**Noname manuscript No.**
(will be inserted by the editor)

# Diagnosis and Compensation of Control Program, Sensor and Actuator Failures in Nonlinear Systems Using Hierarchical State Space Checks

**Md Imran Momtaz · Abhijit Chatterjee**

Received: date / Accepted: date

**Abstract** Autonomous systems with nonlinear dynamics need to be extremely resilient to errors in sensors, actuators and on-board electronics for the purpose of overall vehicle safety. Prior research has focused on control-theoretic methods with significant computational burden with a focus on failures in actuation. In contrast, we propose the use of hierarchical machine learning driven state space checks that detect and diagnose errors in control program execution, sensors and actuators with high sensitivity and low latency. Each check produces a time-varying error signal that facilitates effect-cause diagnosis of the system, while allowing rapid parameter estimation from each check. Since the checks are over small subsets of system parameters, estimation is fast and accurate. The estimated parameters are then used to reconfigure the system controller parameters for rapid system recovery. We use a quadcopter system to demonstrate and validate our approach. Controller, sensor and actuator errors can be detected, diagnosed and compensated using a common checking platform with low computational overhead. The technique is validated on a quadcopter hardware test vehicle.

**Keywords** Real-time systems, Autonomous systems, Resilience, Sensor fault, Actuator fault, Quadcopter system, State space check, On-line test, Sequential model, Time-series analysis

M. I. Momtaz
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta GA 30332
E-mail: momtaz@gatech.edu

A. Chatterjee
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta GA 30332
E-mail: abhijit.chatterjee@ece.gatech.edu

## 1 Introduction

With increasing dependence on autonomous machines that can *sense their environment and govern their own actions*, it is becoming imperative that they be completely safe, secure and resilient. This research focuses on the problem of mitigating resilience threats to autonomous quadcopters from failures in sensors, actuators and soft errors in on-board processors running control program. The scope of the problem is well illustrated with data for self-driving cars that is more readily available. In current world, safety standard of the complex autonomous systems is described in ISO 26262 [1], where operation quality of different subsystems are pointed out with their minimum safety standard. Autonomous vehicle disengagement data filed with the California Dept. of Motor Vehicles [2] for 2016 shows that a self-driving car *failed about every 3 hours due to hardware or software malfunction*. Other examples abound [3–5]. The most recent Boeing incident [3, 4], was diagnosed to a malfunctioning sensor generating incorrect measurement data.

In this research, we focus on low overhead diagnosis and compensation of sensor and actuator malfunction in quadcopters. Both transient and parametric failure effects are addressed. We also consider detection and correction of malfunction in control program running on an on-board digital processor. The key idea is to exploit a hierarchy of checks for rapid parametric diagnosis and control adaptation of the quadcopter to enable the system *to sustain its performance for the maximum possible length of time without human intervention.*

In the following, we first discuss prior research in autonomous system resilience and present the key contributions of this research in relation to the state of the art. Next, we discuss the basics of quadcopter operation and control, present our proposed resilience approach and end with a discussion of experimental results and conclusions.

## 2 Prior Work

There has been significant work in the past on failure tolerance in autonomous systems: sensors, actuators and control. This can be classified into two broad research themes: *anomaly detection* and *control adaptation*. An anomaly is defined to be an operating condition different from normal that the system is not designed to handle. After an anomaly is detected, system control is reconfigured in such a way as to compensate for the effect of the anomaly on overall system function.

With regard to *anomaly detection*, there has been work on statistical estimation algorithms for the detection of outliers (anomalies) [6, 7]. These methods are generally compute-intensive and not suitable for applications with hard real-time constraints. The work of [8] develops a methodology for detecting anomalies in high dimensional data while a robot is operating in the field. Positive and negative data models are created and separated using a support vector machine. There is a significant body of work revolving around prediction of the future observable states of a system from prior states and comparison with achieved future state (measured) values for anomaly detection [9–13]. In [10], particle filtering and maximum likelihood methods are used to diagnose and correct sensor anomalies in autonomous ground vehicles. In [11], a sliding window observer is designed to predict future sensor measurements for error detection. In [12, 13], a Kalman filter is designed to perform accurate statistical state estimation in the presence of single inertial sensor faults and thereby enable sensor fault tolerant control of unmanned aerial vehicles. Recently there has been work on sensor data fusion to identify sensor as well as actuator malfunction in robotic systems [14].

Of particular relevance to this research is prior work on the use of neuromorphic networks for anomaly detection and correction. In [15], a suite of neural networks are trained in real time to predict aircraft sensor measurements from values of prior sensor measurements and control inputs. Actuator faults are determined by specific measurement cross-correlation tests. Actuator correction is performed by forcing the neural network to stabilize the aircraft through PID control applied to the non-faulty aircraft actuators. A similar state estimation based failure detection strategy for generic nonlinear systems using a bank of neural networks is developed in [16]. State estimation methods are also used for error detection in [17, 18]. In [18], a neural network is used to learn the normal future and past state and input dependencies. On-line gradient descent on the plant model parameter values is used to minimize the prediction error between the observed and predicted future states for parameter diagnosis. In [19–22], past observed sensor measurements and inputs are used to predict a linear encoding of all the system states using static machine learning techniques which lacks adaptability. In [23], past observed sensor measurements and inputs are used to predict a linear encoding of all the system states using a nonlinear regression mapping. This is shown to detect sensor, actuator failures as well as errors in execution of the control program on a digital processor. A hierarchical error detection and error localization scheme is presented in [21]. However, this does not address error correction and control reconfiguration.

With regard to *control adaptation* there has been significant research in the past [24–26]. In *gain scheduling* [24], relevant gain parameters of the system control algorithm are adapted to meet dynamic performance requirements. For example, the speed of an aircraft and its height (measured by speed and height sensors) can be used to dynamically change the aircraft controller parameters. *Model reference adaptive control* (MRAC) assumes the use of a reference model of the system (plant) continuously running on a processor in the background against which the system behavior is compared in real-time to generate an error signal. The so-called MIT rule [24] relies on the derivative of the controller parameters to this error to tune the controller to minimize this error. There are *indirect* methods for control adaptation as well. In indirect MRAC and self-tuning regulators [24, 26], first plant parameter estimation is performed using observed sensor measurements. The resulting estimated parameters are then mapped to optimize control law parameters using a mapping function or look-up table. In [27], controller parameters were redesigned using absolute value of the encoded state of the system which was improved at [19] by utilizing the time dependent profile of the encoded state. In [28], the value function of a reinforcement learning algorithm is initialized to specific profiles corresponding to clusters of plant parameter values estimated by a probabilistic neural network from sensor measurements. In this case, not all the plant parameter values can be estimated with high accuracy and the selection of the profile concerned significantly speeds up the reinforcement learning process. Recently L1 adaptive controllers [25] have been proposed in which the problem of state estimation (adaptation) is decoupled from that of control. This allows very fast adaptation to changing plant dynamics and actuation failures while allowing robust control under parameters variations and noise using conventional control theoretic techniques. L1 adaptive controllers, however, need the use of state prediction algorithms that are typically derived from linearized models of nonlinear systems. A recent work [29] has investigated at this problem with the help of 'time-series based prediction' where the measurements of the system are predicted using time-series models validated by simulation. The initial research [29] provided initial proof-of-concept ideas to motivate the present research. The present research is significantly detailed compared to [29]. Literature review in the field of failure detection, diagnosis, and correction for autonomous systems is significantly enhanced in this manuscript. We have performed thorough study on neuromorphic model complexity vs performance tradeoff and in this new work, we have used a new neuromorphic model (based on GRU, Gated Recurrent Unit) which is aware about the available hardware resources and timing constraints of the nonlinear system. Significant work was done to build a hardware prototype for demonstration of the key concepts of the paper. [29] did not have any hardware data and only simulation results were presented. This itself was a major effort in this work. We present actual hardware data to support the feasibility of the proposed methodology on a real-time system. Additionally, we have focused on the notion of 'hardware reuse' to implement the proposed failure management infrastructure in hardware. The same hardware that is used to fly the quadcopter is also used for management of failures. In this way, the most efficient use of the resource is ensured. Hence, the entire scheme is implemented using standard hardware and digital processing available for conventional quadcopter flight showing scalability of the approach.

## 3 Key Contributions

We propose to use a *hierarchy of checks* with long prediction horizons using long short term memory (LSTM, machine learning) networks. Each check produces a *time-varying error signature e(t)* which is ideally zero in the absence of failure (except for noise and machine learning inaccuracies). The employed checks *detect sensor and actuator errors as well as errors in control program execution on a digital processor.* The core contribution of this research is in rapid error correction through control reconfiguration. The set of checks employed allow quick localization of detected malfunction in actuators, sensors and control program execution. The key contributions of this research are as follows:
*Control Program Errors:* There has been limited attention to recovery from errors in control program execution on a digital processor [30, 31]. These involve baseline recovery methods or use of computational redundancy. We propose the design of special machine learning assisted checks for control program error detection and use of the same checking mechanism for *actuator value restoration* for error recovery. The method handles both data and control flow errors and performs both error detection and correction with low overhead and low latency as compared to existing techniques.
*Sensor Malfunction:* The proposed machine learning assisted checks are used to detect and localize sensor errors. Both transient errors and sensor value offsets are addressed by estimating the correct sensor values from the implemented checks. Correction is performed over a limited future time horizon via sensor value restoration and involves replacing affected sensor values with their estimated duplicates. This allows additional time for returning a nonlinear system to a "safe" state (for example, returning to ground for a quadcopter) as opposed to its normal operating plan.
*Actuator Malfunction:* It is seen that the error signals *e(t)* taken over the hierarchical set of checks employed bear strong correlation with the actuator parameters that are perturbed under failure. The set of checks is leveraged to perform rapid actuator parameter estimation and the relevant actuator (quadcopter motor) controller parameters are adjusted to restore overall system performance using closed form equations. Alternatively this adjustment to the controller parameters can be *predicted directly* from the observed transient check error signals $e(t)$ using supervised machine learning based models. Note that no reference model of the quadcopter continuously running in the background is needed.

