracking the Path.

failure is detected in the failed motor, e.g. motor 4, failure mode will be in effect and this trigger forces the diagonally opposite motor to stop spinning in order to keep the quadcopter balanced (as shown in Figure 5). Afterwards, the PID and PD gains in the **Attitude/Altitude controller** and **Position controller** blocks are re-configured to the values shown in Table 3 and Table 4.

For failure mode situation, higher PID gains have been chosen in order to allow the remaining two motors to take more and rapid electric power, and therefore achieve higher RPMs to compensate the loss of other two motors. In this state, the quadcopter has only one degree of freedom (altitude) instead of the six, which can be resolved by forcing the remaining two motors produce equal upward thrust (Mueller & D'Andrea, 2014). Operation of only two out of four propellers will also cause the quadcopter to inevitably spin around its vertical axis. Therefore, the PID gains for Yaw controller have been set to zero. Furthermore, decoupling of altitude from translational and attitude control makes the two-propeller architecture more robust and stable (Rible, Arriola, & Ramos, 2020). Moreover, it is not possible for the quadcopter to reverse the opposite motor direction during the flight as it would also reverse the produced thrust, and destabilize the quadcopter. For this reason, we aim to gradually decrease the altitude of the quadcopter over the final position by using two motors, and ensure a safe landing while spinning around the vertical axis.

Table 3. Failure Mode PID Gains Used in the Attitude/Altitude controllers

Controller	K_p	K_i	K_d
Roll	4.5	2.7	0.53
Pitch	7.8	2.7	0.53
Yaw	0	0	0
Altitude	700	10	5

4.3. Communication to remote control system

During the quadcopter operation, the motor RPM values predicted by the *Quad-Sim* simulation model are continuously streamed out to the remote control system. The remote control system monitors these

Table 4. Failure Mode PD Gains Used in the Position Controller

Controller	K_p	K_d
Roll	0	0
Pitch	0	0

data streams and attempts to detect any abnormality. For each detected mode (normal or failure) mode, the corresponding quadcopter PID gains and target path are communicated in real-time to the local quadcopter controller. The PID gains are those obtained from off-line controller design for the two operating modes. The path target in normal mode is the next quadcopter target location along a pre-defined trajectory sequence, whereas in a failure mode it is a pre-defined safe landing sequence. Note that the remote control system does not recognize the type of failure (partial or total) and considers any abnormal behavior as total failure and starts the landing procedure immediately.

5. Remote Run-Time Failure Detection and Recovery Control System

The remote control system, in this work, consists of (1) a statistical model trained to rapidly detect abnormal values in the quadcopter motor RPM data streams and (2) a set of action modes and gain parameter values for normal and failure modes. Here, an abnormal value, or *fault*, is defined as *any* unusual value in the four motor RPM values, that has not been detected by the system in normal quadcopter flight patterns during model training. We do not distinguish between different types of failure modes: The problem of fault detection is a binary classification assignment (normal/failure) for four RPM values at any given operation time.

When a quadcopter is subjected to a failure (loss of power) in one motor, it is unbalanced and tilted, such that it quickly becomes uncontrollable. This process significantly limits the time between measuring the motor RPMs and (potentially) starting the recovery actions. Due to this limitation, the fault detection/recovery time should not violate a certain time limit in the worst case to ensure the applicability of the recovery. The time limit for failure detection is not fixed and depends on the quadcopter state (orientation and angular velocities) at the failure time instant. The source of this time limitation is the limited pre-defined allowable range of operation in quadcopter roll and pitch angles.

5.1. Quadcopter fault detection model

The fault/normal classification model is an adapted version of the *ChangeFinder* change point detection method (Takeuchi & Yamanishi, 2006), combined with a binary logistic regression classifier. The *ChangeFinder* is a two-stage time-series learning approach, which is summarized for convenience below.

Let us define x_t^i to represent the RPM value for the quadcopter motor i (with $i = 1,\ldots,4$) at time instant t. In the first stage, the *ChangeFinder* calculates the preliminary *logarithmic* score:

$$s_i(t) = -\log p_{t-1}(x_t^i|\mathbf{x}_{t-1}^i) \tag{2}$$

where $\mathbf{x}_{t-1}^i = x_1^i x_2^i x_3^i \cdots x_{t-1}^i$ and $p_{t-1}(x_t^i|\mathbf{x}_{t-1}^i)$ is the conditional probability density function (sub-scripted with $t-1$ since it is trained incrementally with all previous samples) specifying the probability of observing sample x_t^i at time t, given the previous samples. In *ChangeFinder*, this conditional probability density function is assumed to be Gaussian and its parameters are dynamically estimated by using a k-order Auto Regression (AR) model and maximizing an exponentially decayed weighted likelihood function using a Sequential Discounting AR (SDAR) algorithm (Takeuchi & Yamanishi, 2006). In the second stage, a moving

average on the last T values of preliminary scores is calculated to produce another time series y_t^i. The SDAR algorithm is then invoked again to construct yet another set of parametric conditional density functions $q_{t-1}(y_t^i|\mathbf{y}_{t-1}^i)$ over new time series data y_t^i. Another intermediate score for time t is computed using the same *logarithmic* scoring function. Based on that, the final possibility for x_t^i being a significant change in the initial time series is defined as an average of the last T computed logarithmic scores as:

$$Score(x_t^i) = \frac{1}{T}\sum_{k=t-T+1}^{t}(-\log q_{k-1}(y_k^i|\mathbf{y}_{k-1}^i)) \tag{3}$$

