REAL-TIME ERROR DETECTION AND CORRECTION FOR ROBUST OPERATION OF AUTONOMOUS SYSTEMS USING ENCODED STATE CHECKS

A Dissertation
Presented to
The Academic Faculty

By

Md Imran Momtaz

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
College of Engineering

Georgia Institute of Technology

August 2021

REAL-TIME ERROR DETECTION AND CORRECTION FOR ROBUST OPERATION OF AUTONOMOUS SYSTEMS USING ENCODED STATE CHECKS

Thesis committee:

Dr. Abhijit Chatterjee
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mark Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hao-Min Zhou
School of Mathematics
Georgia Institute of Technology

Dr. Samuel Coogan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date approved: Apr 30, 2021

To my parents - A K M Momtaz Uddin and Homayra Safina Shabnam, my spouse - Murshida Parven, and my child - Zayaan Momtaz, who encouraged and stood by me in this journey.

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor Prof. Abhijit Chatterjee who has over the past few years guided, taught and helped me. His technical expertise and constant drive and excitement about the state of the art research inspired me a lot. Additionally he has been a role model and a friend to his students and I hope I can imbibe some of his generosity, understanding, kindness in my personal life.

I would also like to thank my committee members for their valuable comments about my research. Georgia Tech would not be the place it is without the many excellent faculty and I would like to thank all the faculty members whom I have been lucky to learn from.

I would also like to thank my funding agency - National Science Foundation, and the Semiconductor Research Corporation for their support to carry out this research. This would not be possible of their support.

I would like to mention all my friends at Georgia Tech and Atlanta who made these years such fun. A special thanks to Debashish Banerjee, Sabyasachi Deyati, Barry Muldrey, Suvadeep Banerjee, Sujay Panday, Chandramouli Amarnath, Kwondo Ma, Joshua Wells, Jun Yang Lei, Sanya Gupta, Suhasini Komarraju and Mouhyemen Khan for making this journey enjoyable.

Finally, I would like to acknowledge my parents: A K M Momtaz Uddin and Homayra Safina Shabnam, my sistem Tanzim Mahrim, my spouse Murshida Parven and my son Zayaan Momtaz. My parents sacrificed a lot to provide me a cushion from the harsh difficulties of life. My spouse and son had provided every support possible during this journey. Their smile was the only thing which made me move forward.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of the proposed research is to develop methodologies, support algorithms and software-hardware infrastructure for detection, diagnosis, and correction of failures for actuators, sensors and control software in linear and nonlinear state variable systems with the help of multiple checks employed in the system. This objective is motivated by the proliferation of autonomous sense-and-control real-time systems, such as intelligent robots and self-driven cars which must maintain a minimum level of performance in the presence of electro-mechanical degradation of system-level components in the field as well as external attacks in the form of transient errors. A key focus is on rapid recovery from the effects of such anomalies and impairments with minimal impact on system performance while maintaining low implementation overhead as opposed to traditional schemes for recovery that rely on duplication or triplication. On-line detection, diagnosis and correction techniques are investigated and rely on analysis of system under test response signatures to real-time stimulus. For on-line error detection and diagnosis, linear and nonlinear state space encodings of the system under test are used and specific properties of the codes, as well as machine learning model based approaches were used are analyzed in real-time. Recovery is initiated by copying check model values to correct error for sensor and control software malfunction, and by redesigning the controller parameter on-the-fly for actuators to restore system performance. Future challenges that need to be addressed include viability studies of the proposed techniques on mobile autonomous system in distributed setting as well as application to systems with soft as well as hard real-time performance constraints.

CHAPTER 1
INTRODUCTION

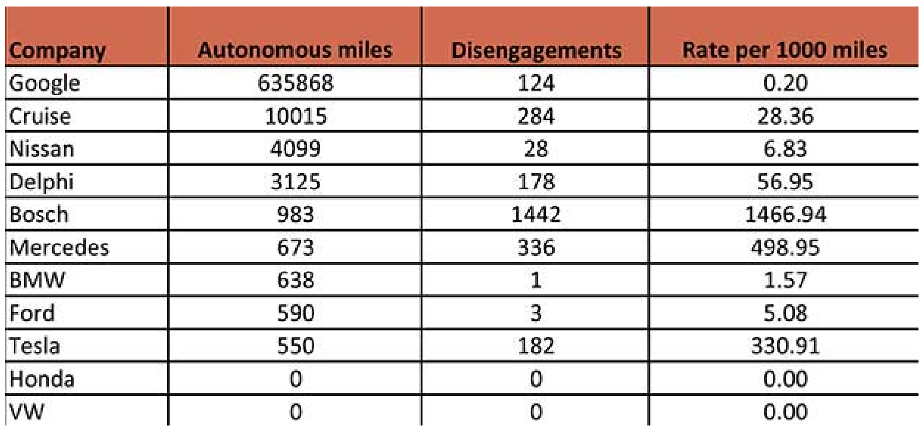
1.1 Motivation

Intel launched the first commercial microprocessor chip, the 4004 containing 2300 tiny transistors, in November 1971. Since then, the computational capabilities integrated chip has been governed by Intel's co-founder Gordon Moore which predicts that processing power doubles roughly every two years as smaller transistors are packed more tightly onto silicon wafers, boosting performance and reducing costs. The attempt to satisfy this rule has achieved exponential progress in computing performance that is orders of magnitude superior than the 1971 processor. A modern Intel Coffee Lake processor contains more than 2 billion transistors and delivers 500,000 times as much computational capability as a 1971 processor. This computing power can be experienced from the fact that a modern smartphone is more capable than a room-sized supercomputer in the 1980s.

After almost five decades of successful execution of Moore's law, the trend of exponential performance achievement are slowed down as reported in [1–6]. Geometric scaling of transistor feature sizes were performed Until the 2000s. After the 2000s, continuing further geometric scaling posed numerous problems and the semiconductor industry kept devising sophisticated technical measures to keep pace with the predictions of Moore's law [7] - at 90 nm, strained silicon was introduced; at 45 nm, new materials (high-κ dielectric) to increase the capacitance of each transistor layered on the silicon were introduced; at 22 nm, trigate transistors [8] maintained the scaling. Technology scaling down to the 22nm node and beyond has led to new challenges [9] in the design of analog and digital signal processing and control systems. Noise margins have reduced considerably making operation of analog, switched-capacitor and digital circuits on the same silicon substrate difficult and hazardous. As an example, power supply and ground bounce induced noise and signal coupling from adjacent “aggressor” interconnections are critical problems that can cause malfunction of electronic circuits. The problem is made worse by the fact that precise simulation of all electrical aspects of the design including the interfaces between digital and analog circuitry, coupling through the power and ground planes, etc., across all process variability effects is a very difficult problem to solve and is very expensive as well. In this context, *resilience* of mixed-signal Systems-on-Chips (SoCs) to soft errors due to radiation, electrical bugs and errors induced by low voltage operation (for reasons of reduced power consumption) has become a critical design problem.

While the last decade saw major developments in the areas of computing and communication, the next decade will see a proliferation of autonomous sensor networks and robots. The successful deployment and assimilation of the intelligent autonomous systems in human society depends on the trustworthiness of these systems. Unless these centralized and distributed autonomous systems are [10–18] *dependable* (*reliable, fault-tolerant, available, maintainable, safe* and *secure*), such systems may never see fruition in the commercial arena or be useful for critical operations. As an example of autonomous system, the disengagement data of self-driven cars can be mentioned. Autonomous vehicle disengagement data filed with the California Department of Motor Vehicles [19] for 2016 shows that a self-driving car *failed about every 3 hours due to hardware or software malfunction* as shown in Figure 1.1. Other examples abound [20–22]. The most recent Boeing incident [20, 21], was diagnosed to a malfunctioning sensor generating incorrect measurement data.

As can be observed from Figure 1.1, future autonomous systems will have to be designed to be extremely reliable for them to be assimilated into day to day living functions of human societies. This is because many such emerging applications can have life-threatening consequences if they malfunction in the field. While proponents of such systems argue that automation will make self-driven cars safer than human-driven vehicles, they ignore the fact that the controlling hardware and software itself may have bugs, that



Company	Autonomous miles	Disengagements	Rate per 1000 miles
Google	635868	124	0.20
Cruise	10015	284	28.36
Nissan	4099	28	6.83
Delphi	3125	178	56.95
Bosch	983	1442	1466.94
Mercedes	673	336	498.95
BMW	638	1	1.57
Ford	590	3	5.08
Tesla	550	182	330.91
Honda	0	0	0.00
VW	0	0	0.00

Figure 1.1: Autonomous vehicle disengagement data

sensors and actuators can degrade over time and that control principles will need to adapt in real-time to account for electrical and mechanical wear-and-tear.

For reliable and dependable vehicle operation of autonomous system, cross-layer (sensor, actuator and control) design methods need to be developed that deliver ultra-high levels of dependability (compared to the safety/dependability of aircraft and space travel). To make such autonomous systems dependable, errors in the sensors, actuators, electronics and control subsystems [13] of the autonomous system assembly will need to be continuously monitored. If any deviation from normal expected performance is detected, then corrective action must be taken to allow time for maintenance operations to be initiated on the failing device. For instance, *mechanical performance degradation* (such as due to increased friction from loss of lubrication in the ball bearings of a motor) must be compensated via *electrical means* (such as increased motor drive current). As another example, if a sensor produces incorrect readings, the condition must be detected and the control system for the device must be compensated for the lack of sensor performance. This has to happen on a per-vehicle basis to thousands of autonomous vehicles for the technology to ever become commercially successful on a massive scale. A single vehicle malfunction resulting in injury or loss of life can spell doom for the commercialization of such technology in the future. In this context, the control operations of modern autonomous systems are evolving towards electronics as opposed to mechanical/hydraulic means of control as evident from the proliferation of ‘steer-by-wire’ and ‘brake-by-wire’ control systems. While this improves the reliability of the system due to use of fewer mechanical components, the overall dependability of the system is now contingent on error-free operation of the control program, sensors and actuators. Detection and mitigation of such control malfunction in real-time forms the core objective of this research.