## 4 Preliminaries: Quadcopter and Brushless DC Motor Models and Control

### 4.1 State Variable System: Overview

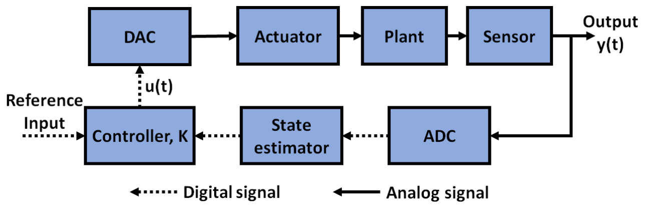


Fig. 1: A nonlinear state variable system

Figure 1 shows the block diagram of a state variable system which can be of linear or nonlinear type. For a general state variable system, the plant behavior is expressed in the form of an ordinary differential equation.

$$\dot{s}(t) = f(s(t), u(t)) + w(t) \tag{1}$$

where, $s(t)$ is the states of the system, $u(t)$ is the plant inputs and $y(t)$ is the outputs of the system. The function $f(.)$ represents the relationship between $s(t)$, $u(t)$ and the derivative of the vector $s(t)$ and $w(t)$ represents zero-mean process noise. The output equation of the plant is given by,

$$z(t) = h(s(t), u(t)) + v(t) \tag{2}$$

where $h(.)$ represents the relationship between $s(t)$, $u(t)$ and the system output $z(t)$ and $v(t)$ represents zero-mean measurement noise. For both linear and nonlinear state variable systems, the input $u(t)$ is computed from the system output $z(t)$ and the reference signal $r(t)$ by an external controller $K$ that strives to maintain system performance under dynamically changing plant conditions. The control actions performed by the controller can be based on closed form equations (for example, PID controller [32], Lyapunov based controller design [33] or on a reinforcement learning (RL) based controller [34]). The controller analyzes the system outputs, the reference input and determines the best input which drives the plant to its desired performance goals. We use a quadcopter test vehicle and control system to demonstrate the proposed state space check based error detection and compensation approach. This is discussed next followed by a discussion of the error detection and compensation methodology.

## 4.2 Quadcopter Overview

As an example test case of a nonlinear state variable system, quadcopter control is considered in this work. The quadcopter has 12 state variables which are given as: $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$. Here, $x, y$ and $z$ represent the position of the quadcopter in 3 dimensional inertial reference frame and $\phi, \theta$ and $\psi$ represent the roll, pitch and yaw angle in the body frame of the quadcopter (see Figure 2). The quadcopter has an inertial measurement unit (IMU) as the prime sensor which comprises of an accelerometer and a gyroscope, and has 4 brushless DC motors which are used as actuators. The expressions of linear acceleration and angular acceleration for a quadcopter system are given below [35]:

$$\begin{aligned}[\ddot{x}, \ddot{y}, \ddot{z}]^T &= [0, 0, -mg]^T + RT_B + F_D \\ [\ddot{\phi}, \ddot{\theta}, \ddot{\psi}]^T &= I^{-1}(\tau - \omega \times (I\omega))\end{aligned} \tag{3}$$

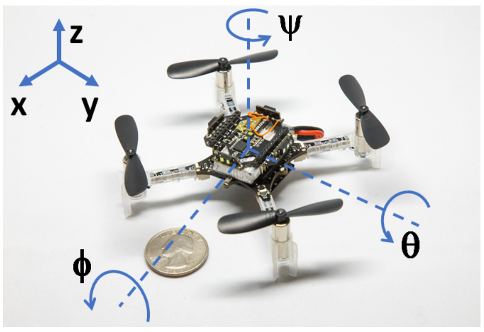


Fig. 2: An Example Quadcopter System (adopted from [36])

where, $m$ = mass of the quadcopter, $g$ = acceleration due to gravity, $R$ = rotational matrix ($R \in$ vector space $\mathbb{R}^{3\times3}$), $T_B$ = thrust vector, $F_D$ = drag force, $I$ = inertia matrix, $\tau$ = external torque vector, $\omega$ = angular velocity vector. The expressions for the aforementioned quantities are given as below:

$$\begin{aligned}
\omega &= \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta S_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \\
F_D &= -k_D \times [\dot{x}, \dot{y}, \dot{z}]^T \\
R &= \begin{bmatrix} (C_\phi C_\psi - C_\theta S_\phi S_\psi) & (-S_\phi C_\psi - C_\theta C_\phi S_\psi) & (S_\theta S_\psi) \\ (C_\phi C_\psi + C_\theta S_\phi C_\psi) & (-S_\phi S_\psi + C_\theta C_\phi C_\psi) & (-C_\psi S_\theta) \\ (S_\phi S_\theta) & (C_\phi S_\theta) & (C_\theta) \end{bmatrix} \\
T_B &= \left[ 0, 0, k \sum_{i=1}^{i=4} \Omega_i^2 \right]^T \\
I_{XX} = I_{YY} &= 2m(\frac{r^2}{5} + L^2) \\
I_{ZZ} &= 2m(\frac{r^2}{5} + 2L^2) \\
I &= diag([I_{XX}, I_{YY}, I_{ZZ}]) \\
\tau &= \begin{bmatrix} Lk(\Omega_1^2 - \Omega_3^2) \\ Lk(\Omega_2^2 - \Omega_4^2) \\ b(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{bmatrix}
\end{aligned}$$

In these expressions, $C_*$ and $S_*$ represent $\cos(*)$ and $\sin(*)$ respectively. As an example, $C_\theta S_\phi S_\psi$ represents $\cos\theta \sin\phi \sin\psi$. Additionally, $\Omega_i$ is the rotor speed of the $i^{th}$ motor, $r$ is radius of quadcopter body as point mass, $L$ is distance of one actuator from center of gravity, $b$ is drag co-efficient and $k$ and $k_D$ are constants of proportionality, and $k$ depends on propeller type, number of blades in each propeller, air density etc. Applying appropriate thrust from each motor changes the dynamics of the system, and hence control the quadcopter in the desired way.

### 4.3 Actuator Overview: Brushless DC (BLDC) Motor

For actuation, the use of brushless DC (BLDC) motors has been investigated in this work. To operate a motor, a three phase AC source is generated from battery (a DC source) with the help of 'power electronic' circuitry. The AC source is then used to actuate the actuator. The simplified state space representation of a BLDC motor can be expressed as $\dot{x}_{act}(t) = A_{act}x_{act}(t) + B_{act}u_{act}(t)$ where $A_{act}$ and $B_{act}$ are state dependent and they are defined as follows [37]:

$$A_{act} = \begin{bmatrix} -R_s/L_1 & 0 & 0 & \hat{f}_a(\lambda\theta_{act})/J & 0 \\ 0 & -R_s/L_1 & 0 & \hat{f}_b(\lambda\theta_{act})/J & 0 \\ 0 & 0 & -R_s/L_1 & \hat{f}_c(\lambda\theta_{act})/J & 0 \\ \hat{f}_a(\lambda\theta_{act})/J & \hat{f}_b(\lambda\theta_{act})/J & \hat{f}_c(\lambda\theta_{act})/J & -B_f/J & 0 \\ 0 & 0 & 0 & P/2 & 0 \end{bmatrix}$$

$$B_{act} = \begin{bmatrix} 1/L_1 & 0 & 0 & 0 \\ 0 & 1/L_1 & 0 & 0 \\ 0 & 0 & 1/L_1 & 0 \\ 0 & 0 & 0 & -1/J \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where, $x_{act}(t) = [I_{a,act}, I_{b,act}, I_{c,act}, \omega_{act}, \theta_{act}]^T$ and

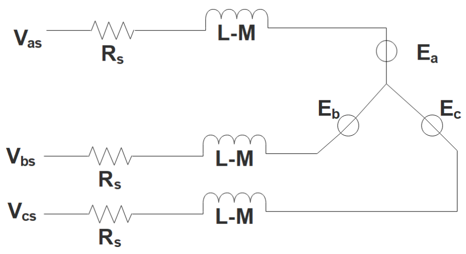


Fig. 3: Brushless DC motor equivalent circuit

$u_{act}(t) = [V_{as}, V_{bs}, V_{cs}, T_l]^T$ are the motor state and input respectively. $R_s, \lambda, J, B_f$ and $P$ are the stator resistance per phase, back emf constant, moment of inertia of the rotor, friction coefficient, number of magnetic pole pairs respectively. $L_1$ is defined as $L_{act} - M_{act}$ where $L_{act}$ and $M_{act}$ are self inductance and mutual inductance per phase respectively. $I_{*,act}, \omega_{act}, \theta_{act}, V_*$ and $T_l$ are the stator phase currents, motor speed, motor electrical angle, applied armature voltage and load torque respectively. $\hat{f}_*(.)$ (i.e. $\hat{f}_a(.), \hat{f}_b(.)$ and $\hat{f}_b(.)$) are trapezoidal functions for modeling generated back emf which are nonlinear in nature [37] and are related by:

$\hat{f}_b(\theta) = \hat{f}_a(\theta - 120^\circ), \hat{f}_c(\theta) = \hat{f}_a(\theta + 120^\circ)$

From the above relationships, we can see that, the quadcopter system have six degrees of freedom and four actuators, making this an underactuated system. Additionally, the system dynamics of a quadcopter and its actuators can be represented in a hierarchical manner: a) for the entire quadcopter, b) for the controller and c) for individual motor.