In the quadcopter fault detection model, one *ChangeFinder* model is instantiated for each four RPM data stream, and, therefore, there are four score values to potentially indicate a change of quadcopter state at any given time. In order to decide on final failure/normal classification, these four scores are fed into a logistic regression model, with weight parameters optimized on the training data set. More formally,

$$p(\text{fault occurs at } t) = \frac{1}{1+e^{-z(t)}}, \quad \text{with } z(t) = \sum_{i=1}^{4} w_i Score(x_t^i) \tag{4}$$

where the weights w_i were optimized during training.

5.2. Hyper-parameter optimization

For the *ChangeFinder* algorithm, values for the following three hyper-parameters that control the learning process must be chosen before applying the model:

(1). Order of the autoregressive model (k);

(2). Size of the moving average window (T);

(3). Decay value for the exponentially weighing likelihood function (r) for SDAR.

To optimize the values for these hyper-parameters, we utilized the Latin Hypercube Sampling method (McKay, Beckman, & Conover, 2000). This method is used for generating near-random three-dimensional sample vectors for hyper-parameters that fill the pre-specified parameter search space. Then, for each sample vector containing model hyper-parameters (k,T,r), the *ChangeFinder* and LR models were optimized on the training data and the corresponding fault detection model performance was evaluated over the previously unseen drone flight validation data set. The chosen fault detection model parameters were those that displayed the best performance on validation data.

5.3. Training data and failure mode detection accuracy

We used the *Quad-Sim* simulation model explained in Section 4.1 (adapted to be capable of introducing failure in run-time) to generate labeled time series data for normal and failure operation modes of the drone. In order to cover possible RPM variation in the drone operation (e.g. normal high-speed maneuvers), we have created 500 distinct 40-second-long random paths with 5 different randomly distributed target points in each path. Since the transition between normal and failure modes must also appear in the training data, during each path, at some random time between 20 and 30 seconds, a fault is injected to one of the motors by imposing the RPM reduction by the randomly chosen factor between 0% to 90%. In the pre-processing stage, in each flight time series data, the leading part from 0 to 5 seconds has been excluded to mitigate the effect of model initialization. Similarly, any data following the quadcopter hitting the ground is excluded from the training data. In total, the training data set consists of 12529.13 and 1082.44 seconds of normal and failure operational modes, respectively. The data set has imbalanced label distribution as the majority of the

samples belong to the normal flight operation mode. Optimizing the parameters of the LR model following the *ChangeFinder* score values (described in Subsection 5.1) is used as a solution to the imbalanced classification problem (and can be considered as equivalent to finding the optimal threshold for binary imbalanced classification).

The simulated data set has been divided into training and test sets. 20% of the data samples have been left out from training and are used for extracting the classification evaluation metrics: precision, recall and F1 score. The results are presented in Table 5.

Table 5. Evaluation of Failure Detection Classification Model over Test Data Set

Evaluation metric	result %
Precision	93.31
Recall	98.92
F1 Score	96.03

It is also noteworthy here that the results shown in Table 5 are for simulated data samples which might not represent all patterns in the real-world behavior of the quadcoper exactly, but still the proposed *technique* is applicable to the real environment. The trained model with simulated data can also be used as initial state for new model which will be fine-tuned with considerably smaller amount of real-world time-series data. The simulations also facilitate rigorous analysis of acceptable communication delays in failure mode recovery. Moreover, providing real-world experimental data for such failure situations are challenging due to the risk of equipment damage and left for future work.

5.4. Recovery control

The response of remote control system consists of: (1) operation mode in action which could be either normal or failure, (2) PID controller gain parameters to configure the on-board positional, attitude and altitude PID controllers, and (3) position of the next target point to follow.

6. *MCX* - An Open-Source Digital Twin Co-Execution Platform

To co-execute the *Quad-Sim* quadcopter model (Section 4) and the remote run-time control and failure detection method (Section 5) simultaneously, we propose to use the *MCX* framework. The object-oriented *MCX* framework facilitates executing the simulation models (digital twins) in real-time along with physical devices (Aho & Immonen, 2020; Shahsavari et al., 2021). In the present work, the remote failure detection method embeds a (statistical and data-based) digital twin for the state of the drone, whereas the physical drone is represented by the *Quad-Sim* quadcopter model. *MCX* arranges all data exchange between the entities by establishing an asynchronous data communication channel between them, as explained below.