1.2 Prior Work

In this section, we formally define the different facets of the research landscape and present the challenges faced. We also discuss the merits and demerits of the state-of-the-art methodologies for the different areas.

Dealing with failures has been a major concern even with the earliest electronic systems [23, 24]. The use of checksums for failure tolerant matrix arithmetic and signal processing algorithms was first investigated by Hua, Jou and Abraham in [25, 26] and led to the field of *algorithm-based fault tolerance* (ABFT) in the context of complex signal processing algorithms. The underlying research focused on exact detection and correction of errors in classes of widely used signal processing algorithms such as for matrix arithmetic and the Fast Fourier Transform [27], among others. In [28], a methodology for soft error detection and correction called *algorithmic noise tolerance* (ANT) was developed and further explored in [29–31]. Here, the focus was on using reduced-precision duplication codes to detect and correct errors with the understanding that not all errors need to be detected and corrected precisely and that ‘perfect’ error compensation is not always required for various classes of signal processing algorithms such as for video compression. In [32, 33], the authors used the idea of *‘algorithmic soft error tolerance’* (ASET) which employs low-complexity estimators of a main DSP block to achieve reliable operation in the presence of soft errors in the form of rollback. Three distinct ASET techniques — spatial, temporal and spatio-temporal are presented. For frequency selective finite-impulse response (FIR) filtering. ANT schemes are also used to change the supply voltage dynamically to save energy and to match critical path delay in [34]. A class of codes called ‘real-number checksums’ for matrix-vector computations was developed by Nair and Abraham in [35]. This class of codes was used by Chatterjee and d’Abreu in [36] to perform error detection and correction in linear digital state variable systems. While error detection could be performed with low overhead in [36], error correction incurred large overheads both in area and time using the proposed algorithms. To resolve this, probabilistic and guided error compensation schemes were proposed in [37, 38]. In [39] and [40], continuous checksums, derived from the real number codes [35] were used for the first time to detect and correct faults in linear analog state variable systems. The state matrix was encoded using checksums and by tracking the system state variables, any change in the analog transfer function of the system could be detected in real-time. Trial and error based compensation learning methods combined with the use of less than minimum distance codes are used for failure tolerance for analog circuits are shown in [41]. Additionally, the idea of linear checksum was used for compensation of induced noise and transient errors at linear analog system [42].

Along a similar line, the *ANT* has been applied to detect various faults including transient errors and noise induced by signal coupling, electromagnetic interference and power supply/ground bounce in electronic circuits used for different applications. This is particularly important as these types of faults are difficult to simulate pre-silicon and very difficult to diagnose by current design verification algorithms that are designed mostly for digital timing validation and property checking [43, 44]. In prior research, methods targeting the detection and correction of catastrophic and parametric faults have been proposed. Many of these approaches include the use of redundancy (voting circuits, time averaging schemes, etc.) in the design and target the detection/mitigation of structural faults. In [45], a method for error detection in linear analog filters using state estimation was proposed and in [46] a checker for fully differential analog circuits was developed. In [47], the method was applied for checking linear analog circuits using DC signals. Of increasing concern, in this context, are nonlinear digital filters that are extensively and routinely used in signal processing, communication and control applications where reliability and dependability are critical issues. In such safety-critical applications, traditional methods for error detection and correction rely on hardware duplication/triplication. Error detection in non-linear systems generally involves partitioning of the circuit into linear and nonlinear components. Checksum codes are applied to the linear components for error detection whereas hardware and/or software redundancy is applied for error detection in the nonlinear modules. However, the high overhead in terms of power and area associated with these methods makes them relatively expensive in general applications. A more elegant error detection technique for non-linear systems based on *time-freeze linearization* was proposed in [48, 49] which models a nonlinear digital filter using a time-varying linearized representation. The checksum circuit generates a time-varying checksum code for each single time frame by *freezing* the system dynamics between two adjacent time frames and linearizing the circuit behavior between the two frames. However, the method can result in duplication of all the nonlinear functions in the worst case and further requires that the word-length precision of the checking circuitry be the same as that of the circuit under test (CUT). This incurs higher cost in terms of both complexity, area and power.

While prior works focused on concurrent error detection in different applications, it does not apply directly to linear or nonlinear autonomous systems, as these consist of *plant* and *controller* equations which must be combined together in a meaningful way to allow detection of perturbations in the plant as well as its controller. This necessitates the development of a general methodology for combining the plant and controller functions into a single state variable representation to which checksum codes can be applied. The second major difference is that while in [39, 45], it is possible to associate the state variables and their derivatives with specific nodes in the corresponding electrical circuits, the same is not necessarily true in generic control systems where sensors are used to monitor the system states but may not explicitly provide information about both the state values and their derivatives.

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) for wide varity of test cases [50–60]. These methods are generally compute-intensive and not suitable for applications with hard real-time constraints. The work of [61] 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 [62–66]. Particle filter based approaches have been emplpyed extensively to detect anomalies for different applications [67–72]. In [63], particle filtering and maximum likelihood methods are used to diagnose and correct sensor anomalies in autonomous ground vehicles. In [64], a sliding window observer is designed to predict future sensor measurements for error detection. In [65, 66], 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 [73]. Similar to [32], the idea of rollback to recover from error also was introduced in [74] at Boeing 737 simulator.

Of particular relevance to this research is prior work on the use of neuromorphic networks for anomaly detection and correction. In [75], 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 [76]. State estimation methods are also used for error detection in [77, 78]. In [78], 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 [79–82], 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 [83], 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 [81]. However, this does not address error correction and control reconfiguration, which was performed in [84, 85]. A hierarchical framework for fault propagation analysis was also addressed in [86].

With regard to *control adaptation* there has been significant research in the past [87–93]. In *gain scheduling* [87], 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 [87] 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 [87, 89], 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 [94], controller parameters were redesigned using absolute value of the encoded state of the system which was improved at [79] by utilizing the time dependent profile of the encoded state. In [95], 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. Similar work was also pursued by researchers in [96] and, by the imitation learning, in [97–100]. In [97], researchers have demonstrated an approach to control a vehicle from expert demonstrations of low-level controls and high-level commands. At training time, the commands resolve ambiguities in the perceptuomotor mapping, thus facilitating learning. At test time, the commands serve as a communication channel that can be used to direct the controller. In [98], the researchers present an end-to-end imitation learning system for agile, off-road autonomous driving using only low-cost on-board sensors. This method requires neither state estimation nor on-the-fly planning to navigate the vehicle. More research on health management, resiliency and fault tolerance of autonomous vehicles and systems are addressed in [101–103].

Recently L1 adaptive controllers [88] 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 [84] 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

[84] provided initial proof-of-concept ideas to motivate the present research. The present research is significantly detailed compared to [84].

There has also been significant body of work in the field of safe and robust operation of autonomous systems [104]. Researchers have looked at Unscented Kalman Filter [105] to project an uncertainty distribution ahead in time for vehicles on the road, use that to mathematically define collision risk, model risk beyond prediction horizon, and have considered sensing and communication losses. Stochastic model predictive controller [106] with a cost function based approach was used to have optimal control action when and where the vehicle changing lanes can satisfy a defined condition of 'safe' lane change. The uncertainty in the stochastic MPC cost is propagated by an Extended Kalman Filter. Stochastic model predictive controller was also pursued in [107] where an approach was demonstrated for driver-aware vehicle control based on stochastic model predictive control with learning (SMPCL). The framework combines the onboard learning of a Markov chain that represents the driver behavior, a scenario-based approach for stochastic optimization, and quadratic programming.

Researchers also worked on real-time filtering algorithms with a robotic vehicle to smooth emergent traffic waves [108]. They look at trade-offs in estimation accuracy to provide both distance and velocity estimates, with ground-truth hardware-in-the-loop tests with a robotic car.

1.3 Contributions of Dissertation

The key innovations of the proposed research are summarized as follows:

1. State encodings of linear and nonlinear control systems are utilized to generate error signature with low overhead that can monitor, and detect errors in a linear and nonlinear system operating under arbitrary failure mechanisms. Typically, a nonlinear system switches across different operating points around which the system behavior is weakly nonlinear [109]. Prior methods have focused on system operation around

fixed equilibrium points where the nonlinear behavior is limited and linearization techniques can be applied. The proposed research in this dissertation makes no such assumption and is applicable over a broad class of systems across all operating modes - i) linear, ii) weakly nonlinear and iii) strongly nonlinear.

2. State space check based approach is developed to diagnose parametric failures, and reconfiguration of controller based correction for different linear state variable system to enable real-time adaptability with low latency.

3. Hierarchical check based approach is developed to detect and diagnose the health of a nonlinear autonomous system decomposing the system down to individual subsystems while allowing multiple failures to occur simultaneously without compromising detectability.

4. State encoding based correction approaches are developed to correct sensor, actuator and control program failures for linear and nonlinear autonomous systems.

5. Dynamically adjusted encoded checking scheme is proposed, designed and demonstrated to detect, diagnose and correct failures in sensors, actuators, control execution core of a nonlinear autonomous systems.

6. Error detection and real-time compensation were demonstrated using machine learning model and state estimator assisted encoded checks on an actual quadcopter hardware.

1.4 Dissertation Overview

The primary objective of this dissertation is to develop methodologies, support algorithms and software-hardware infrastructure for detection, diagnosis, and correction of parametric failures for actuators, transient and slow-changing parametric errors for sensors and transient and permanent error at control program and transient and parametric error at actuators in linear and nonlinear state variable systems.

Chapter 2 discusses error detection and correction of switched capacitor circuits. Three different failure mechanisms are addressed in this work namely - parametric failure, alpha particle strike, and induced noise. Real-time noise cancellation is demonstrated for all of these mechanisms on linear analog filters.