### 4.4 Controller Design for Quadcopter

A PID controller has been employed for altitude and attitude control of the quadcopter. The controller was designed using the 'Successive Loop Closure' method [38]. For the quadcopter system, the system dynamics is defined in subsection 4.2. We have the following as dynamics for roll, pitch and yaw angles [39]:

$$\ddot{\phi} = \frac{I_{YY} - I_{ZZ}}{I_{XX}}\theta\psi - \frac{J_{TP}}{I_{XX}}\theta\omega + \frac{U_2}{I_{XX}}$$
$$\ddot{\theta} = \frac{I_{ZZ} - I_{XX}}{I_{YY}}\phi\psi - \frac{J_{TP}}{I_{YY}}\phi\omega + \frac{U_3}{I_{YY}}$$
$$\ddot{\psi} = \frac{I_{XX} - I_{YY}}{I_{ZZ}}\phi\theta + \frac{U_4}{I_{ZZ}}$$

where, $[U_2, U_3, U_4]^T$ are individual components of the external torque vector $\tau$ and $J_{TP}$ is angular momentum. All other quantities are explained in subsection 4.2. In the nominal case, the values of $\phi, \theta, \psi$ and $\omega$ all will be small. Consequently the above equations can be approximated as:

$$\ddot{\phi} \approx \frac{U_2}{I_{XX}}, \ddot{\theta} \approx \frac{U_3}{I_{YY}}, \ddot{\psi} \approx \frac{U_4}{I_{ZZ}}$$

From these simplified equations, a PID controller can be designed which controls the roll, pitch, and yaw angles of the quadcopter. For altitude control of the quadcopter, we consider the following:

$$\ddot{z} = -g + \cos\phi\cos\theta\frac{U_1}{m} \tag{4}$$

where, $U_1 = k(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$ and other terms are described in subsection 4.2. Similar to the design of previous PID controller, the values of $\phi$ and $\theta$ will be very small under normal operating condition. For this reason, it is assumed that $\cos\phi \approx 1, \cos\theta \approx 1$. This again results in a simplified equation for which a PID controller was designed to control the altitude. Further details about the controller design can be found in [39].

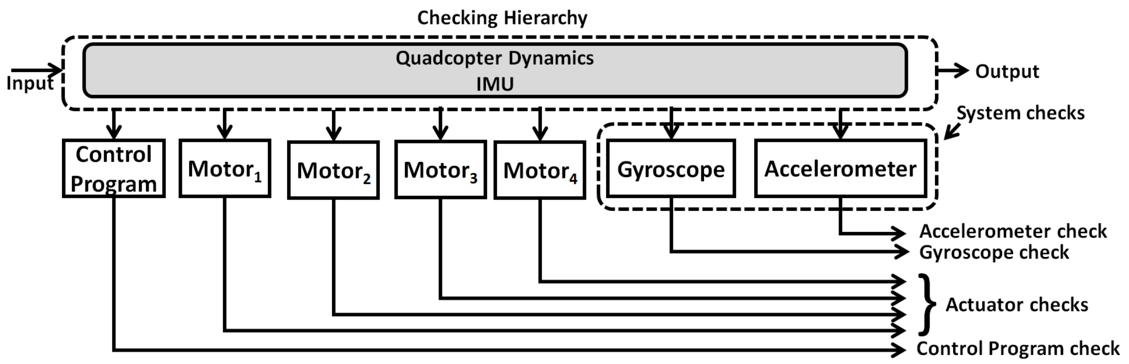


Fig. 4: Hierarchical checking methodology

## 5 Hierarchical Checking Approach

### 5.1 Failure Model

We consider failures in control program execution running on a digital processor, in sensors as well as actuators of the quadcopter system. *Errors in control program execution* are modeled as caused by soft errors in data and datapath control of the digital processor. *Sensor failures* are modeled with transient or permanent effects. Spurious bit-flips in digitized sensor values are used to model transient sensor errors. Permanent effects are modeled by parametric deviations that cause quadcopter control algorithms to malfunction. Finally, we considered *actuator failures*. These are modeled as parametric deviations in electro-mechanical parameters of the brushless DC motors of the quadcopter (such as loss of torque).

### 5.2 Checking methodology: state space checks

The checks are implemented in hierarchical manner as the proposed checking methodology. For the quadcopter case, the system can be divided as an ensemble of a couple of subsystems, namely - the controller of the system and the BLDC motors as actuators. Controller generates the control action and each BLDC motors receives this control action which produces the thrust. Additionally, the IMU unit is producing the sensor readings for the controller. A block diagram of the proposed hierarchical checking mechanism for a quadcopter is shown in Figure 4. At the highest level of the design, state space checks (described next) are implemented to detect gyroscope and accelerometer sensor errors. At the lower level of the design, one check was implemented for control program errors and four checks are implemented for each of the four quadcopter actuators. Such a hierarchical decomposition of checks allows ease of failure diagnosis down to individual subsystems while allowing multiple simultaneous failures to occur without compromising detectability. We assume that control program errors can occur concurrently with motor errors but that sensor errors can occur only in isolation. Table 1 summarizes the diagnosis strategy.

Table 1: Diagnosis summary

| | | Checks | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | $chk_{act}1$ | $chk_{act}2$ | $chk_{act}3$ | $chk_{act}4$ | $chk_{ctrl}$ | $chk_{acc}$ | $chk_{gy}$ |
| Failures | Motor1 | × | | | | | × | × |
| | Motor2 | | × | | | | × | × |
| | Motor3 | | | × | | | × | × |
| | Motor4 | | | | × | | × | × |
| | Control Program | | | | | × | × | × |
| | Linear acceleration | | | | | | × | |
| | Angular rate | | | | | | | × |

#### 5.2.1 Sequential model based state checking

Figure 5 shows the generation of the implemented state space based check for a nonlinear state variable system [23]. Each column within the dashed block of Figure 5 represents a vector of observable state measurements and inputs obtained across different slices of time. Here, the state trajectory (estimated using an Extended Kalman filter) of the nonlinear state variable system and inputs are recorded across a pre-defined observation window $t_W$. For prescribed state of the system and possible input $u(t)$, a quantity of the states of the system $v_c$ is computed using a linear weighted sum, $F_c$ [23]. In this work, $v_c$ consists of the sensor or actuator value for which the check is computed (the objective being to detect errors in the respective sensor or actuator or a control program error). A machine learning algorithm based nonlinear model, $F_m$ is used which takes the ensemble trajectory of state vectors

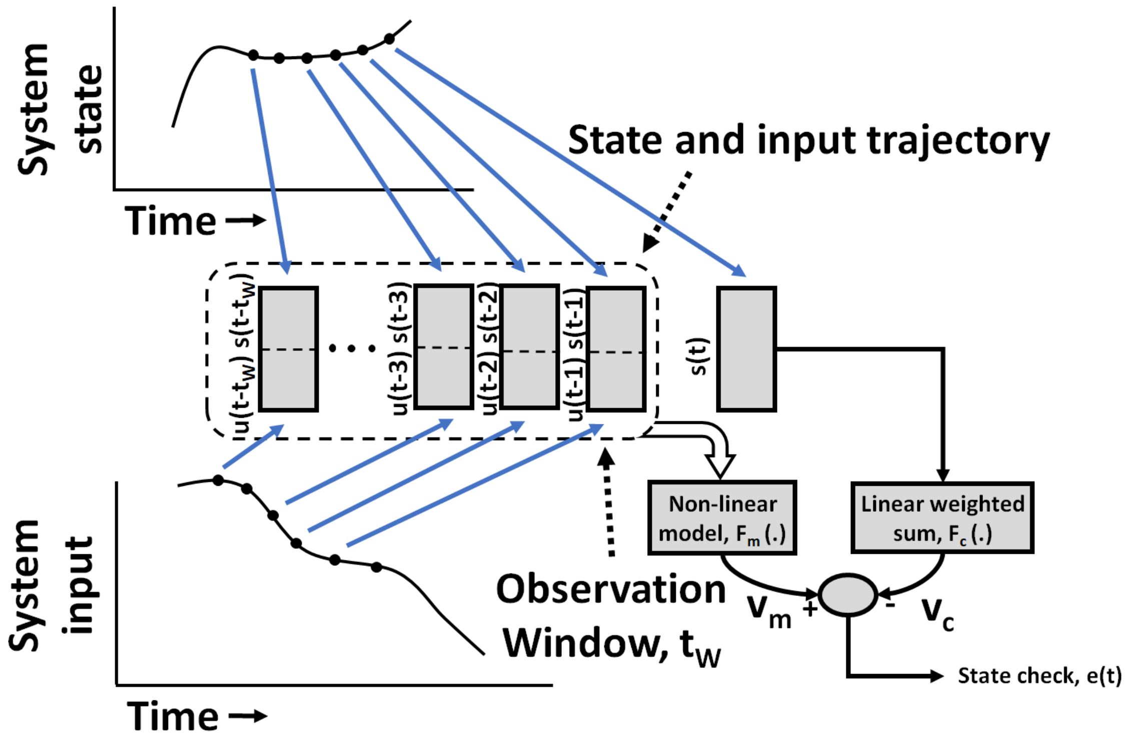


Fig. 5: Computation of state space based check

$s(t-1), s(t-2), \ldots, s(t-t_W)$ and the corresponding inputs $u(t-1), u(t-2), \ldots, u(t-t_W)$ as input and estimates $v_m$. Training of the machine learning system is performed for normal system operation under diverse input stimuli. The quantity $v_m - v_c$ is defined as the *error signal*. Ideally, when the learning of $F_m$ is complete, the *error signal* given by $v_m - v_c$ is zero or near zero under fault free conditions, whereas this will not be true when the system is going through failure. This will detect the failure. To compute $v_m$, a recurrent neural network based neuromorphic system is used in this research [40] which is explained in subsection 5.2.2.

### 5.2.2 Use of time-series model for computing $v_m$

To compute $v_m$, the set of prior states of the system as well as inputs (see Figure 5) are analyzed in a sequential manner, and *Recurrent Neural Networks (RNNs)* (or its variants) can be utilized to perform this function. RNNs were introduced by Siegelmann et al. in [40] and are based on feedback of the output of a perceptron to its input, and they have been successfully used in various applications [41–48]. Although RNN expresses memory effects and can analyze sequential data, it suffers from long term memory loss. To resolve this, a variant of RNNs is introduced, called *Long Short Term Memory (LSTM)* [49], and is used in this research. A block diagram of an LSTM cell is shown in Figure 6 where im-

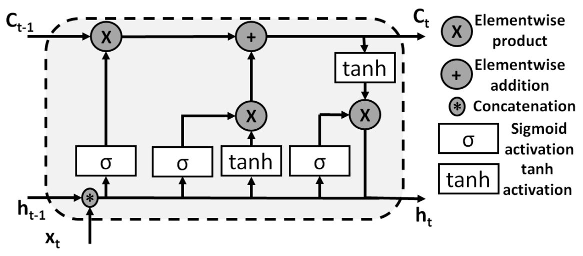


Fig. 6: Long Short Term Memory Block

portant information from one time instant to next pass through with intermediate variables $C_{t-1}$ and $h_{t-1}$. Unlike RNNs, LSTMs selectively suppress unimportant information from the input by multiplying the input with a *sigmoid* activation function. Also, the *tanh* activation function limits the inputs to lie between -1 and 1. The details of this approach can be found in [49].