6.1. Operation principle

The overview of the framework is shown in Figure 6, with publisher assets in one side and subscribers in the other. This structure is based on the Message Queuing Telemetry Transport (MQTT) network protocol which was initially designed for Internet of Things (IoT) applications. The MQTT broker (queue) at the middle will handle the message asynchronous delivery mechanism with dynamic buffering. In this architecture, each publisher asset will specify a *tag* for every outgoing message indicating its content category. Each subscriber asset, on the other hand, will register to the queue for specific tags in advance and then are provided with the messages published by those tags.

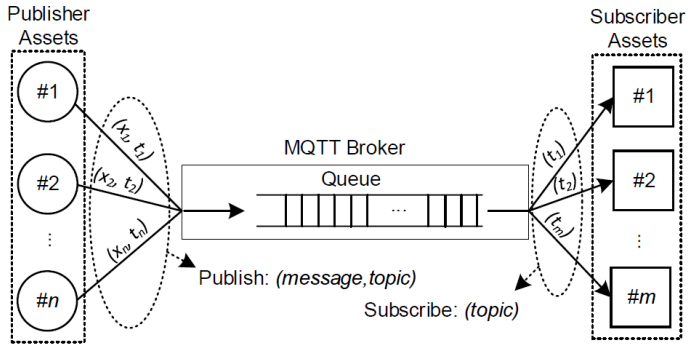

Fig. 6. Publisher/Subscriber Structure in *MCX* (Shahsavari et al., 2021)

Note that a single asset (client of the queue) could be both publisher and subscriber simultaneously to enable two-way data communication between different clients through the queue. Once a single registered subscriber receives and processes a message, it can publish the response to the queue using the same registered client with a new tag.

Nevertheless, it should be noted that *MCX* itself does not include any simulation model, measurement or sensory data, but, on the other hand, it includes (1) the placeholder for running the digital twin i.e. different types of simulation models on-line, and (2) an asynchronous message communication mechanism which basically consists of a queue data structure and publishing/subscribing utilities. To create a digital twin, the simulation/prediction model should be embedded into the framework using one of these three options:

(1). Functional Mock-up Unit (FMU): This simulation model unit is basically a single package which comprise communication and interaction utilities for model exchange/co-execution, standardized as Functional Mock-up Interfaces (Blochwitz et al., 2011). Many widely used design and simulation software products (e.g. MATLAB/SIMULINK, GT-SUITE, ANSYS etc.) support exporting the simulation model into FMU package.

(2). Scikit-learn machine learning model: if the digital twin is composed of a machine learning prediction model which is designed and trained in Python Scikit-learn library its exported package could be placed into *MCX* to act as a digital twin.

(3). Custom Python function: Due to object-oriented design of the framework, the corresponding method of the simulation model producing the response object (forward *step* method) could be overwritten in a custom Python class.

In the present study, both options (1) and (3) have been utilized, FMU co-execution utility for embedding the *Quad-Sim* quadcopter model and custom step method in Python for injecting the fault in run-time and perform recovery actions issued from remote control system.

7. Case Study

For an experimental simulation case study, we implemented the remote control system, failure occurrence and recovery strategy on the *MCX* platform. We used the *Quad-Sim* model, augmented with the failure representation functionality (Section 4), wrapped into an FMU package. This FMU model accepts inputs (namely the controller parameters and target path points) from the remote control system *during execution of the simulation model*. The FMU model also outputs the four predicted RPM values of the quadcopter motors, *during run-time*, to be communicated to the remote controller.

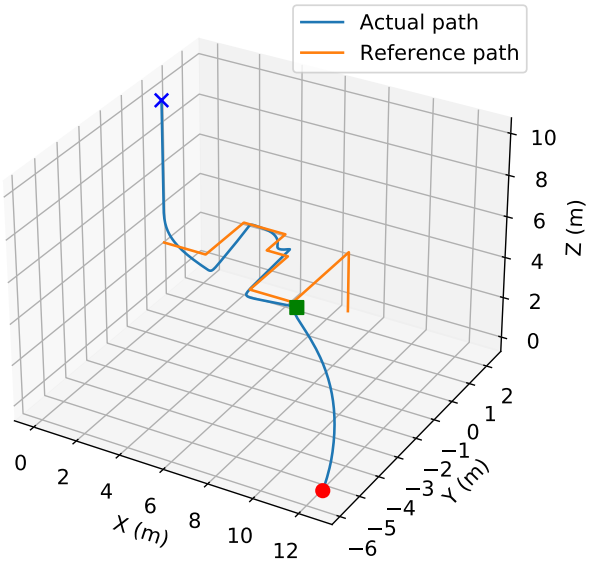

Fig. 7. The Reference Path of the Quadcopter (orange), the Actual Flight Path (blue), the Initial Location (Blue Cross), the Failure Point (Green Square) and the Final Landing Position (Red Circle).