Chapter 3 introduces the foundational theory of system level state-space encoding for detection, diagnosis and correction of parametric failures for multiple linear autonomous systems. The theory of error detection is developed, designed and implemented. The diagnosis of the failure was demonstrated with the help of machine learning algorithms. It is also shown how a linear encoding can be used for error correction with low latency. Simulation results on a linear system are presented to illustrate the detection and correction capabilities.

Chapter 4 extends the state encoding based error checking infrastructure to nonlinear state variable system. A properly learned machine learning model has been employed in hierarchical manner to diagnose the health of a nonlinear autonomous system decomposing the system down to individual subsystems, namely - sensors, actuators and control program execution. Error is detected and corrected in each of these subsystems. The proposed approach is implemented in a quadcopter hardware.

Chapter 5 looks at computationally inexpensive method to detect and correct failures for nonlinear autonomous systems where the encoded checking scheme dynamically adapt to the operating condition of the system by adjusting its threshold. The proposed approach has been applied to quadcopter and ‘Steer-by-wire’ automotive system in simulation. Additionally, the approach was validated in quadcopter hardware system.

Finally, chapter 6 presents the conclusions of this dissertation and recommends avenues for future research.

CHAPTER 2
PROBABILISTIC ERROR DETECTION AND CORRECTION IN SWITCHED CAPACITOR CIRCUITS USING CHECKSUM CODES

As technology scales to smaller dimensions and operating speeds of circuits increase, CMOS mixed-signal circuits and systems are becoming increasingly vulnerable to errors due to *parametric variations, external radiation (alpha particle strikes)* and induced circuit noise. Parametric variations are *static* in nature and result in permanent deviations in the values of passive components or behavioral parameters of active components (such as transistor gain). The effects of external radiation are, in comparison, *transient* in nature and result in dynamic changes in the voltage values of internal circuit nodes. In digital circuits, such voltage changes can result in logic values to be flipped resulting in *soft errors*. Such soft errors can occur randomly on any circuit node since the trajectory of an alpha-particle strike through the layers of a physical design cannot be predicted a-priori. Induced circuit noise also results in changes in the node voltage or branch current values within a circuit. However, its effects are localized to a small set of circuit elements since noise is coupled onto a victim circuit through specific design vulnerabilities. For example, crosstalk between two wires affects only the two wires concerned and no other circuit nodes. Hence, the sources of induced circuit noise can be localized to a subset of the complete set of nodes in a circuit. In this research, error detection and correction in a class of circuits called *switched-capacitor circuits* was focused for all three sources of errors discussed above. An interesting aspect of switched-capacitor circuits is that they combine the principles of operation of digital state machines (clocked circuits) along with those of analog circuits (continuous-time operation). Hence, *novel and elegant solutions for error detection and correction in switched-capacitor circuits can be found that combine aspects of digital error correction with analog error feedback to achieve analog error correction* *in ways that cannot be applied to continuous-time analog circuits.*

2.1 Key Contributions and Approach

2.1.1 Key Contributions:

Digital error detection and correction schemes are difficult to apply to continuous-time analog circuits because of the lack of a system clock. A key benefit of digital systems is that errors corrected before the onset of future clock cycles prevents them from propagating to other system states allowing for ease of correction [36]. On the other hand, error correction for analog circuits is performed by feeding back the detected error value with high negative gain to the corrupted circuit state variable [39]. *Both techniques require accurate error diagnosis requiring the use of complex circuitry as well as incur significant correction latency making them infeasible for use in practice. Worse yet, circuit noise can render the diagnosis incorrect where digital or analog division is used for diagnosis purposes [36, 39].*

To solve this problem, the use of *probabilistic correction algorithms* that have traditionally been applied to digital circuits which *do not require error diagnosis* and are very attractive from the viewpoint of hardware overhead and real-time, near-zero-latency correction. However, probabilistic correction algorithms are not amenable to high-gain error feedback as has traditionally been used for analog circuit error correction [39]. In this research [110], *for the first time, probabilistic correction algorithms [111, 112] was applied to switched-capacitor circuits that are clock-enabled but retain all the properties of continuous-time analog circuits in terms of input-output behavior.* Detected errors are corrected, to the maximum extent possible, before the onset of future clock systems using probabilistic correction techniques, without the need for high gain error value feedback [39]. No error diagnosis is necessary as claimed in [41, 113] and the circuit performs its normal functions automatically, in the absence of errors. In this work, different errors in system are only considered.

2.1.2 Approach

Checksum encodings are utilized by using circuit states and inputs to generate an error signal that detects presence of any fault, permanent or transient, indicating deviation from nominal performance. This checksum error signal is then fed back with appropriate gains to all system states, eliminating the need of expensive error diagnosis routines. The proper gain values are computed by performing pre-design statistical optimization experiments on the checksum error values over the assumed fault models. Such a concurrent error feedback to different states reduces the error power in a probabilistic sense, improving the overall SNR, without the need of error diagnosis and still achieving low-latency error compensation.

2.2 Switched-Capacitor Circuits: Overview

In switched capacitor circuits [114], resistances are emulated with capacitors with switch-based charge transfer mechanisms to the capacitor. These switches are operated with clocks that run at frequencies significantly higher than the circuit bandwidth. Such circuits are inherently resistant to process variability effects making them attractive to circuit designers. In theory, switched capacitor circuits function as analog sampled circuits where charge is stored in relevant capacitors for one phase of the clock cycle and in the next phase of the clock cycle, the same charge is distributed to other parts of the circuit.

2.2.1 Example 1

To show the efficacy of our proposed approach for error detection and probabilistic error correction, a biquadratic (also known as bi-quad) filter has been selected as the test circuit. The bi-quad circuit is very popular to implement second-order low-pass, band-pass, and high-pass filters. The system of bi-quad circuit is given in Figure 2.1, which is generally

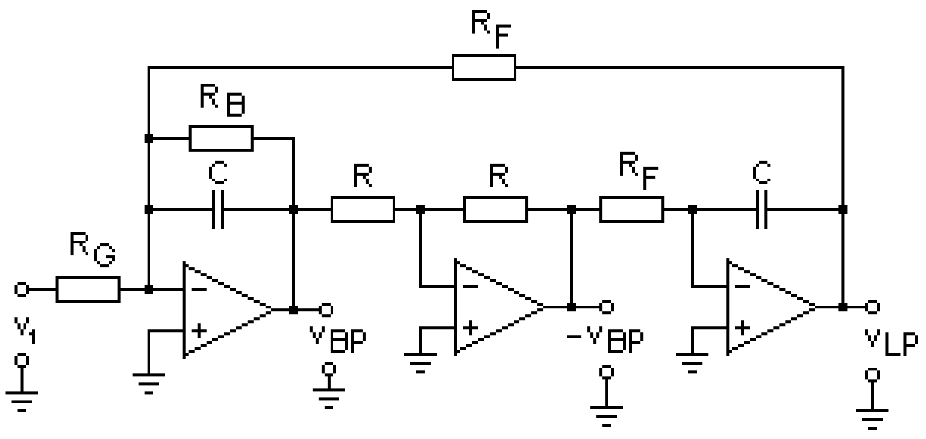


Figure 2.1: Biquadratic Circuit

modeled as [115]:

$$\dot{x} = \begin{bmatrix} 0 & a_{12} \\ a_{21} & a_{22} \end{bmatrix} x + \begin{bmatrix} 0 \\ b_2 \end{bmatrix} u \tag{2.1}$$

Where, $a_{12} = \frac{1}{RC}, a_{21} = -\frac{1}{R_F C}$, $a_{22} = -\frac{1}{R_B C}, b_2 = \frac{1}{R_G C}$ and $x = [v_{LP}\ v_{BP}]^T$, $u = v_{in}$, v_{LP} and v_{BP} being the low-pass and band-pass outputs, respectively with v_{in} as the input signal and $v_{LP} (= x_1)$ is the system response.

The model represented in 2.1 can also be expressed as a transfer function in the following way:

$$H(s) = -\frac{\alpha_6 s^2 + \dfrac{s\alpha_3\alpha_5}{T} + \dfrac{\alpha_1\alpha_5}{T^2}}{s^2 + \dfrac{s\alpha_4\alpha_5}{T} + \dfrac{\alpha_2\alpha_5}{T^2}} \tag{2.2}$$

where, $\alpha_{i|i=1,\cdots,6}$ are design parameters, and T is the clock period that is used to design its switched capacitor counter-part. The switched capacitor realization of the aforementioned biquad circuit is shown in Figure 2.2 as depicted in [116]. In this circuit, the resistors are replaced by switched capacitors. By replacing $s = \frac{2}{T}\frac{z-1}{z+1}$, in 2.2, the transfer function of

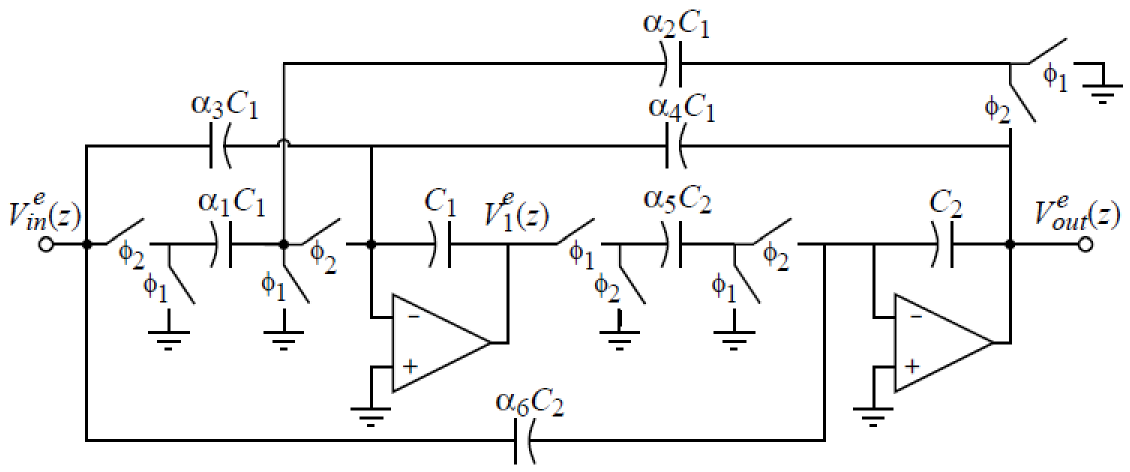


Figure 2.2: Switched capacitor realization of a biquadratic circuit

the switched-capacitor representation is obtained as follows:

$$H(z) = -\frac{\alpha_6 z^2 + (\alpha_3\alpha_5 - \alpha_1\alpha_5 - 2\alpha_6)z + (\alpha_6 - \alpha_3\alpha_5)}{z^2 + (\alpha_4\alpha_5 + \alpha_2\alpha_5 - 2)z + (1 - \alpha_4\alpha_5)} \tag{2.3}$$