### 5.2.3 Trade-off between observation window size and accuracy of $v_m$ prediction

In this work, $v_m$ prediction by LSTM based model $F_m$ is employed to match the observed $v_c$ value based on prior sensor and actuator measurements as well as system inputs. However, the training data for such an LSTM based predictor is subject to process and measurement noise. In addition, the length of the observation window $t_W$ is important as a value of $t_W$ less than the memory latency of the system causes prediction errors while too long a value of $t_W$ can cause overfitting. We have studied the LSTM based $v_m$ prediction methodology to understand how prediction accuracy depends on the size of the observation window. Figure 7 shows

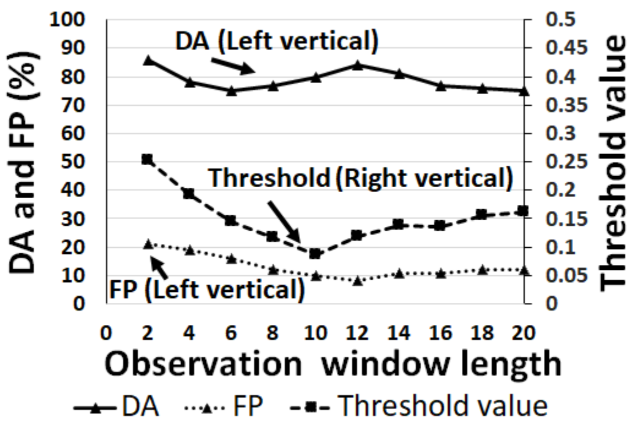


Fig. 7: Trade-off between observation window and accuracy for $\sigma^2_{noise}$ = 0.05

performance of the LSTM based state space check with respect to size of the observation window for a nominal fault-free system. Here, the model was constructed with two hidden layers and 10 LSTM cells per layer. Process and measurement noise was injected to study the trade-offs. The threshold in Figure 7 refers to the *largest value of* $v_m$-$v_c$ for a given $t_W$. An error is detected only if it causes the instantaneous value of $v_m$-$v_c$ to be larger than this threshold. In the nominal system, the presence of noise can cause spurious detection of error resulting in *false positives (FP)* which are defined as $\frac{\#\, of\, detected\, faults\, due\, to\, noise}{\#\, of\, total\, fault-free\, experiments}$. On the other hand, for systems experiencing failures, we define *detection accuracy (DA)* = $\frac{\#\, of\, detected\, faults}{\#\, of\, total\, faulty\, experiments}$ to denote the coverage of such failures using the proposed checking methodology. In Figure 7, the x-axis shows the length of the observation window and the y-axis shows DA(%), FP(%) and threshold value. From this figure, it is observed that, increasing the observation window reduces the value of the threshold at first and then the value of threshold increases again. At first, the threshold is high as the model has access to reduced state transition information. For this reason, the check shows higher tendency to flag errors. This phenomenon results in a high $FP$ value. Additionally, with a small observation window, injected faults are detected as well. That is why, high values of $DA$ and $FP$ are observed with a small observation window. However, the threshold decreases with increase in the size of the observation window as the model has access to more information and can learn the dynamic behavior of the quadcopter more accurately. This reduces the $FP$ value, however the $DA$ value stays almost the same. So, the model gradually performs better with increase in the observation window. However, as the observation window increases further, the model for computing $F_m$ results in overfitting. As a result, the threshold increases, and this deteriorates the $DA$ and $FP$ value again.

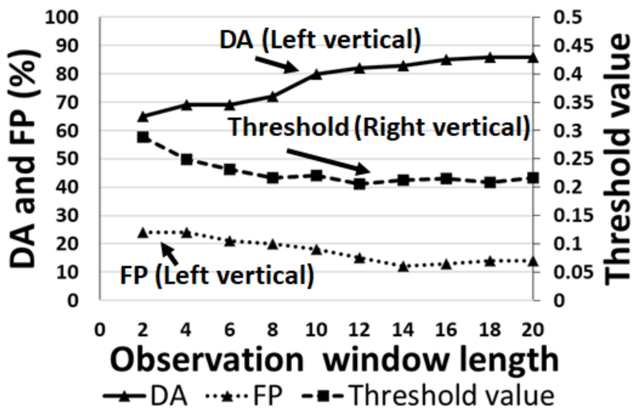


Fig. 8: Trade-off between observation window and accuracy for $\sigma^2_{noise}$ = 0.25

To determine the dependence of the window length with respect to noise power, we repeated the experiment for noise power, $\sigma^2_{noise}$ = 0.25 as shown in Figure 8, whereas it was $\sigma^2_{noise}$ = 0.05 in previous study. For the higher level of noise power, higher value of threshold was observed as the data are noisier compared to previous one, and hence the error threshold has higher value in nominal case. For this reason, it is more difficult to detect the noisy but nominal response (hence, high values of false positives were observed) and to have better detection accuracy. As a result, we observe that, for higher noise power, in general, the threshold is higher and FP is higher for same window length. Additionally, we observe similar kind of DA for higher value of window length as noise power increases. We observe that these three metrics are almost plateaued for window length of greater than 12, which represents that for higher noise power, the window length is increasing as the model needs access to more data to better understand the system behavior.

We considered $\sigma^2_{noise}$ = 0.05 for this work and from Figure 7, it is observed that for prediction window length of 10, the $DA$ ≥ 80% and $FP$ ≤ 10% and the model stays comparatively simpler, which makes a corresponding selection of window length a good choice.

Hence, this was selected as the check model architecture in this work.

## 5.3 Hierarchical check infrastructure for the quadcopter system

According to the study conducted in Section 5.2.3, the LSTM based models used in this study consists of 2 hidden layers with 10 LSTM cells in each layer. In the following, detailed description of each of the checks are presented.

### *5.3.1 Control program check*

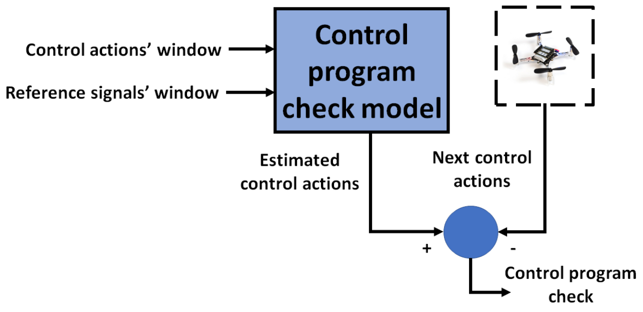

Control actions' window
Reference signals' window
Control program check model
Estimated control actions
Next control actions
+
-
Control program check

Fig. 9: Control program check

The control program check takes the current control action and the current reference signal of the quadcopter system as input and estimates the next control action. The control actions are separate for each actuator and hence, the control action is a vector of size $4 \times 1$. Additionally, the system reference inputs are the $x$, $y$, and $z$ coordinates of the quadcopter destination which are 3 dimensions in nature. Hence, this quantity is a vector of size $3 \times 1$. So, the control program check's inputs and outputs sizes are $7 \times 1$ and $4 \times 1$ respectively. If the ensemble control actions are denoted by $U(t)$ $(= [U_1, U_2, U_3, U_4]^T$, defined in subsection 4.4), then the control program check, $chk_{ctrl}$ is defined as $chk_{ctrl} = U(t)_{predicted} - U(t)_{actual}$, and it is of size $4 \times 1$.

### *5.3.2 Sensor check*

Two checks for *gyroscope* and *accelerometer* are employed which check for angular rate and linear acceleration respectively. They take the trajectory of inputs of the system, $u(t)$, angular rate $\omega(t)$, and linear acceleration, $a(t)$ as the input and predicts the next angular

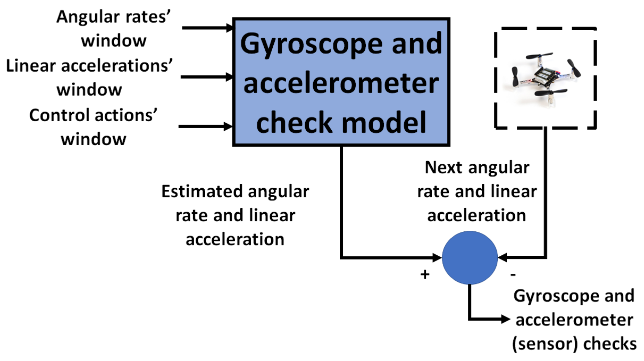

Angular rates' window
Linear accelerations' window
Control actions' window
Gyroscope and accelerometer check model
Estimated angular rate and linear acceleration
Next angular rate and linear acceleration
+
-
Gyroscope and accelerometer (sensor) checks

Fig. 10: Sensor (gyroscope and accelerometer) check

rate $\omega(t)_{predict}$ and linear acceleration, $a(t)_{predict}$ respectively. Then gyroscope check $chk_{gy}$ and accelerometer check $chk_{acc}$ are computed as, $chk_{gy} = \omega(t)_{predict} - \omega(t)_{actual}$, and $chk_{acc} = a(t)_{predict} - a(t)_{actual}$. We looked at linear acceleration and angular rate at each direction separately, and hence both $chk_{gy}$ and $chk_{acc}$ are vectors of size $3 \times 1$. In this work, they have been merged together to share model resources efficiently, and hence they are considered as one unit.

### *5.3.3 Actuator check*

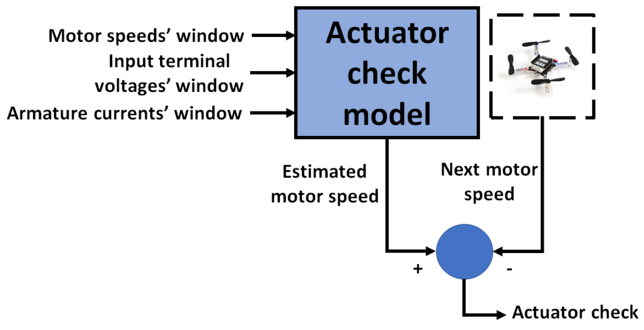


Fig. 11: Actuator check

For the actuator check, the LSTM based machine learned model receives the windows of motor speed $\Omega$, input terminal voltage and armature current as model input and estimates the next motor speed. Here, we assume a small resistor to sample the armature current, which does not increase the cost significantly. Additionally, this resistance is normally very small compared to the rotor resistance and thus, they have minimal effect on actuator behavior. Finally, the actuator check, $chk_{act}$ is defined as $chk_{act} = \Omega(t)_{predict} - \Omega(t)_{actual}$. As we are comparing only one quantity in this check, this check is scalar in nature. However, as there are four actuators in this system, four actuator checks are employed.