The system overview is shown in Figure 1. In the bidirectional data exchange, arranged by *MCX*, the drone FMU simulation model publishes live motor speed values, and, at the same time, is registrant as a subscriber to receive remote control system response. On the other hand, the remote controller system (executing the failure detection model) is registered to the queue to receive the motor speed values, and also publishes the predicted response regarding the drone operation.

It is important to emphasize that both normal and failure operation modes take place in the *same* FMU packaged simulation model without restart or manual user intervention. Moreover, although the quadcopter always starts in the normal mode, the time instant of motor failure is random and unknown to the remote control system. Assuming that a failure occurs at $t_{failure}$, it is detected with a delay at $t_{detection}$ where $t_{detection} > t_{failure}$. The delay includes network transmission delay and fault detection model execution time. Once the failure is detected at $t_{detection}$ the *failure* action mode along with new controller parameters are transmitted to the drone. This issues a shut off command to the motor diagonally opposing the failed motor, and re-configuration of the on-board PID controllers to those designed off-line (Subsection 4.2.2). Moreover, the path reference is changed to a linear reduction of flight altitude (z) to ground level.

7.1. Results

The simulation results of the experiment are shown in Figures 7, 8, 9 and 10, all for the same flight path. In this experiment, the drone is following the reference path shown in the orange curve of Figure 7 in normal operation. Then, unknown to the remote controller, at $t_{failure} = 15.00\ s$ the failure occurs. It is subsequently detected by the remote control system at $t_{detection} = 15.026\ s$. This 26 ms of delay includes network transmission delay and execution time of the model.

In Figure 8, the reference signal and drone positional values are plotted. Green horizontal line is drawn on $t_{detection}$ and red dashed horizontal line shows the landing time. For $t < 15\ s$, the drone follows the

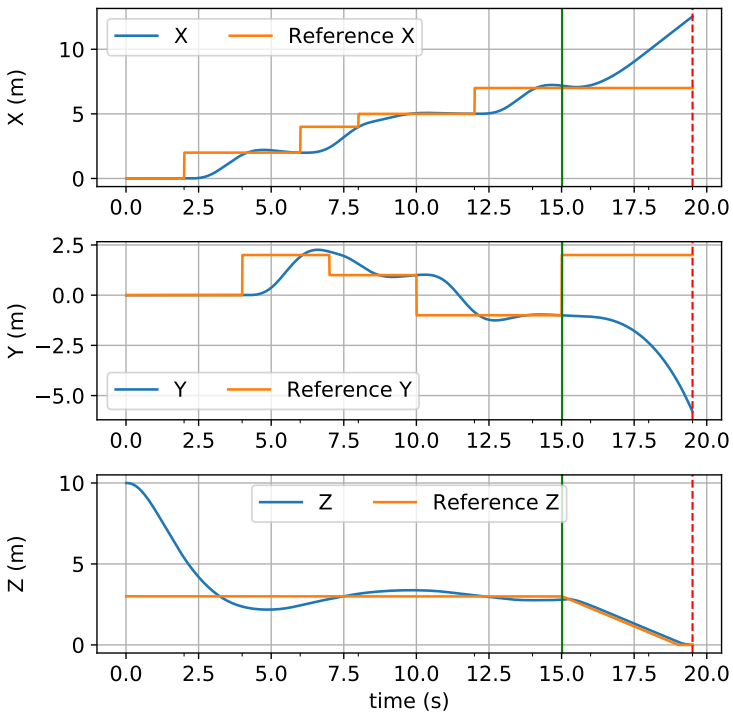

Fig. 8. The (x,y,z) Position of the Quadcopter During the Flight in Blue and Reference Signal for Each in Orange. Green and Red Horizontal Lines are Failure Detection and Landing Time, Respectively

reference steadily but during the recovery process altitude of the drone will be decreased under control until it reaches the ground. The positional values (x,y) are not constant during the landing procedure because of oscillation of the drone alignment due to shutdown of two motors. Nonetheless, the primary interest here is controlled reduction of the altitude (z).

Figure 9 shows the values of four motor RPMs. After the detection of failure in motor 1, motors 1 and 3 have been shut off. Therefore, the on-board controllers increase the power of other two motors (2,4) to compensate the loss of upward force due to the shutting motor 1 and 3 off. The motor RPMs recorded after the failure event, during the landing sequence, represent decaying oscillations as typically seen in PID control applications.