The z-domain transfer function of 2.3 is translated into the appropriate state-space representation as follows:

$$\begin{aligned} x[k+1] &= \begin{bmatrix} -a_1 & 1 \\ -a_0 & 0 \end{bmatrix} x[k] + \begin{bmatrix} -b_1 \\ b_0 \end{bmatrix} u[k] \\ y[k] &= \begin{bmatrix} 1 & 0 \end{bmatrix} x[k] + [0]\, u[k] \end{aligned} \tag{2.4}$$

where, $a_1 = \alpha_4\alpha_5 + \alpha_2\alpha_5 - 2$, $a_0 = 1 - \alpha_4\alpha_5$, $b_1 = \alpha_3\alpha_5 - \alpha_1\alpha_5 - 2\alpha_6$, and $b_0 = \alpha_6 - \alpha_3\alpha_5$. By this way, the system can be generally represented as the discrete-time state-space form of $x[k+1] = Ax[k] + Bu[k]$, and a checksum circuit can be constructed from this representation.

2.3 Proposed methodology on Error detection and probabilistic Error correction: A checksum based approach

2.3.1 Error Detection Scheme

The general representation of a linear digital state variable system is given as follows:

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] \\ y[k] &= Cx[k] + Du[k] \end{aligned} \tag{2.5}$$

Here, $x[k] = \left[x_1[k],\ x_2[k],\ ...,\ x_n[k]\right]^T$ is a vector of system state variables, $y[k] = \left[y_1[k],\ y_2[k],\ ...,\ y_m[k]\right]$ is a vector representing the system outputs and $u[k] = \left[u_1[k],\ u_2[k],\ ...,\ u_p[k]\right]^T$ are the inputs to the system. The dimensions of the matrices A, B, C and D are of size $n \times n$, $n \times p$, $m \times n$ and $m \times p$ respectively. The next sample of the j^{th} state variable, $x_j[k+1]$ is expressed as a weighted sum of the state variables $x_j[k], 1 \leq j \leq n$ and inputs $u_j[k], 1 \leq j \leq p$. Similarly the outputs $y_i[k]$ for $1 \leq i \leq m$ are expressed as a weighted sum of $x_j[k], 1 \leq j \leq n$ and inputs $u_j[k], 1 \leq j \leq p$.

The generalized theory of analog checksum has been introduced in [36], which is reviewed here for the sake of continuity. The signal flow graph of a switched-capacitor implementation of a state variable system has been shown in Figure 2.3 as represented by 2.4 and 2.5 where $x[k]$'s are the states of the system and $u[k]$ is the input variable. With this mathematical setup, following theorem is proposed:

Theorem 2.3.1. *If a linear state variable system is represented by A and B, and if there is a row vector CV. Then the quantity $e[k] = c[k] - CV.x[k]$ will produce non-zero value when there is a deviation in system model A and B, where $c[k] = M.x[k] + N.u[k]$, $M = CV.A$ and $N = CV.B$.*

Proof. The plant of a linear state variable system follows the model shown in 2.5, M and N are computed as $M = CV.A$ and $N = CV.B$, which is saved in the checking computer. If the system is in nominal condition, the matrices A and B does not change. In this situation:

$$x[k+1] = Ax[k] + Bu[k] \tag{2.6}$$

$$\implies CV.x[k+1] = CV.Ax[k] + CV.Bu[k] \tag{2.7}$$

$$\implies e[k] = CV.Ax[k] + CV.Bu[k] - CV.x[k+1] = 0 \tag{2.8}$$

i.e. the quantity, 'chekcksum error', $e(k)$, stored in the checking computer, produces a zero valued signal.

Now, lets assume that the plant matrices A and B are going through some deviation, and the new values are $A'(= A+\delta)$ and $B'(= B+\gamma)$. Here δ and γ are matrices with appropriate dimensions having one row i containing non-zero elements, and zero otherwise. Now, the checksum error computation becomes:

$$e[k] = CV.Ax[k] + CV.Bu[k] - CV.x[k+1] \tag{2.9}$$

$$= CV.Ax[k] + CV.Bu[k] - (CV.A'x[k] + CV.B'u[k]) \tag{2.10}$$

$$= CV.Ax[k] + CV.Bu[k] - (CV.(A+\delta)x[k] + CV.(B+\gamma)u[k]) \tag{2.11}$$

$$= -CV.\delta x[k] - CV.\gamma u[k] \tag{2.12}$$

$$\tag{2.13}$$

From this analysis, it is proved that the quantity, 'chekcksum error', $e(k)$, will produce non-zero signal, and hence, detect failure when there is a deviation in system model A, and B. $\square$

As seen from the Theorem 2.3.1, the new variable *checksum error*, $e[k] = c[k] - CV.\ x[k]$ was defined that generates a value less than a predefined threshold for nominal system operation. This predefined value depends on measurement noise statistics and modeling inaccuracy and chosen accordingly from pre-deployment simulations and post-deployment calibrations. If the error signal $e[k]$ generates a value greater than the chosen threshold, an error is detected.

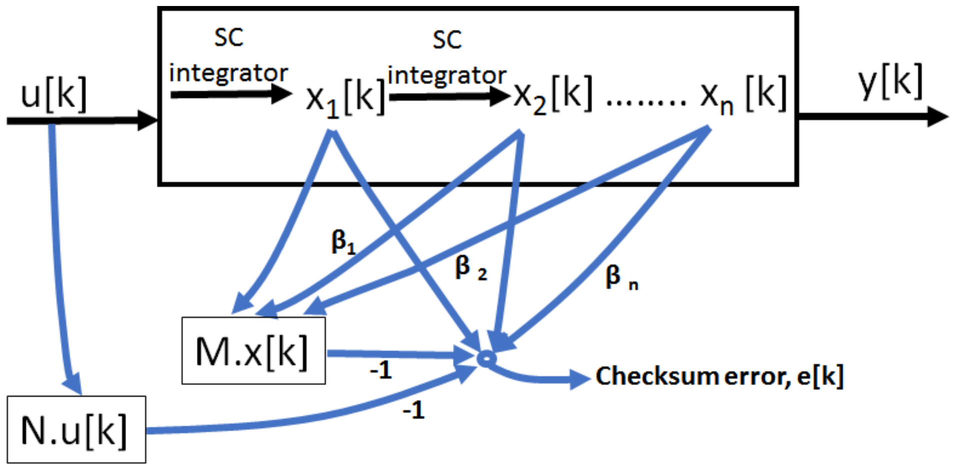


Figure 2.3: Error detection signal flow graph

Example

To implement checksum circuit in the example system introduced in Section 2.2, A *coding vector*, $CV = [\beta_1, \beta_2]$ was selected. Then the following was obtained as the expressions of $M = CV.A$, and $N = CV.B$:

$$\begin{aligned} M &= \begin{bmatrix} (-\beta_1 a_1 - \beta_2 a_0) & \beta_1 \end{bmatrix} \\ N &= \begin{bmatrix} (-\beta_1 b_1 - \beta_2 b) \end{bmatrix} \end{aligned} \tag{2.14}$$

The *data checksum*, $c[k]$ is computed as $c[k] = M.\,x[k] + N.\,u[k]$, and *checksum error*, $e[k]$ becomes $e[k] = CV.\,x[k] - M.\,x[k] - N.\,u[k]$. This shows how the *checksum error* is computed in the example circuit. In practical implementations, this is always constructed using only two op-amps with necessary resistors and capacitors and the hardware overhead is independent of the number of system states.

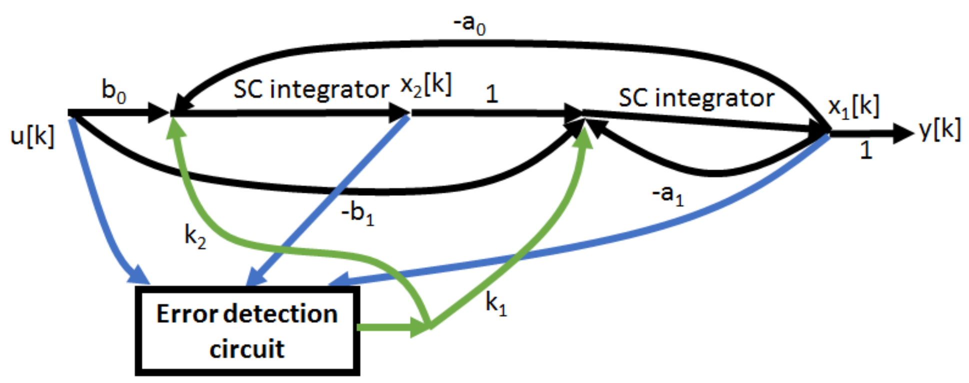


Figure 2.4: Error correction signal flow graph

2.3.2 Error correction scheme

The proposed methodology for error correction is illustrated in the signal flow graph of Figure 2.4. Following theorem is proposed for error correction scheme:

Theorem 2.3.2. *To correct the failure, the 'checksum error', $e[k] = -CV.\delta x[k] - CV.\gamma u[k]$ should fed back to the state with high gain.*