### 5.4 Resilience methodology

A procedural pseudo code for the resilience methodology for managing failures in control program, sensors, and actuators is given in Algorithm 1, and the resilience methodologies are discussed in this subsection.

**Algorithm 1** Resilience methodology pseudo code

```
Require: $chk_{ctrl}, chk_M, chk_{gy}, chk_{acc}$
 1: while $t \le t_{End}$ do
 2:   Evaluate $chk_{ctrl},, chk_M, chk_{gy}, chk_{acc}$
 3:   if $chk_{ctrl} \ge th_{ctrl}$ or $chk_{act} \ge th_{motor}$ or $chk_{gy} \ge th_{gy}$ or $chk_{acc} \ge th_{acc}$ then
 4:     if $chk_{ctrl} \ge th_{ctrl}$ then
 5:       /* Control program fault */
 6:       $Estimate\, control\, action$
 7:       $Restore\, control\, action$
 8:     end if
 9:     if $chk_{gy} \ge th_{gy}$ and ($chk_{act} < th_{motor}$ and $chk_{ctrl} \ge th_{ctrl}$) then
10:       /* Gyroscope fault */
11:       $Estimate\, gyroscope\, value$
12:       $Restore\, gyroscope\, value$
13:     end if
14:     if $chk_{acc} \ge th_{acc}$ and ($chk_{act} < th_{motor}$ and $chk_{ctrl} \ge th_{ctrl}$) then
15:       /* Accelerometer fault */
16:       $Estimate\, accelerometer\, value$
17:       $Restore\, accelerometer\, value$
18:     end if
19:     if $chk_{act} \ge th_{motor}$ then
20:       /* Actuator fault */
21:       $estParam \leftarrow actuatorParameterEstimator$
22:       $Reconfigure\, controller$
23:     end if
24:   end if
25: end while
```

#### 5.4.1 Control program fault

Control program execution suffers from two failure mechanisms, namely control flow and data flow errors. Control flow errors can be detected by 'watchdog timers' or 'control flow error checks' [50]. Both control and data flow errors that occur due to spurious bit-flips can result in incorrect system actuation.

In our approach, both control and data flow errors are managed using *state restoration* (line 7 in Algorithm 1). In this approach, once an error is detected, the erroneous control actions are replaced with their predicted values at time $t$ from the respective checks as described earlier. Additionally, to address the control flow error, a register is periodically set and reset at the start and end of control action computation routine at the rate at which actuation is enacted by the control program. In the case of undesired jump/halt/branch to unexpected/undefined locations etc., the register will not reset at the end of the routine which denotes the existence of such failure. In this case, we reset the control computation routine and continue applying the last properly computed action. In implementing the checks, care has to be taken to ensure that the software kernels implementing the checks are run as independent tasks on a digital co-processor. Also, redundant checks are used to flag errors in the check software itself, in which case no corrective action is taken.

#### 5.4.2 Sensor fault

Two failure mechanisms, namely transient failures (which occur due to induced noise in the system, power supply and ground bounce) and parametric deviations (these occur due to field degradation or regular wear and tear) were considered for sensors (gyroscope and accelerometer). In this work, we consider only internal failure of the systems. In the presence of long-lasting deliberate attempt to attack the system, the best step would be to take fail-safe approach (for example, bringing the quadcopter down on the ground). The only other way to manage the latter is through the use of redundant sensors. This is a broad topic and not within the scope of the presented research (the work focuses on failure mitigation, not security attacks).

In a real-life system, both gyroscope and accelerometer readings are converted from analog to digital signal with the help of Analog to Digital Converter (ADC) whose sampling interval is in the neighborhood of $\mu s$ compared to time constant of quadcopter (in $ms$). Hence, we have assumed the output of ADC to be readily available. Hence, once the failure is detected in any sensors, it is enough to replace their faulty value with predicted values at time $t$, in digital domain, from their respective checks (line 12 and 17 in Algorithm 1), and this is what was adopted in our approach.

#### 5.4.3 Actuator fault

For the correction of actuator parametric deviations (such as due to change of torque in a motor), we propose controller reconfiguration on-the-fly. The fault can be detected and diagnosed in real-time by observing the check values. After the fault is diagnosed, the observed actuator check is sampled in high frequency for $t_k$ time points (to adequately capture system behavior) and this recorded quantity is defined as $chk_{observed}(t_k)$. Similar to Section 5.2.3, a trade-off study of check signal length

$t_k$ and optimization accuracy was performed to obtain the best check signal length, and a length of 55 time points (equivalent to 55 ms time) ensured both fast and accurate estimation. We have performed optimization of the state/check prediction function over several signal lengths and we found that for high values of signal length, above 55 ms, the accuracy saturates and does not improve further. This was done with respect to the specific quadcopter model considered in our simulation studies. A 'Golden Section Search' based optimization [51] is performed to estimate the changed parameter (line 22 in Algorithm 1), $p$ which is defined as:

$$p^* = \arg\min_p (\|chk_{observed}(t_i) - chk(act(p, t_i))\|_2) \\ \text{subject to, } 0 < t_i \leq 55 \quad (5)$$

where, $chk(act(p, t_i)$ is the actuator check for parameter $p$ and at time point $t_i$. After the changed parameter value is obtained, a controller is designed using 'Successive Loop Closure' method [38]. For different back-emf constants, the optimal motor controller coefficients were pre-computed and stored in a form of 'Look up table' (LUT). In this work, these controller parameters are the coefficients of the PID controller. Once the estimated parameter is obtained, this LUT is used to reconfigure the controller on the fly (line 19 in Algorithm 1).

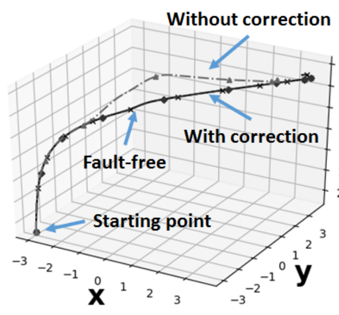
## 6 Experimental Results

A quadcopter with the following design parameters were considered: $L = 0.3\,\text{m}$, $r = 0.1\,\text{m}$, $m = 1.2\,\text{kg}$, and $b = 0.0245$ , propeller diameter $d = 10\,\text{inch}$, and propeller pitch $= 4.5\,\text{inch}$ . As the actuator, brushless DC motors were used with the following parameters: $R_s = 0.52\,\Omega$, $\lambda = 0.0137\,V/rad/s$, $J = 1.2 \times 10^{-5}\,Kg/m^2$, $B_f = 0.01\,Nms$, $P = 2$, $L_{act} = 3.6 \times 10^{-5}\,H$, and $M_{act} = 1.2 \times 10^{-6}\,H$. $\hat{f}_a(.)$, $\hat{f}_b(.)$ and $\hat{f}_c(.)$ are assumed to be trapezoidal (which follows linear relation and gets clipped when absolute value is larger than $0.01264\,V/rad/s$). A quadcopter flight was simulated from one source to one destination through some points along the way. These points were selected such that all subsystems of the quadcopter would be adequately exercised. Here, we are interested in the quadcopter system behavior as a whole. As the time constants of accelerometer/gyroscope are in $\mu$s and that of the quadcopter system is in ms, we have assumed that the accelerometer and gyroscope readings are stabilized within short amount of time. That is, the internal subsystem blocks are stabilized (i.e. their own time-constants are elapsed), and hence, it will be able to capture the behavior of these subsystems well. However, the actuator's time constant is comparable to quadcopter system time constant (in ms) and this was considered in this study. As the actuators are always in continuous stimulation, the new actuation gets stabilized quickly.

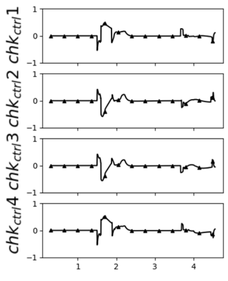
For the failure in execution of control program, faults were introduced in the data of 16-bit long control action in the form of spurious bit-flips in multiple locations where bit-flips were injected in random bit position in random manner. Same word length was considered for accelerometer and gyroscope sensor readings as well for transient failures. As they are introduced in the digital word and every bits of a digital word have same failure probability, this translates into high probability of faults in these subsystems. Additionally, we observed a few cases where a failure in the nominal system resulted in a crash during the hardware validation stage (explained in Subsection 6.8). For parametric failures of accelerometer and gyroscope, a 15% offset was introduced in the readings which are the most common form of parametric failures. For parametric failures of actuators, the back emf constant was gradually reduced to 80% as the failure model. To evaluate the performance of the proposed approach on different experiments, 2 quantities $v_1 = \|Trajectory_{withoutCorrection} - Trajectory_{reference}\|_2$ and $v_2 = \|Trajectory_{withCorrection} - Trajectory_{reference}\|_2$ were defined. These are the L2 norm of quadcopter's 2 trajectories (without correction and with correction) with respect to a reference trajectory which the system would follow in absence of all faults. For every injected faults, the fault detection, and diagnosis were performed by the method explained in [29].