Finally, in Figure 10, the drone attitude (orientation) is demonstrated during the flight in terms of angles: ϕ (Phi) for roll, θ (Theta) for pitch and ψ (Psi) for yaw. Evidently, the drone starts to spin around its vertical axis when working only with two diagonal motors with the same spinning direction (similar to Mueller and D'Andrea (2014); Muhamad et al. (2017)). To the authors' knowledge, no research has been reported on successfully controlling *traditional* quadcopters (such as here) rotation with two propellers only, although a solution has been reported for *tilting-rotor* quadcopters (Nemati, Kumar, & Kumar, 2016).

7.2. Performance evaluation

The timing analysis is presented in Table 6 including the mean and standard deviation of measured delay value (or processing time) of each component in the system over 2500 samples of the particular flight in this experiment. There are four different sources of delay in the system operation: (1) Processing time of the FMU simulation model that represents the drone behaviour: this delay should not be the matter of concern because in real environment it does not appear, (2) more importantly, the fault detection model execution time, i.e. duration that the model needs to predict normal or failure operation, (3) network transmission

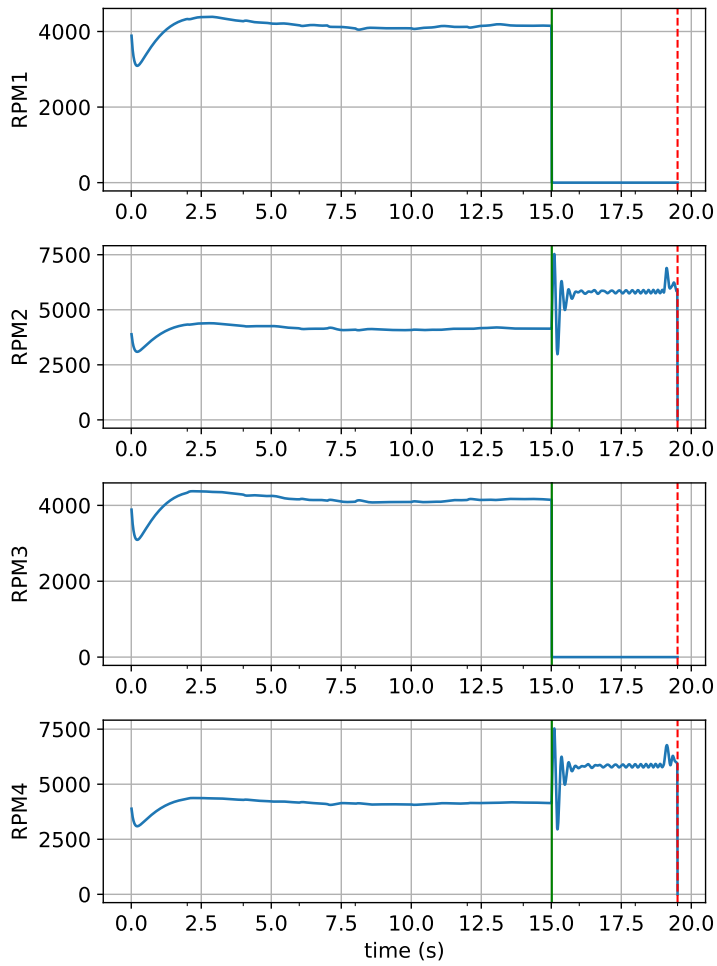

Fig. 9. Motor RPM Values for Four Motors of the Drone, Before and After the Failure.

delay from failure mode detection model to the FMU *Quad-Sim*, and (4) the reverse delay of that. Based on the statistics it is clear that the failure detection model is sufficiently fast and could be employed in real-time operation.

8. Conclusions and Directions for Future Work

In this paper a fault detection and recovery control system for quadcopter drones is proposed. The failure detection is proposed to be performed remotely according to analysis of the real-time simulated measurements. As the failure situation is predicted, the remote control system issues run-time re-configuration to the on-board controllers to overcome the occurred fault. Experimental results based on the *MCX* co-execution platform, show the efficiency and applicability of the proposed technique in detecting the failures and safely landing the drone after failure detection.

The most important direction for future work is using the real platform in validation of the proposed remote control framework by physical experiments. With suitable on-board hardware and wireless connectivity on a real quadcopter drone, the *MCX* platform can be utilized for replicating the case study presented in Section 7. The digital-first approach proposed in the present article can then be used for designing remote drone control systems for more complex assignments than the simple trajectory tracking one considered