Proof. Consider the system referred in Theorem 2.3.1. In presence of failure, the system shown in 2.5 will change according to following:

$$\tilde{x}[k+1] = (A+\delta)\tilde{x}[k] + (B+\gamma)u[k] \tag{2.15}$$

And, $\tilde{x}[k]$ corresponds to the modified state which the system having parametric failure exhibits. It is clear from this representation that the error in state x is given as

$$\Delta x[k] = \delta x[k] + \gamma u[k] \tag{2.16}$$

Thus an accurate compensation scheme will exactly compensate for this error of the state x by providing $-\Delta x$ as feedback into state $x_i[k]$ with a gain ρ. In this situation, the expression of checksum error becomes the following:

$$e[k] = CV.Ax[k] + CV.Bu[k] - CV.x[k+1] \tag{2.17}$$

$$= CV.Ax[k] + CV.Bu[k] - (CV.A'x[k] + CV.B'u[k] + \rho CV.e[k]) \tag{2.18}$$

$$= CV.Ax[k] + CV.Bu[k] - (CV.(A+\delta)x[k] + CV.(B+\gamma)u[k] + \rho CV.e[k]) \tag{2.19}$$

Now, as the failure is in ith row, elements in all other element will get cancelled. As ith element of CV is β_i, following is obtained after simplification:

$$e[k] = -\beta_i.\delta x[k] - \beta_i.\gamma u[k] - \rho\beta_i.e[k] \tag{2.20}$$

Which implies:

$$(1+\rho\beta_i)e[k] = -\beta_i.\delta x[k] - \beta_i.\gamma u[k] \tag{2.21}$$

$$\implies \rho e[k] = -\delta x[k] - \gamma u[k] \tag{2.22}$$

when $\rho \longrightarrow \infty$. It is also observed that the quantity *checksum error* approaches zero when gain ρ is sufficiently large and it is independent of coding vector CV. i.e. $\rho e = -\delta x[k] - \gamma u[k] = -\Delta x$

Now, when this signal is fed back to the state x_i, the expression of modified state becomes:

$$\tilde{x} + \rho e = x + \Delta x + \rho e = x + \Delta x - \Delta x = x \tag{2.23}$$

Thus, it can be seen that the compensation signal $\rho e[k]$ fed back into the state x_i is exactly equal and opposite to the error in the state x_i and by this way, the compensation is performed. $\square$

It should be noted, the checksum error can also be constructed by $e[k] = CV.x[k+1] - CV.Ax[k] - CV.Bu[k]$. This expression will be able to detect the failure in real-time, however, it should be phase inverted to for correction. This definition will be used for the to explain the remaining part of this research. To correct the error caused by any failure mechanism, the phase-inverted *checksum error* signal $e[k]$ is fed back to the states of the switched capacitor circuit with appropriate gains k_1 and k_2. Pre-deployment design-stage statistical experiments are performed on different instances of assumed fault models where the gain values are varied over possible range of choices and error power is measured. Such Pareto-surface based optimization provides us with the knowledge of the disparate effects of different faults on the *checksum error* signal $e[k]$ along with optimal gain choices for best error correction. In post-deployment operation of the circuit, the temporal profile of the error signal $e[k]$ as well as the magnitude of the signal provide an indication of the type of fault. A transient alpha-particle induced soft error causes a temporary glitch of low magnitude on the error signal line whereas persistent fault effects such as coupled noise sources or parametric shifts due to process variations generate a permanent error signal. For transitory faults, the error signal is fed back for a short duration after observing the non-zero error signal whereas the feedback is performed over longer periods of time for persistent error sources. A decision-making system observes the error signal and chooses the appropriate pre-computed gain values from look-up table based memory (which is available in modern systems) and performs the pertinent temporal feedback to the system states, resulting in significant SNR improvements as reported in the following section.

Implementation Example

The implementation of the error correction scheme in the bi-quad circuit is illustrated in Figure 2.5. The states $X_1(z)$ and $X_2(z)$ and the input $u(z)$ are used in the 'error detection circuit' to generate the *checksum error* signal. This signal is then fed back through appropriate gains to states by capacitive coupling of the signal to the corresponding op-amp

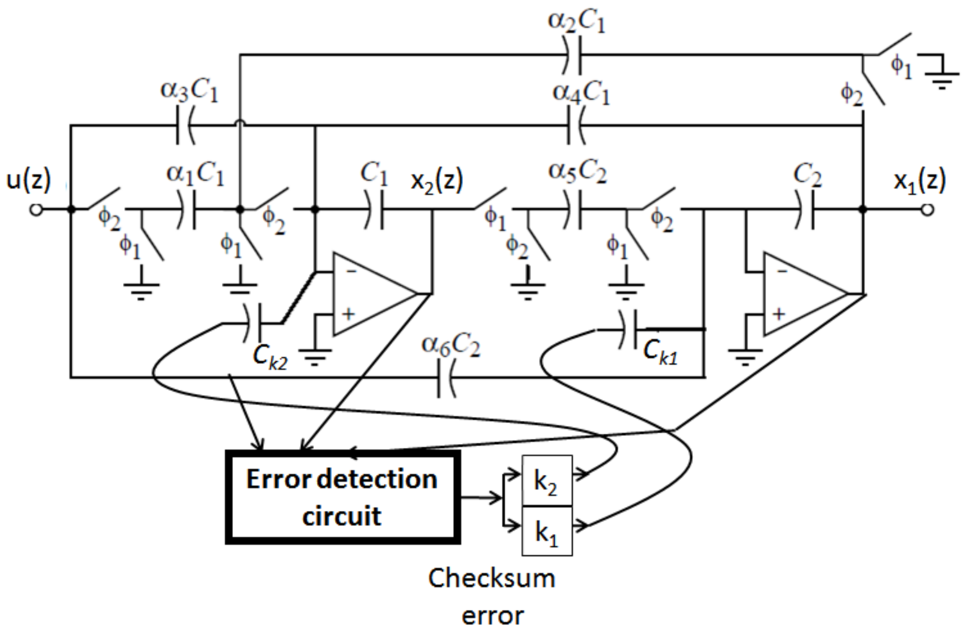


Figure 2.5: Error correction scheme for biquadratic circuit

inputs. During fault-free operation of the circuit, the error signal lies below the pre-defined threshold and the gain values are chosen to be zero such that there is no feedback of error signal. Upon observance of the error signal above the threshold, the feedback is performed by a separate decision-making system (can be implemented by either software or comparator-based techniques) as mentioned above.

2.4 Real time error correction: experimental results

To demonstrate the viability of the proposed method, simulation experiments on the biquad filter were performed where three different kinds of faults were applied. The results are explained below:

2.4.1 Parametric faults

Parametric faults have been considered as the first fault model. Here, the parameters (capacitance values) of the filter circuit shown in Figure 2.2 are varied as time progresses, which results in non-zero *checksum error*. In the simulation experiments, this *checksum*

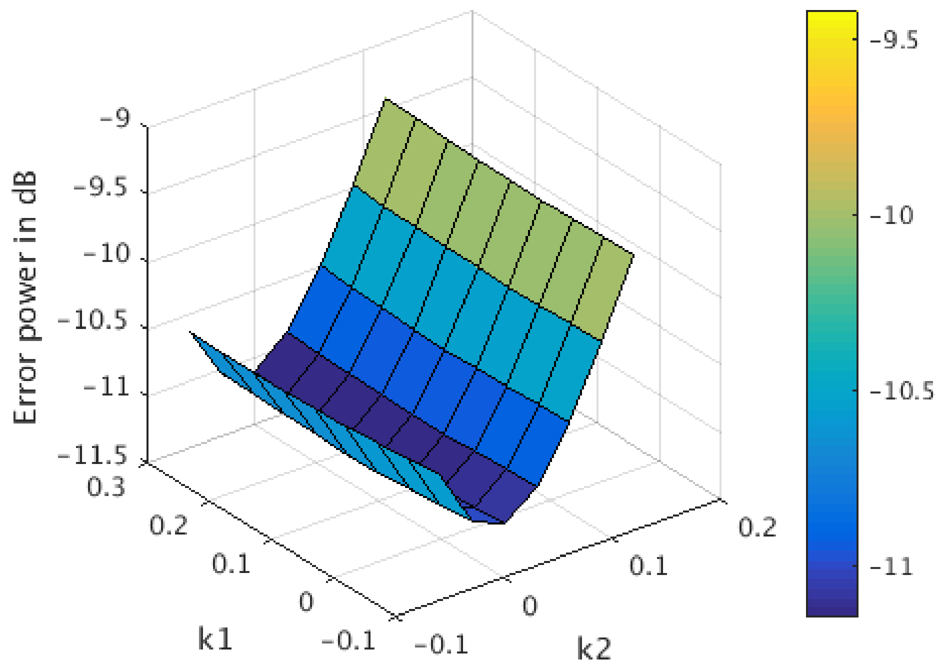


Figure 2.6: Pareto surface for parametric failure

error is fed back to the states with different gain value choices and the error power is computed for each case. This experiment is repeated by varying all the capacitors in the filter. It must be noted here that the faults being considered in this fault model are permanent in nature. This necessitates a permanent feedback of the *checksum error* ensuring continual correction of the faults. By selecting different combination of feedback gains from *checksum error* to different states, different error power values were obtained which is shown in the form of a Pareto surface in Figure 2.6. Here, k_1 and k_2 represent gain values from *checksum error* to x_1, and x_2 respectively that allows error mitigation while keeping the circuit operation stable. From the convex contours of the Pareto surface, it can be seen that by selecting appropriate gain values $k_{1,opt}$, and $k_{2,opt}$, the error power can be minimized. This shows the viability of the proposed approach for correcting the parametric faults in switched capacitor circuits.

2.4.2 Alpha-particle strike

The faults due to alpha-particle strikes have been considered as the second type of faults in this work. An alpha-particle strike introduces an ionized path between different nodes of the electronic system which changes the voltage level at different nodes and hampers the operation of the electronic circuit. To model such a phenomenon in this test case, soft errors are introduced in different nodes of the circuit by a fast temporary discharge of circuit nodes. This creates a temporary error in the circuit operation as shown in Figure 2.7. Due to transitory nature of these errors, it is sufficient to feed the *checksum error* only once to nullify the effects of the temporary error. This is accomplished by feeding the *checksum error* back to the states at next phase of the clock cycle with different gains. Like the previous case, a Pareto surface of error power was obtained by selecting different gain values k_1 and k_2 within stability boundary as shown in Figure 2.8. From this surface plot, the optimum gain $k_{1,opt}$, and $k_{2,opt}$ were selected to minimize the error power introduced by alpha particle strike.