### 6.1 Errors in Execution of control Program

Figure 12 shows the detection, and correction efficacy of the proposed approach for Control program malfunction. The trajectory of the quadcopter system, along with its control program check, is plotted where a transient fault was injected at $t = 1.5$ sec when the control program data gets corrupted and it creates incorrect control action. Here, the faults were injected in the data of the control program in the form of spurious bit-flips which can result from highly energetic radiation such as alpha particle strike. As seen from Figure 12a, the trajectory is changed to great extent when no correction is performed which can result into crash with obstacles. However, when the correction approach is employed, the system followed its reference trajectory. Additionally, as observed from Figure 12b, the faults have been



(a)

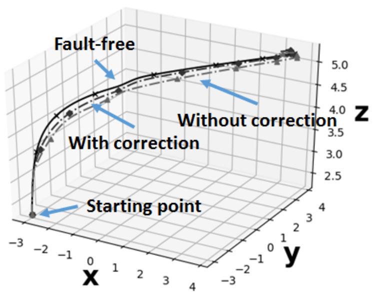


(b)

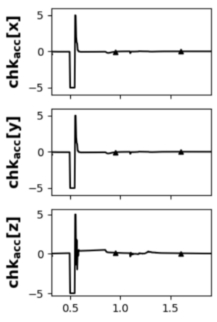
Fig. 12: a) Trajectories of quadcopter for control program fault, and b) Corresponding control program check plot

detected successfully, in real-time, by the control program check. The performance metrics of the proposed approach, $v_1$ and $v_2$ were evaluated and are shown in Table 2. This shows the improvement and hence the efficacy of the proposed approach.

### 6.2 Sensor Transient Errors: Accelerometer



(a)

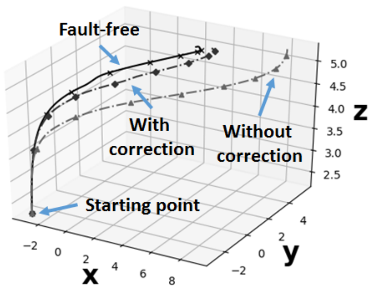


(b)

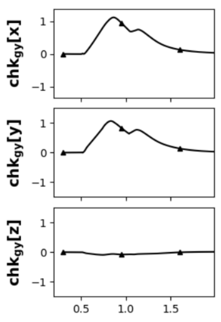
Fig. 13: a) Trajectories of quadcopter for accelerometer transient fault, and b) Corresponding accelerometer check plot

Proposed approach was applied to transient error injected in accelerometer sensor as 2nd experiment. A transient error was injected at $t = 0.5$ sec, as a form of spurious bit-flips, at accelerometer sensor reading which changed the trajectory of the quadcopter as observed in Figure 13a. It can also be observed from Figure 13b that the accelerometer check were able to detect the failure instantaneously and proposed correction method was able to correct the sensor error and to restore the trajectory of the system. The performance metrics $v_1$ and $v_2$ of the proposed approach were computed and are given in Table 2.

### 6.3 Sensor Transient Errors: Gyroscope



(a)

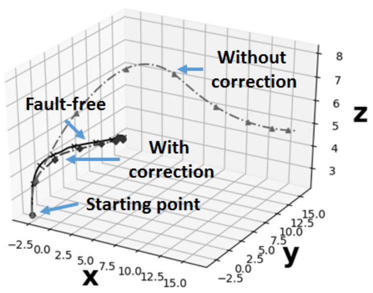


(b)

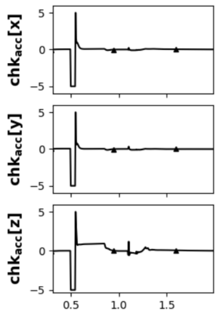
Fig. 14: a) Trajectories of quadcopter for gyroscope transient fault, and b) Corresponding gyroscope check plot

Figure 14 shows the detection, and correction efficacy of the proposed approach when transient error was injected in gyroscope sensor at $t = 0.5$ sec. The fault was detected in real-time as shown in Figure 14b, and the trajectory is changed to a great extent when no correction is performed. However, after applying the proposed methodology, it was possible to improve the trajectory of the system. The comparison of $v_1$ and $v_2$ are given in Table 2. As can be seen from these trajectories and the table, the corrected trajectory was very close to reference one.

### 6.4 Parameter Deviations in Sensors: Accelerometer



(a)



(b)

Fig. 15: a) Trajectories of quadcopter for accelerometer parametric deviation, and b) Corresponding accelerometer check plot

Figure 15 demonstrates the behavior of the proposed approach for the parametric variation of accelerometer. Parametric variation has been modelled as a gradual change in accelerometer behavior from $t = 0.5$ sec to $t = 1.3$ sec. Because of this variation, the system goes through unexpected trajectory (possibly dangerous) which is observed from the Figure 15a. As seen from the accelerometer check, the check produces a non-zero error signal ($chk_{acc}[z]$ produced non-zero signal for the whole duration) which performs the detection, and diagnosis. Additionally, replacing accelerometer reading from the model improved the system trajectory. Similar to previous cases, the comparison of performance is performed in this experiment and it is given in Table 2. Here, the machine learning models were trained under closed loop configuration where the system behavior was monitored and observed behavior was utilized to train the model. During the deployment phase, the behaviors were validated with respect to learned model. In this situation, they are operating in the same manner like in the training phase. Additionally, this corrected information is processed further in the system and these information are used in future time-steps. In this way, they are working in closed loop.

## 6.5 Parameter Deviations in Sensors: Gyroscope

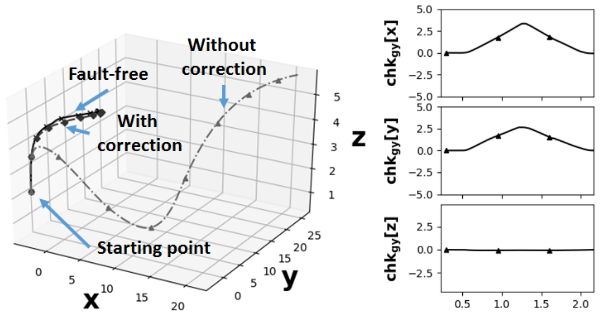


Fig. 16: a) Trajectories of quadcopter for gyroscope parametric deviation, and b) Corresponding gyroscope check plot

We investigated parametric variation of gyroscope as the fifth study which is shown in Figure 16. Similar to accelerometer case, the parametric variation happened in very long time window (from $t = 0.5$ sec to $t = 2.0$ sec), and it was successfully detected and diagnosed in gyroscope check. The system went through widely different and possibly dangerous trajectory with no corrective action, and the trajectory was restored to its expected behavior when the proposed corrective action is applied. The performances were compared and are provided in Table 2.

## 6.6 Parametric Deviations in Actuators

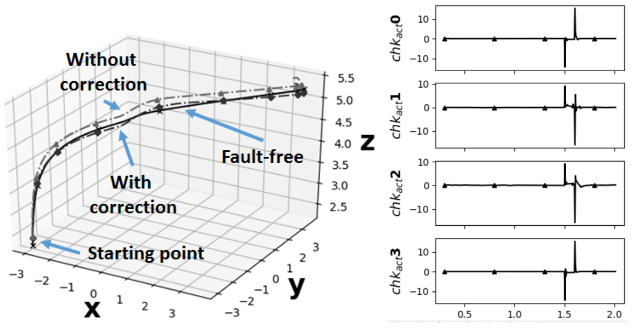


Fig. 17: a) Trajectories of quadcopter for parametric deviation in actuator, and b) Corresponding actuator check plot

In this experiment, the back emf constant of the actuators were slowly changed to 80% of its nominal value from $t = 1.5$ sec to $t = 1.6$ sec to model the parametric deviation. This gradually changed the behavior of the actuator which results in change in control input as well as the output speed of the actuator, and the system trajectory changes. However, parametric deviation introduced non-zero check value in actuator check $chk_{act}$ (see Figure 17b) which detects and diagnoses the source of the fault. An optimization as described in Equation 5 was performed to estimate the changed parameter value, and this estimated parameter value was fed to the '*Look up table*' which predicts new controller parameters. The new control law was employed and this resulted into improved system performance. The trajectories of fault-free, faulty without correction and faulty with correction cases are shown in Figure 17. As seen from Table 2, the L2 norm of trajectory improves when corrective action is employed.

## 6.7 Summary of Experimental Results

Table 2 shows the summary of the experiments where, different failures are indicated across rows of the table and performance of the correction approach is indicated across the columns of the table. As observed from the

Table 2: Summary of Experiments

| | L2 Norm Comparison | |
|---|---|---|
| Faults | Without Correction, $v_1$ | With Correction, $v_2$ |
| control program | 16.17 | 0.69 |
| Accelerometer: Transient | 32.17 | 15.50 |
| Gyroscope: Transient | 64.88 | 12.49 |
| Accelerometer: Parametric | 327.10 | 60.26 |
| Gyroscope: Parametric | 350.17 | 71.30 |
| Actuators: Parametric | 5.42 | 2.28 |

table, the proposed approach was able to correct different kinds of failures with very high accuracy which clearly shows the efficacy of the approach.

## 6.8 Hardware experiments

We implemented the proposed checking methodology on a '*Crazyflie 2.1*' [36] quadcopter system. The quadcopter was physically flown along a reference trajectory, different faults were injected, and proposed correction approaches were applied to manage the faults. The hardware configuration of the system is shown in Table 3. The quadcopter communicates with a PC via Bluetooth interface and a proprietary 'Crazy RealTime Communication Protocol (CRTP)'. The server (on PC) and client (the quadcopter itself) platforms for the setup were implemented using Python and C, respectively. The server was used to send commands to the client and all the necessary computations related to control action generation, necessary system level consistency check for the client, Extended Kalman filter based state estimation, our proposed checks and correction strategy etc. were implemented in the Crazyflie Microcontroller unit (MCU). The client, residing in the quadcopter, performed the necessary maneuver according to its objective and the check values were read back from the client in the form of a log variable. The minimum period for a log variable to be read from the client was 10 ms. This system has limited hardware resource and timing cycle available, and for this reason, model and code optimization for our proposed approach was necessary which is described below:

Table 3: Hardware configuration of Crazyflie 2.1

| Physical parameters/Components | Specification/Model |
|---|---|
| System mass | 27 gm |
| Size (W × H × D) | 92 × 92 × 29 mm |
| Radio Band | 2.4 GHz |
| Coomunication type | Bluetooth |
| Coomunication protocol | Crazy RealTime Protocol (CRTP) |
| Main microcontroller Unit | STM32F405<br>Cortex-M4<br>Clock frequency: 168MHz<br>RAM: 192KB<br>Flash: 1MB |
| Radio and power management unit | nRF51822 |
| 3 axis accelerometer / gyroscope | BMI088 |
| Pressure sensor | BMP388 |
| Actuator | 4 DC coreless motors |