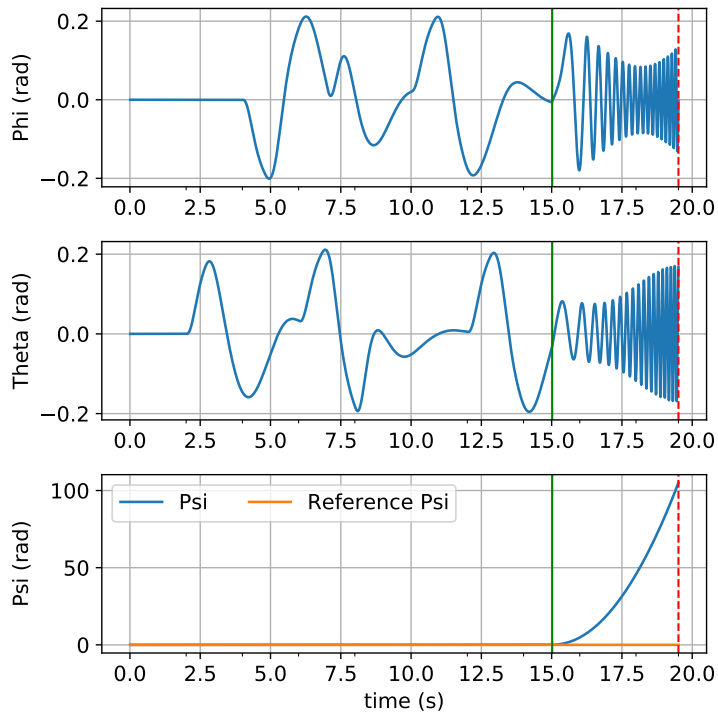

Fig. 10. Drone Attitude in Terms of Phi for Roll, Theta for Pitch and Psi for Yaw During the Flight.

Table 6. Time Delay Analysis of the Experiment (in Milliseconds)

Delay type	Mean	Standard deviation
FMU Quad-Sim execution time	39.81	8.12
Fault detection model execution time	0.49	1.41
Forward network delay	1.69	2.30
Backward network delay	4.74	4.18

herein. Another interesting future direction would be to investigate different operation modes and fault models, and recovery algorithm based on each of them.

Acknowledgments

This work has been financially supported by the Academy of Finland funded projects 335512 - ADAFI (Adaptive-Fidelity Digital Twins for Robust and Intelligent Control Systems) and 330493 - AURORA (Autonomous Performance Management in Digital Manufacturing), and by Nokia Jorma Ollila Grant. We gratefully thank anonymous reviewers at Society for Design and Process Science (SDPS) for commenting on earlier versions of this paper that undoubtedly improved its quality, and for pointing good future directions.

References

Aho, P., & Immonen, E. (2020). Modelconductor: An on-line data management architecture for digital twins. In *2020 25th ieee international conference on emerging technologies and factory automation*

(etfa) (Vol. 1, pp. 1397–1400).

Aivaliotis, P., Georgoulias, K., Arkouli, Z., & Makris, S. (2019). Methodology for enabling digital twin using advanced physics-based modelling in predictive maintenance. *Procedia Cirp*, *81*, 417–422.

Alabsi, M. I., & Fields, T. D. (2019). Real-time closed-loop system identification of a quadcopter. *Journal of Aircraft*, *56*(1), 324–335.

Almasi, A. (2016). Latest lessons learned, modern condition monitoring and advanced predictive maintenance for gas turbines. *Australian Journal of Mechanical Engineering*, *14*(3), 199–211.

Alvarez, O. H., Gutierrezzea, L., Bilb, C., Napolitano, M., & Fravolini, M. (2019). Digital twin concept for aircraft sensor failure. In *Transdisciplinary engineering for complex socio-technical systems: Proceedings of the 26th iste international conference on transdisciplinary engineering, july 30–august 1, 2019* (Vol. 10, p. 370).

Ambroziak, L., Simha, A., Pawluszewicz, E., Kotta, Ü., Bożko, A., & Kondratiuk, M. (2019). Motor failure tolerant control system with self diagnostics for unmanned multirotors. In *2019 24th international conference on methods and models in automation and robotics (mmar)* (pp. 422–427).

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., ... others (2011). The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th international modelica conference* (pp. 105–114).

Cimino, C., Negri, E., & Fumagalli, L. (2019). Review of digital twin applications in manufacturing. *Computers in Industry*, *113*, 103130.

Colombo, A. W., Karnouskos, S., Kaynak, O., Shi, Y., & Yin, S. (2017). Industrial cyberphysical systems: A backbone of the fourth industrial revolution. *IEEE Industrial Electronics Magazine*, *11*(1), 6–16.

Dai, X., Quan, Q., Ren, J., & Cai, K.-Y. (2018). Efficiency optimization and component selection for propulsion systems of electric multicopters. *IEEE Transactions on Industrial Electronics*, *66*(10), 7800–7809.

Da Silva, N., & Eloy, S. (2018). Will drones have a role in building construction? In *4th international symposium formal methods in architecture.*

de Azevedo, H. D. M., Araújo, A. M., & Bouchonneau, N. (2016). A review of wind turbine bearing condition monitoring: State of the art and challenges. *Renewable and Sustainable Energy Reviews*, *56*, 368–379.

Demirhan, M., & Premachandra, C. (2020). Development of an automated camera-based drone landing system. *IEEE Access*, *8*, 202111–202121.