In Figure 2.7, the temporal output response of the bi-quad filter is plotted where, an alpha-particle strike has happened in the circuit at $t = 300\mu s$, and by feeding back the *checksum error* to the states with optimum gains, the mean of the system response was restored in that instant. The zoomed-in image of the inset portrays the error mitigation effects of the proposed correction scheme. In the absence of any error correction, the circuit slowly recovers from the soft error towards the nominal performance as seen in the inset of the faulty response. The proposed error scheme is able to correct the system response instantaneously with near-zero latency. This again demonstrates the effectiveness of the proposed scheme in correcting different fault effects in a probabilistic sense by appropriate feedback of the *checksum error* to different system states.

2.4.3 Induced noise

As a third fault model, the performance of the proposed approach was explored in presence of induced noise sources such as coupled noise from digital substrates or nearby conductors, power supply switching, etc. In this study, an additional persistent noise source was coupled in one of the states of the switched capacitor circuit and the output of *checksum error* was observed. Similar to previous experiments for the other fault models, the gain values k_1 and k_2 are varied and the *checksum error* is fed back to the states in continuous manner for the duration the noise source is active. The Pareto surface obtained by varying the values of k_1 and k_2 within the stability limit of circuit performance is given Figure 2.9. As observed from this surface plot, the error power reduces as an optimum set of gain values was applied. This observation also demonstrates the viability of the proposed approach to correct induced noise faults.

2.4.4 Experimental Data

The experimental data for 2 different instances of the assumed fault models in the test circuit have been tabulated in Table 2.1. The error power before and after correction for the different fault models clearly demonstrate the SNR improvement performed by the correction mechanism. It is noteworthy that this noise power reduction has been achieved without any diagnosis mechanisms[113].

Table 2.1: Error correction comparison

Failure mechanisms	Error power (dB) without correction	Error power (dB) with correction
Parametric, fault at $\alpha_1 C_1$	-34.68	-37.81
Parametric, fault at $\alpha_3 C_1$	-62.07	-74.47
Alpha particle strike, fault at $\alpha_3 C_1$	-62.07	-74.47
Alpha particle strike, fault at $\alpha_4 C_1$	-55.22	-79.35
Induced noise at state x_1	-70.33	-80.71
Induced noise at state x_1	-86.60	-92.50

2.5 Summary

A real time approach to detect and correct errors in switched capacitor circuits has been demonstrated in this work. This has been accomplished by taking advantage of the clock separation which is an inherent property of switched capacitor circuits. System level checksum encodings have been exploited to detect any deviation from nominal system performance. This error signal is then fed back with appropriate gain values to all system states without performing any dedicated diagnosis routines. Three different fault models have been considered and the proposed approach has been able to correct faults in all of this situations with very high degree of accuracy minimizing the error power. The experimental data represent the effectiveness of the proposed approach. Future work will involve demonstration of error correction on other complex linear and nonlinear electronic systems operating under arbitrary failure mechanisms.