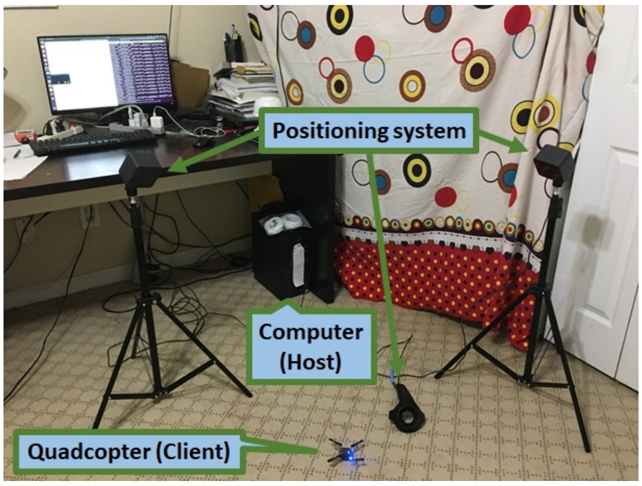


Fig. 18: Hardware setup

### 6.8.1 Neuromorphic model preparation for hardware

The 'Crazyflie 2.1' system has only 192 KB RAM and 1 MB flash memory available which contains all firmware code of the quadcopter system. For this reason, very little RAM and flash spaces are available after the regular code of the system is loaded. Additionally, the quadcopter system is battery operated. Therefore, every computation execution has a direct impact on the total 'operation time' of the system, which means more computation will drop the battery (i.e. flight time) of the system quickly.

Hence, we looked for the best combination for minimally complex, yet accurate neuromorphic models. We have captured the system behavior during its flight and applied different model architectures which would ensure enough accuracy with expense of minimal hardware and computation overhead. For this reason, we have employed the time-series model based on Gated Recurrent Unit (GRU) [52] which has less parameter (less computation overhead and simpler) with almost same accuracy. Additionally, we have performed a combined analysis to assess the performance for all the neuromorphic checks to ensure efficient hardware reuse. We have adopted the method described in Section 5.2.3 to come up with the optimum model (the average mean square error of estimation was less than $5 \times 10^{-3}$), and finally, a generalized model architecture given in Figure 19 was obtained. Note that in the last layer of this model, the number of perceptrons is equal to the number of the output, which varies with model and it is needed to ensure the functionality. However, the remaining model reuses the same hardware.

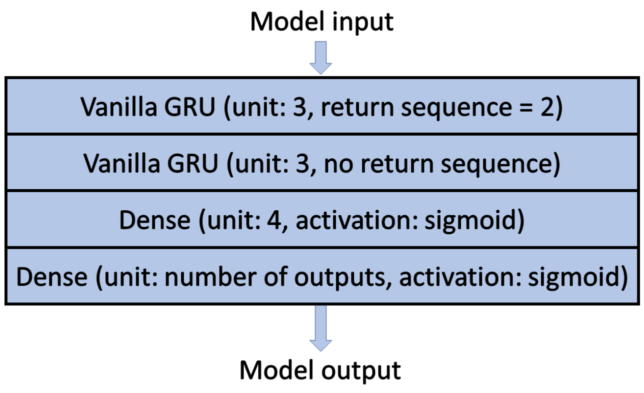

Model input
Vanilla GRU (unit: 3, return sequence = 2)
Vanilla GRU (unit: 3, no return sequence)
Dense (unit: 4, activation: sigmoid)
Dense (unit: number of outputs, activation: sigmoid)
Model output

Fig. 19: Neuromorphic model architecture for Crazyflie 2.1

In this model, the first layer receives data at two time instants, processes the data, and hands this to second layer. This and final layer further process the data and generates the model output. As the model input and output, we have the same quantities which we used to implement previous models. Control program check has windows of control action and reference signal as input and next control action as output. Sensor check has windows of sensor values and control action as input and next sensor reading as output. Actuator check has windows of motor speed, input terminal voltage, and armature current as input and next motor speed as output. Faults were injected into the sensor, actuator circuit, and control program of the quadcopter and the obtained results are discussed below:

### *6.8.2 Control program fault*

We look at failure at the control program execution as the first test case. We introduced transient failure in control program execution in the form of spurious bit-flips in the data of the processor core. Introduction of spurious bit-flips results in incorrect quadcopter control action. Three trajectories of the quadcopter namely - at fault-free or nominal condition (solid), with control program fault when no correction is performed (dashed), and finally with correction approach applied (dotted) are showed in Figure 20. These plots show that after

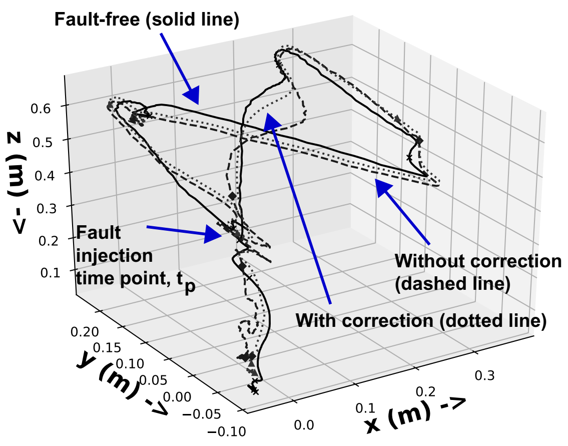


Fig. 20: Quadcopter trajectory in presence of control program fault

'fault injection time point, $t_p$', the proposed correction approach was able to correct the behavior of the quadcopter which demonstrates the efficacy of the proposed approach.

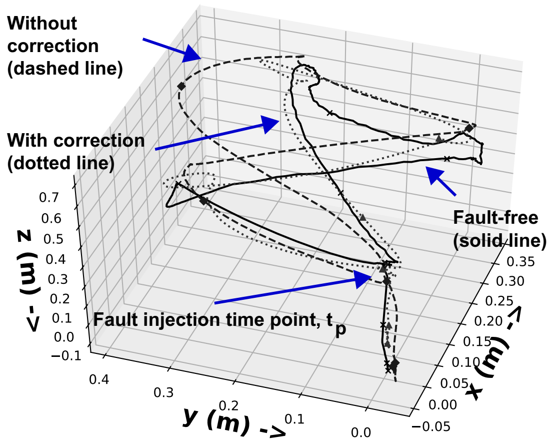


Fig. 21: Quadcopter trajectory in presence of sensor fault

### *6.8.3 Sensor fault*

A transient fault is introduced in the gyroscope reading as the sensor fault in the form of spurious bit-flips. As the gyroscope are micro-electro-mechanical systems (MEMS), it can experience sudden change in movement and register incorrect reading. The trajectories of the system during fault-free (solid), with fault and no correction (dashed), and with correction (dotted) are shown in Figure 21. As can be observed from these trajectories, without fault correction, the trajectory of the quadcopter varies widely from its fault-free case. However, when the correction approach is employed after 'fault injection time point, $t_p$', the proposed error correction approach was able to improve the quadcopter behavior in presence of sensor error.

### *6.8.4 Actuator fault*

We look at the actuator fault as the next failure mode. The quadcopter system has DC motors which produces necessary thrust from its input voltage. The generated thrust from the actuator can vary due to numerous reasons which include change in terminal resistance of the motor or temporary variations at the propeller driven by the motors. The crazyflie system has an internal pulse-width modulator (PWM) circuit which effectively changes the generated torque of an arbitrary motor. In this study, we trained the actuator check model with the PWM module and we introduced temporary speed changes in the motors through PWM module (due to loss of torque which results in change of thrust). We performed experiments with different PWM input which would result into generated torque to 100%, 90%, 80%, and 70% of its nominal value, and computed appropriate controller parameter which would restore the system performance in each case. Finally, a LUT was populated with this information.

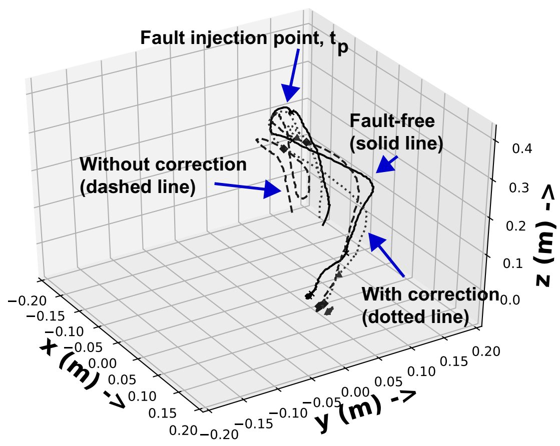


Fig. 22: Quadcopter trajectory in presence of actuator fault

During the deployment, we performed an experiment (see Figure 22) where the system was supposed to hover 0.4 m above the ground in 3-dimensional reference frame. We show three trajectories, namely - fault-free case (solid), when fault is injected but no correction is done (dashed), and finally when fault is injected and correction is performed (dotted). As this was a hovering experiment, the vertical height is of concern and it is plotted in Figure 23 to provide another view of the results. As observed from these plots, the fault was injected at time point, $t_p$ ($t$ = 3.5 sec) when actuators were losing their thrusts, and as a result the system was coming down. When no the controller reconfiguration was performed (dashed line), the height dropped to almost 0.2 m, and it took the system until $t$ = 5.0 sec to take care of this event. However, when the correction approach is applied (dotted), the change in thrusts were readily detected in the actuator check and the controller was reconfigured on-the-fly from LUT. As a result, the system went down to only 0.35 m and was able to recover from the fault within $t$ = 4.5 sec (faster than earlier case). We also observe a small deviation of 0.05m at the beginning of the experiments between without correction and fault-free cases, which can happen due to various reasons. Here, as the experiments were performed in real environment, no two environments are exactly same. For this reason, their behavior may deviate a little (0.05 m deviation at the beginning of Figure 23). Additionally, the behavior will also depend on how the system was initialized in that particular time instant before each experiment, which may change as well. The sensors, and actuators may also be initialized differently. We think these are some of the reasons for this behavior. From this discussion, it shows that, by applying appropriate controller parameter, the behavior of the system was improved.