Eclipse, F. (2020). *Ditto: where iot devices and their digital twins get together.* Retrieved 2020-11-01, from https://www.eclipse.org/ditto/

Emran, B. J., & Najjaran, H. (2018). A review of quadrotor: An underactuated mechanical system. *Annual Reviews in Control*, *46*, 165–180.

Fu, J., Sun, C., Yu, Z., & Liu, L. (2019). A hybrid cnn-lstm model based actuator fault diagnosis for six-rotor uavs. In *2019 chinese control and decision conference (ccdc)* (pp. 410–414).

Furukawa, Y., & Deng, M. (2020). Fault detection of tank-system using changefinder and svm. In *2020 international conference on advanced mechatronic systems (icamechs)* (pp. 260–265).

Goessens, S., Mueller, C., & Latteur, P. (2018). Feasibility study for drone-based masonry construction of real-scale structures. *Automation in Construction*, *94*, 458–480.

Hartman, D., Landis, K., Mehrer, M., Moreno, S., & Kim, J. (2014). *Quad-sim.* https://github.com/dch33/Quad-Sim. GitHub.

Hasan, K. M., Newaz, S. S., & Ahsan, M. S. (2018). Design and development of an aircraft type portable drone for surveillance and disaster management. *International Journal of Intelligent Unmanned Systems*.

He, R., Chen, G., Dong, C., Sun, S., & Shen, X. (2019). Data-driven digital twin technology for optimized control in process systems. *ISA transactions*, *95*, 221–234.

Hooman, J., & Hendriks, T. (2007). Model-based run-time error detection. In *International conference on model driven engineering languages and systems* (pp. 225–236).

Howard, M. (2017, Mar). Martek named on eu maritime drone contract. *MarineLink*. Retrieved from https://www.marinelink.com/news/maritime-contract-martek423262

Kim, D., & Oh, P. Y. (2020). Human-drone interaction for aerially manipulated drilling using haptic feedback. In *2020 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 9774–9780).

Luukkonen, T. (2011). Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, *22*, 22.

McKay, M. D., Beckman, R. J., & Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *42*(1), 55–61.

Microsoft. (2020). *Azure digital twins: Next-generation iot solutions that model the real world.* Retrieved 2020-11-01, from https://azure.microsoft.com/en-us/services/digital-twins/

Mueller, M. W., & 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)* (pp. 45–52).

Muhamad, S., Nasir, H. M., Kari, S., & Ali, M. S. M. (2017). Hovering recovery strategies for single rotor inoperative quadcopter. *International Journal of Applied Engineering Research*, *12*(24), 14585–14588.

Nemati, A., Kumar, R., & Kumar, M. (2016). Stabilizing and control of tilting-rotor quadcopter in case of a propeller failure. In *Dynamic systems and control conference* (Vol. 50695, p. V001T05A005).

Rabah, M., Rohan, A., Haghbayan, M.-H., Plosila, J., & Kim, S.-H. (2020). Heterogeneous parallelization for object detection and tracking in uavs. *IEEE Access*, *8*, 42784–42793.

Rabah, M., Rohan, A., Han, Y.-J., & Kim, S.-H. (2018). Design of fuzzy-pid controller for quadcopter trajectory-tracking. *International Journal of Fuzzy Logic and Intelligent Systems*, *18*(3), 204–213.

Rabah, M., Rohan, A., Mohamed, S. A., & Kim, S.-H. (2019). Autonomous moving target-tracking for a uav quadcopter based on fuzzy-pi. *IEEE Access*, *7*, 38407–38419.

Rabah, M., Rohan, A., Talha, M., Nam, K.-H., & Kim, S. H. (2018). Autonomous vision-based target detection and safe landing for uav. *International Journal of Control, Automation and Systems*, *16*(6), 3013–3025.

Rehman, A. U., Shinde, S., Singh, V. K., Paul, A. R., Jain, A., & Mishra, R. (2013). Cfd based condition monitoring of centrifugal pump. In *Proceedings of comadem.*

Rible, G. P. S., Arriola, N. A. A., & Ramos, M. C. (2020). Fail-safe controller architectures for quadcopter with motor failures. In *2020 6th international conference on control, automation and robotics (iccar)* (pp. 384–391).

Rydell, J., Tulldahl, M., Bilock, E., Axelsson, L., & Köhler, P. (2020). Autonomous uav-based forest mapping below the canopy. In *2020 ieee/ion position, location and navigation symposium (plans)* (pp. 112–117).

Sadhu, V., Zonouz, S., & Pompili, D. (2020). On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification. In *2020 ieee international conference on robotics and automation (icra)* (pp. 5255–5261).

Seo, J., Duque, L., & Wacker, J. (2018). Drone-enabled bridge inspection methodology and application. *Automation in Construction*, *94*, 112–126.

Shahmoradi, J., Talebi, E., Roghanchi, P., & Hassanalian, M. (2020). A comprehensive review of applications of drone technology in the mining industry. *Drones*, *4*(3), 34.