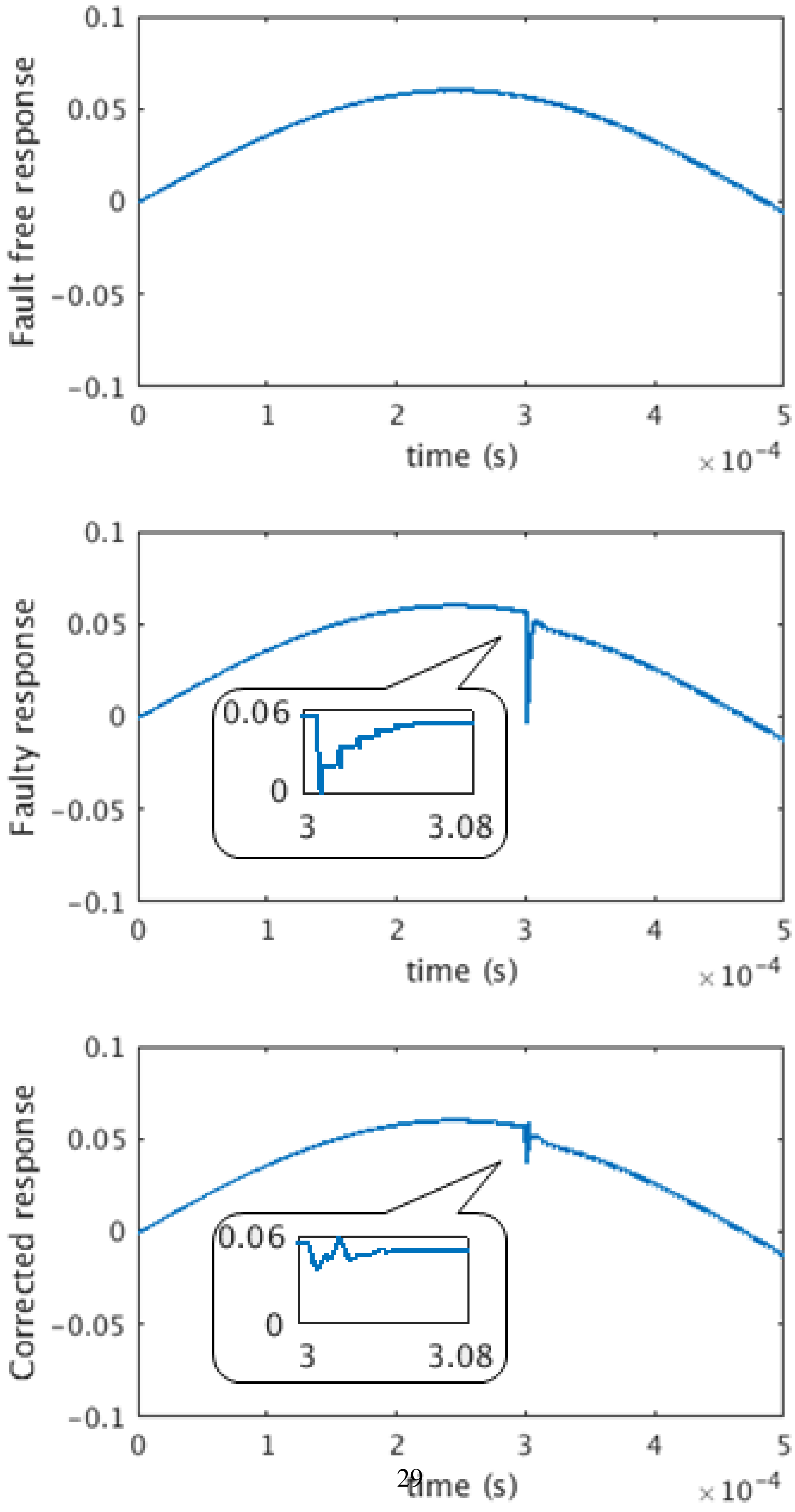


Figure 2.7: Response corrected by checksum error feedback

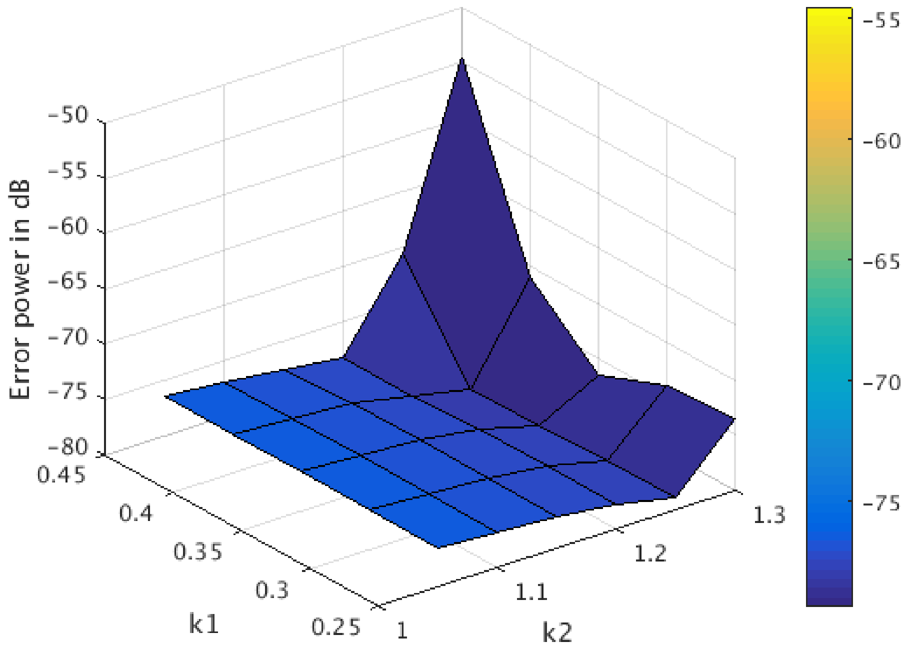


Figure 2.8: Pareto surface for alpha-particle strike failure

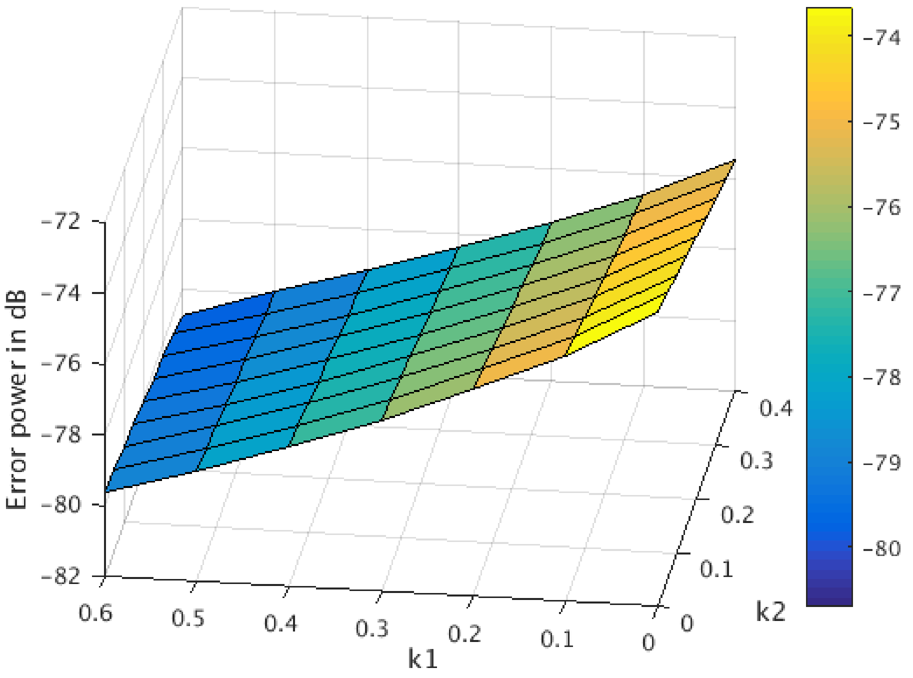


Figure 2.9: Pareto surface for induced noise failure

CHAPTER 3
ON-LINE DETECTION, DIAGNOSIS, AND COMPENSATION FOR FAILURES IN LINEAR STATE VARIABLE CIRCUITS AND SYSTEMS USING TIME-DOMAIN CHECKSUM OBSERVERS

3.1 Real-Time Error Detection and Control Compensation From Steady State Checksum

3.1.1 Introduction and Key Contribution

While the last two decades witnessed revolutions in computing and communications, the next decade will witness a revolution in robotics. From self-driven cars to robot-assisted surgery, there will be a plethora of applications that will depend on the reliable operation of sensors and actuators at the various man-machine interfaces of future autonomous systems. There is wide variety of application of DC motor in real life. Although its operation is well understand, real time detection, diagnosis and correction of different electro-mechanical faults in an electro-mechanical system like a DC motor is yet to be looked at. In this section, real-time detection of mechanical anomalies, namely co-efficient of viscous friction, torque constant etc were investigated that arise in the operation of a simple actuator due to field wear-out of a DC motor. Purely electrical tests are used for anomaly detection and once an error is detected, the motor control algorithm is adjusted almost instantaneously to compensate for loss of motor performance. The idea is to allow the motor to run uninterrupted until such time that more detailed (off-line) tests can be carried out to diagnose the problem and create a permanent fix for it.

A device such as a brush-activated DC motor can be represented as a linear state variable system, traditionally referred to as the *plant*, in control theoretic terminology. The plant is accompanied by a *controller* (proportional or PID [10, 18]) which measures specific plant parameters (such as input current and angular velocity) to maintain the performance of the plant under changing load conditions. In the past, there has been work on detection of faults using plant models derived from off-line and real-time experiments [117–121]. In each of these research papers, the plant model is estimated using off-line experiments (plant parameter measurements). Subsequently, in real-time, the plant and its model are both stimulated with the same input and the plant output is compared against the model output. Any discrepancies between the two are flagged as faults. However, failure diagnosis is not real-time and therefore error compensation cannot be performed in real-time either.

Real-time analog checksums [94] was proposed to perform real-time error detection and compensation. The key contributions of this research are as follows:

1. Use of checksum codes for encoding the state matrix of the DC motor in such a way that allows real-time diagnosis of the state variable of the motor that is in error. This allows detection/diagnosis of mechanical effects (e.g. friction) using electrical measurements.

2. Development of a checksum error driven performance compensation algorithm that dynamically modifies the motor control parameters to recover motor performance under diverse electro-mechanical field degradation conditions in near real-time until the motor can be powered off for regular maintenance.

3.1.2 DC Motor Control: An Overview

Two popular methods of speed control in a DC motor are armature control (controlling the torque by varying armature current) and flux control (controlling the field current). The approach of armature control results in a linear state space representation, and was followed in this work. The DC motor is modeled by the armature resistance R_a, self-inductance of the armature L_a, back emf constant K_b, and torque constant K_e. Under ideal condition, the electrical torque T_e developed at the shaft is directly proportional to armature current i_a and the induced emf v is directly proportional to speed ω.

$$\begin{aligned} T_e &= K_e i_a \\ v &= K_b \omega \end{aligned} \tag{3.1}$$

The speed of the motor can be controlled by applying a DC voltage across its armature terminals. For a proportional control system the terminal voltage V is equal to $K(\omega_{ref} - \omega)$ where, K is the proportionality constant. K_e and K_b are numerically equal in magnitude if K_b is expressed in the unit of volt-s/rad. The terminal relationship assuming $K_e = K_b$ is given by the following equation:

$$\begin{aligned} V &= K_b\omega + i_a R_a + L_a \frac{dt_a}{dt} \\ \implies i_a &= -\frac{R_a}{L_a}\int i_a dt - \frac{K_b}{L_a}\int \omega dt + \frac{1}{L_a}\int V dt \\ \implies i_a &= -\frac{R_a}{L_a}\int i_a dt - \frac{K_b + K}{L_a}\int \omega dt + \frac{K}{L_a}\int \omega_{ref} dt \end{aligned} \tag{3.2}$$

The mechanical shaft driven by the motor is modelled as a moment of inertia J for the sake of simplicity with a co-efficient of viscous friction B_1. Then, the torque equation which drives the load is given by,

$$\begin{aligned} J\frac{d\omega}{dt} &= K_b i_a - B_1\omega - T_L \\ \implies \omega &= \frac{K_b}{J}\int i_a dt - \frac{B_1}{J}\int \omega dt - \frac{1}{J}\int T_L dt \end{aligned} \tag{3.3}$$

Combining 3.2 and 3.3, following was obtained as the state space model of a DC motor speed control system. Here, $x(t)$ represents $[i_a, \omega]^T$ and control action $u(t)$ represents

$[\omega_{ref},\, T_L]^T$.

$$\begin{bmatrix} i_a \\ \omega \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \int \begin{bmatrix} i_a \\ \omega \end{bmatrix} dt + \begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} \int \begin{bmatrix} \omega_{ref} \\ T_L \end{bmatrix} dt \tag{3.4}$$

Where, $a_{11} = -\frac{R_a}{L_a}, a_{12} = -\frac{K_b+K}{L_a}, a_{21} = \frac{K_b}{J}, a_{22} = -\frac{B_1}{J}, b_{11} = \frac{K}{L_a}$ and $b_{22} = -\frac{1}{J}$

Similar to linear digital state variable system expressed in 2.5, the linear analog state variable system takes the following form:

$$\begin{aligned} \dot{x} &= Ax(t) + Bu(t) \\ y &= Cx(t) + Du(t) \end{aligned} \tag{3.5}$$

The state space representation is rewritten in the integral form to eliminate noise sensitivity as,

$$x = A \int x(t)dt + B \int u(t)dt \tag{3.6}$$

3.1.3 Proposed Methodology: Error Detection

Similar to linear digital state variable system, a row vector namely '*coding vector*', CV is introduced where $CV = [\alpha_1\ \alpha_2 \cdots \alpha_n]$. Additionally, two new matrices $M = CV.A$ and $N = CV.B$ a quantity '*data checksum*', $c(t) = M\int x(t)dt + N\int u(t)dt$ were introduced. From this representation it can be shown that, in a fault free system, c(t) = CV.x(t) where, c(t) is called as '*state checksum*'. The 'Checksum error', $e(t)$ is defined as $e(t) = c(t) - CV.x(t)$ which will produce non-zero output when there is a fault. By this way, the error in the system can be detected. To make this system stable, a small value of this error is fed back to the data checksum circuitry. The feedback gain should be small enough to keep the system dynamics unchanged. In case of DC motor system, the following is obtained as the expression of M and N:

$$M = \begin{bmatrix} \alpha_1 a_{11} + \alpha_2 a_{21} & \alpha_1 a_{12} + \alpha_2 a_{22} \end{bmatrix}$$
$$N = \begin{bmatrix} \alpha_1 b_{11} & \alpha_2 b_{22} \end{bmatrix} \tag{3.7}$$

And, the expression of 'checksum error' becomes: $e(t) = M \int_0^t x(t) + N \int_0^t u(t) - CV.x(t)$. In presence of parametric faults in the system, this error signal produces non-zero signal output. The state transition graph of the DC motor along with the proportional controller and the analog checksum circuitry is shown in Figure 3.1. To keep this figure simple, the variables $i_a, \omega, \omega_{ref}$ and T_L are shown separately.

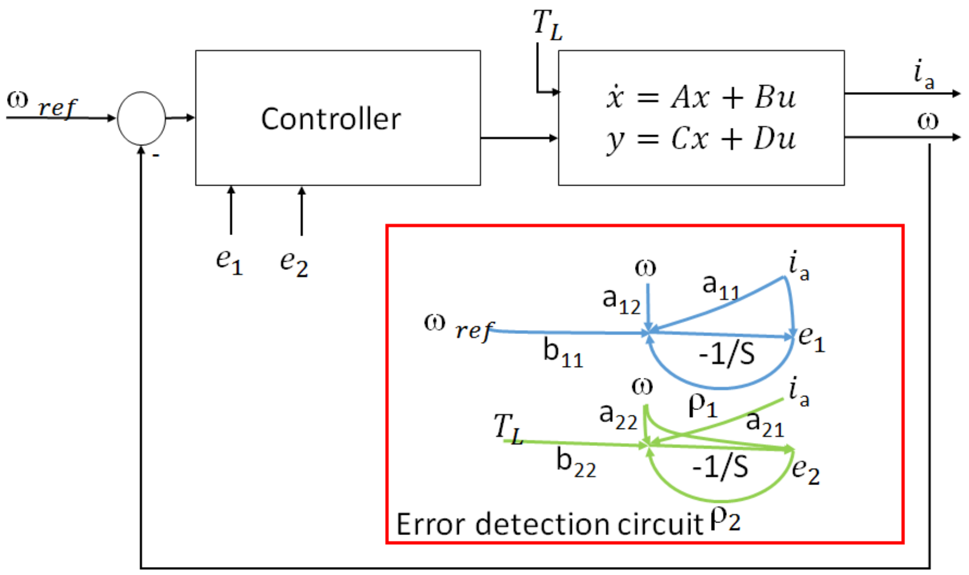


Figure 3.1: State transition graph showing the motor, controller and the error detection circuit

For error compensation, in this work, two checksums e_1 and e_2 were employed, with coding vectors of $[1,\ 0]$ and $[0,\ 1]$ respectively such that e_1 and e_2 check for explicit errors in the first and second equations of the state space representation in 3.4. A fault in the first state equation is uniquely identified by the signal $e_1(t)$ and similarly $e_2(t)$ detects errors in the second state equation.