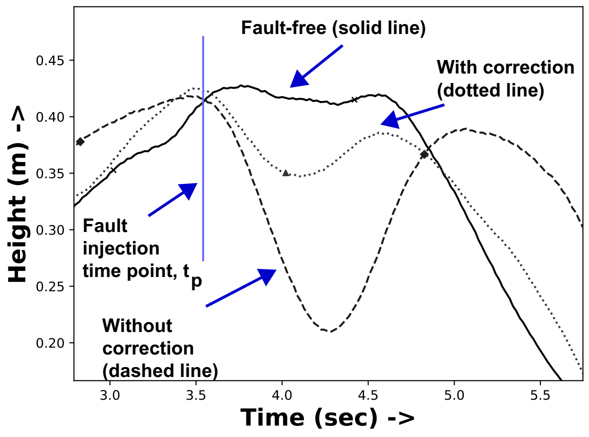


Fig. 23: Controller compensation effect observed from height due to actuator fault

## 7 Conclusion

In this paper, hierarchical checks for general nonlinear systems are proposed. The approach is able to successfully detect, and diagnose failures in different subsystems with small latency. Data obtained for different fault models in different subsystems corroborate the efficacy of the proposed technique. Even compensation based on diagnosed failures was demonstrated for control program execution, sensor, and actuator faults. The method incurs low overhead. Simulation and hardware experiments prove the viability of the proposed resilience methodology.

**Acknowledgements** This research was supported by the Semiconductor Research Corporation under Auto Task 2892.001 and in part by the U.S. National Science Foundation under Grant S&AS:1723997.

## Conflict of interest

The authors declare that they have no conflict of interest.

## References

1. International Organization for Standardization (2011) ISO 26262: Road Vehicles : Functional Safety
2. Harris M (2017) The 2,578 problems with self-driving cars. URL `https://spectrum.ieee.org/cars-that-think/transportation/self-driving/the-2578-problems-with-self-driving-cars`
3. Levin A, Beene R (2019) Sensors linked to boeing 737 crashes vulnerable to failure. URL `https://www.bloomberg.com/news/articles/2019-04-11/sensors-linked-to-737-crashes-vulnerable-to-failure-data-show`
4. Levin A, Beene R (2019) Boeing's crashes expose reliance on sensors vulnerable to damage. URL `https://www.claimsjournal.com/news/international/2019/04/11/290347.htm`
5. Jones R (2018) Report: Uber's self-driving car sensors ignored cyclist in fatal accident. URL `https://gizmodo.com/report-ubers-self-driving-car-sensors-ignored-cyclist-1825832504`
6. Bishop CM (2006) Pattern recognition and machine learning. springer
7. Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: A survey. ACM New York, NY, USA, vol 41, pp 1–58
8. Cork L, Walker R, Dunn S (2005) Fault detection, identification and accommodation techniques for unmanned airborne vehicles. In: Proceedings Australian International Aerospace Congress, AIAC, pp 1–18
9. Thumati BT, Jagannathan S (2010) A model-based fault-detection and prediction scheme for nonlinear multivariable discrete-time systems with asymptotic stability guarantees. IEEE Transactions on Neural Networks 21(3):404–423
10. Lampiri E (2017) Sensor anomaly detection and recovery in a nonlinear autonomous ground vehicle model. In: 2017 11th Asian Control Conference (ASCC), IEEE, pp 430–435
11. Bouibed K, Aitouche A, Bayart M (2009) Sensor fault detection by sliding mode observer applied to an autonomous vehicle. In: 2009 International Conference on Advances in Computational Tools for Engineering Applications, IEEE, pp 621–626
12. Goel P, Dedeoglu G, Roumeliotis SI, Sukhatme GS (2000) Fault detection and identification in a mobile robot using multiple model estimation and neural network. In: IEEE International Conference on Robotics and Automation, IEEE, vol 3, pp 2302–2309
13. Cork L, Walker R (2007) Sensor fault detection for uavs using a nonlinear dynamic model and the imm-ukf algorithm. In: 2007 Information, Decision and Control, IEEE, pp 230–235
14. Guo P, Kim H, Virani N, Xu J, Zhu M, Liu P (2018) Roboads: Anomaly detection against sensor and actuator misbehaviors in mobile robots. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, pp 574–585
15. Napolitano MR, An Y, Seanor BA (2000) A fault tolerant flight control system for sensor and actuator failures using neural networks. Proceedings of Aircraft Design 3(2):103–128
16. Alessandri A, Baglietto M, Parisini T (1998) Robust model-based fault diagnosis using neural nonlinear estimators. In: Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171), IEEE, vol 1, pp 72–77
17. Napolitano MR, Silvestri G, Windon DA, Casanova J, Innocenti M (1998) Sensor validation using hardware-based on-line learning neural networks. IEEE transactions on aerospace and electronic systems 34(2):456–468
18. Borairi M, Wang H (1998) Actuator and sensor fault diagnosis of nonlinear dynamic systems via genetic neural networks and adaptive parameter estimation technique. In: Proceedings of the 1998 IEEE International Conference on Control Applications (Cat. No. 98CH36104), IEEE, vol 1, pp 278–282
19. Momtaz MI, Banerjee S, Chatterjee A (2017) On-line diagnosis and compensation for parametric failures in linear state variable circuits and systems using time-domain checksum observers. In: Proceedings of the IEEE VLSI Test Symposium (VTS), pp 1–6
20. Momtaz MI, Banerjee S, Pandey S, Abraham J, Chatterjee A (2018) Cross-layer control adaptation for autonomous system resilience. In: Proceedings of the IEEE International Symposium on On-Line Testing And Robust System Design (IOLTS), pp 261–264
21. Momtaz MI, Chatterjee A (2019) Hierarchical check based detection and diagnosis of sensor-actuator malfunction in autonomous systems: A quadcopter study. In: Proceedings of the IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), IEEE, pp 316–321
22. Banerjee S, Samynathan B, Abraham J, Chatterjee A (2019) Real-time error detection in nonlinear control systems using machine learning assisted state-space encoding. IEEE Transactions on Dependable and Secure Computing
23. Banerjee S, Chatterjee A, Abraham JA (2016) Efficient cross-layer concurrent error detection in nonlinear control systems using mapped predictive check states. In: Proceedings of the IEEE International Test Conference (ITC), pp 1–10
24. Åström KJ, Wittenmark B (2013) Adaptive control. Courier Corporation
25. Hovakimyan N, Cao C (2010) L1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation. SIAM
26. Ioannou PA, Sun J (2012) Robust adaptive control. Courier Corporation

27. Momtaz MI, Banerjee S, Chatterjee A (2016) Real-time dc motor error detection and control compensation using linear checksums. In: Proceedings of the IEEE VLSI Test Symposium (VTS), pp 1–6
28. Banerjee S, Chatterjee A (2017) Real-time self-learning for control law adaptation in nonlinear systems using encoded check states. In: Proceedings of the IEEE European Test Symposium (ETS), pp 1–6
29. Momtaz MI, Chatterjee A (2019) Hierarchical state space checks for errors in sensors, actuators and control of nonlinear systems: Diagnosis and compensation. In: Proceedings of the IEEE Asian Test Symposium (ATS), pp 141–146
30. Avram RC (2016) Fault diagnosis and fault-tolerant control of quadrotor uavs. PhD thesis
31. Afman JP, Ciarletta L, Feron E, Franklin J, Gurriet T, Johnson EN (2018) Towards a new paradigm of uav safety. arXiv preprint arXiv:180309026
32. Yu DL, Chang TK, Yu DW (2005) Fault tolerant control of multivariable processes using auto-tuning pid controller. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 35(1):32–43
33. Margaliot M, Langholz G (1999) Fuzzy lyapunov-based approach to the design of fuzzy controllers. Fuzzy sets and systems 106(1):49–59
34. Seborg DE, Mellichamp DA, Edgar TF, Doyle III FJ (2010) Process dynamics and control. John Wiley & Sons
35. Sabatino F (2015) Quadrotor control: modeling, nonlinear control design, and simulation. Master's thesis, KTH Royal Institute of Technology
36. Bitcraze (2020) Crazyflie 2.1. URL https://www.bitcraze.io/products/crazyflie-2-1/
37. Muruganantham N, Palani S (2010) State space modeling and simulation of sensorless permanent magnet bldc motor. vol 2, pp 5099–5106
38. Beard RW, McLain TW (2012) Small unmanned aircraft: theory and practice. Princeton University Press
39. Tanveer M, Ahmed SF, Desa H, Warsi F, Joyo M (2013) Stabilized controller design for attitude and altitude controlling of quad-rotor under disturbance and noisy conditions. American Journal of Applied Sciences 10:819–831
40. Siegelmann HT, Sontag ED (1991) Turing computability with neural nets. Applied Mathematics Letters 4(6):77–80
41. (2016) A neural network for machine translation, at production scale. URL https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html
42. Tomáš Mikolov LBJartin Karafiát, Khudanpur S (2010) Recurrent neural network based language model. pp 1045–1048
43. Mikolov T, Kombrink S, Burget L, Černocký J, Khudanpur S (2011) Extensions of recurrent neural network language model. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 5528–5531
44. Sutskever I, Martens J, Hinton G (2011) Generating text with recurrent neural networks. In: Proceedings of the International Conference on International Conference on Machine Learning, ICML'11, pp 1017–1024
45. Ayata D, Saraclar M, Özgür A (2017) Busem at semeval-2017 task 4a sentiment analysis with word embedding and long short term memory rnn approaches. In: Proceedings of the International Workshop on Semantic Evaluation (SemEval-2017), pp 777–783
46. You Q, Jin H, Wang Z, Fang C, Luo J (2016) Image captioning with semantic attention. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4651–4659
47. Mao J, Xu W, Yang Y, Wang J, Huang Z, Yuille A (2014) Deep captioning with multimodal recurrent neural networks (m-rnn). arXiv preprint arXiv:14126632
48. Miao Y, Gowayyed M, Metze F (2015) Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. In: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp 167–174
49. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural computation 9:1735–80
50. Johnson BW (ed) (1988) Design &Amp; Analysis of Fault Tolerant Digital Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
51. Kiefer J (1953) Sequential minimax search for a maximum. Proceedings of the American mathematical society 4(3):502–506
52. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:14061078