Shahsavari, S., Immonen, E., Rabah, M., Haghbayan, M. H., & Plosila, J. (2021). Mcx — an open-source framework for digital twins. In *Ecms* (pp. 119–124).

Springer, P. J. (2013). *Military robots and drones: a reference handbook*. ABC-CLIO.

Sun, S., Cioffi, G., De Visser, C., & Scaramuzza, D. (2021). Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters*, *6*(2), 580–587.

Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., & Beghi, A. (2014). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, *11*(3), 812–820.

Tajima, T., Borden, M., Olivas, F., Chamberlin, J., Harrison, J., Oothoudt, M., & Canabal, A. (2008). Lansce vacuum system refurbishment plan and vacuum alert system improvements for predictive maintenance. In *Proc. epac* (pp. 3717–3719).

Takeuchi, J.-i., & Yamanishi, K. (2006). A unifying framework for detecting outliers and change points from time series. *IEEE transactions on Knowledge and Data Engineering*, *18*(4), 482–492.

Talha, M., Asghar, F., Rohan, A., Rabah, M., & Kim, S. H. (2019). Fuzzy logic-based robust and autonomous safe landing for uav quadcopter. *Arabian Journal for Science and Engineering*, *44*(3), 2627–2639.

Torabbeigi, M., Lim, G. J., & Kim, S. J. (2020). Drone delivery scheduling optimization considering payload-induced battery consumption rates. *Journal of Intelligent & Robotic Systems*, *97*(3), 471–487.

Werner, A., Zimmermann, N., & Lentes, J. (2019). Approach for a holistic predictive maintenance strategy by incorporating a digital twin. *Procedia Manufacturing*, *39*, 1743–1751.

Author Biographies

Sajad Shahsavari works as a researcher at Computational Engineering and Analysis (COMEA) research group in Turku University of Applied Sciences, and is a Ph.D. student at University of Turku, Finland. His research interests include deep neural networks, time-series prediction, reinforcement learning and data analysis. He received his B.Sc. degree in Computer Engineering from Amirkabir University of Technology, Tehran, Iran in 2014 and his M.Sc. degree in Artificial Intelligence from Sharif University of Technology, Tehran, Iran in 2017.

Mohammed Rabah (Ph.D.) received his B.Sc. degree in Electronics and Communication Engineering from the AL-SAFWA High Institute of Engineering, Egypt in 2015. He completed his M.S. in Electronics and Information Engineering from Kunsan National University, South Korea in December 2017, and the Ph.D. degree from Kunsan National University, South Korea in August 2020. Currently, he is working as a research engineer in Turku University of Applied Sciences, Turku, Finland. His research interests includes automation, control and intelligent systems. Furthermore, he is also interested in UAV's applications, fuzzy systems, and deep learning.

Eero Immonenworks as Principal lecturer and leader of the Computational Engineering and Analysis (COMEA) research group at Turku University of Applied Sciences, Finland. Dr. Immonen has published several contributions to control theory of dynamical systems, in particular Internal Model Control for infinite-dimensional systems, for which work he received the Tampere Science Fund award for the best doctoral thesis in 2006. His research work on robust control has also addressed topics such as practical regulation, hybrid controller structures, and, more recently, agent-based models in finance and staff scheduling. During his over 8 years' work as a modeling specialist and R&D manager at Process Flow Ltd, Dr. Immonen was involved in over 100 commercial digital twin simulation model development projects for industrial customers worldwide. Some of the project results involving fluid-structure interaction models, flow control and CFD-based shape optimization have also been published in the academic literature.

Mohammad-Hashem Haghbayan received the B.Sc. degree in computer engineering from Ferdowsi University of Mashhad, the MS degree in computer architecture from University of Tehran, Iran, and Ph.D. with honour from University of Turku, Finland. Since 2018 he is a postdoc researcher at UTU Department of Computing. His research interests include machine learning, autonomous systems, high-performance energyefficient architectures, and on-chip/fog resource management. He has 63 peer-reviewed publications in international conferences and journals.

Juha Plosila (M'06) is a Professor in Autonomous Systems and Robotics at the University of Turku (UTU), Department of Computing, Finland. He received a PhD degree in Electronics and Communication Technology from UTU in 1999. He is the head of the EIT Digital Master Programme in Embedded Systems at the EIT Digital Master School (European Institute of Innovation and Technology) and represents UTU in the Node Strategy Committee of the EIT Digital Helsinki/Finland node. He has a strong research background in adaptive multi-processing systems and platforms and their design. This includes, e.g., specification, development and verification of self-aware multi-agent monitoring and control architectures for massively parallel systems, machine learning and evolutionary computing-based approaches, as well as application of heterogeneous energy efficient architectures to new computational challenges in the cyber-physical systems and internet-of-things domains, with a recent focus on fog/edge computing (edge intelligence) and autonomous multi-drone systems.