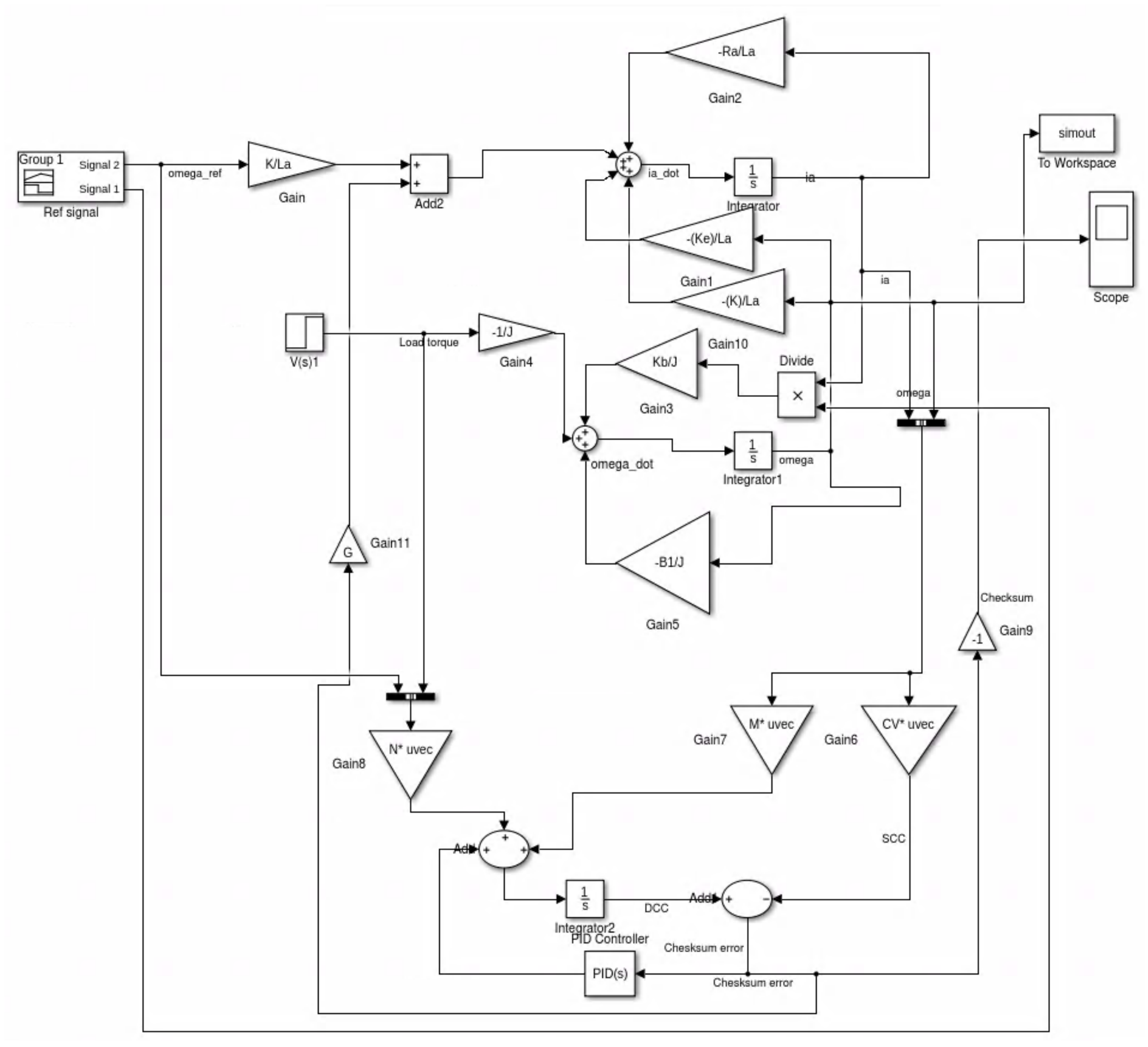


Figure 3.2: Simulink model of the DC motor

3.1.4 Error Detection Result

Simulation

The system was developed in Simulink and is shown in Figure 3.2. Here, the system was simulated along with the error detection circuit. The faults injected in the system are in the form of different parametric variations in coefficient of viscous friction B_1, back-emf constant K_b, and armature resistance R_a. Power supply transients are modelled by the change in motor torque constant K_e during simulation. Injection of faults changes the state-space representation of the system and generates a non-zero checksum error signal

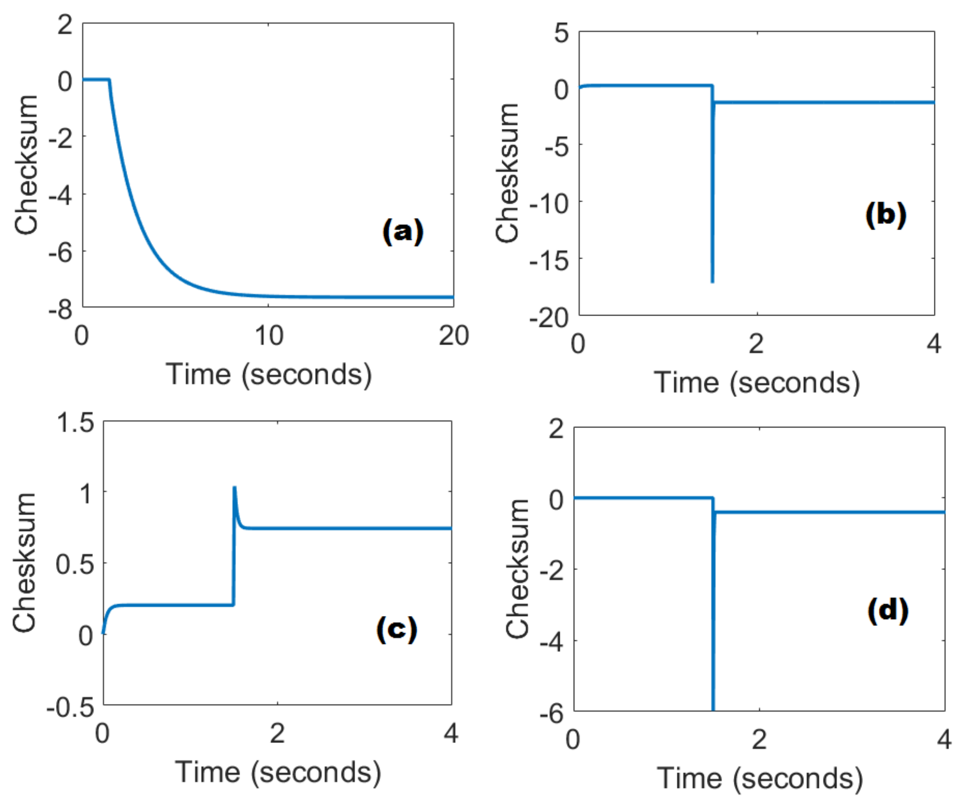


Figure 3.3: Checksum error signal with permanent perturbations in: (a) armature resistance, (b) back-emf constant, (c) viscous friction coefficient and (d) torque constant

$e(t)$ indicating the presence of parametric perturbations in the system. The checksum error signal is shown in the presence of different fault injection conditions in Figure 3.3.

Hardware Validation of the Error Detection Circuit

An experimental test-bench has been built using commercial off-the-shelf components and DC motor. Figure 3.4 shows a picture of the constructed test bench in stripboard using operational amplifiers, passive components and a DC motor. The overhead of the error detection scheme is low and is $O(1/n)$, where n is the number of states in the system. Figure 3.4 shows the experimental hardware prototype.

Figures 3.5 and 3.6 show the analog checksum signal as recorded in an oscilloscope in presence of a blocked rotor (modelling temporary change in viscous friction coefficient B_1), and power supply transient (modelling temporary change in torque constant) respectively.

It is clearly seen that a non-zero signal output indicates the presence of a transient fault in the system. The experimental results match very closely with the simulation.

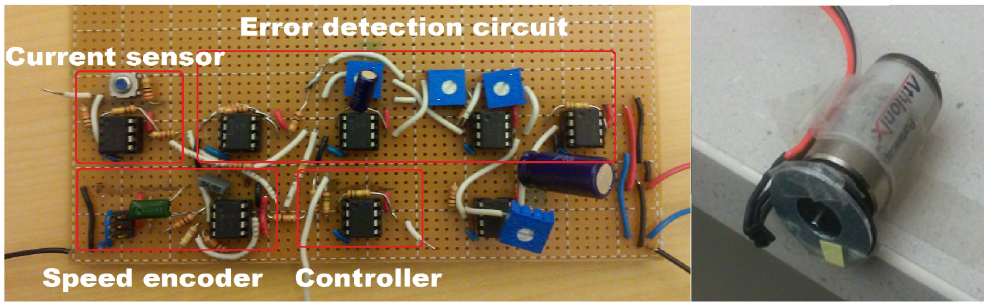


Figure 3.4: Experimental prototype showing the DC motor, speed encoder, error detection circuit, current sensor and the controller

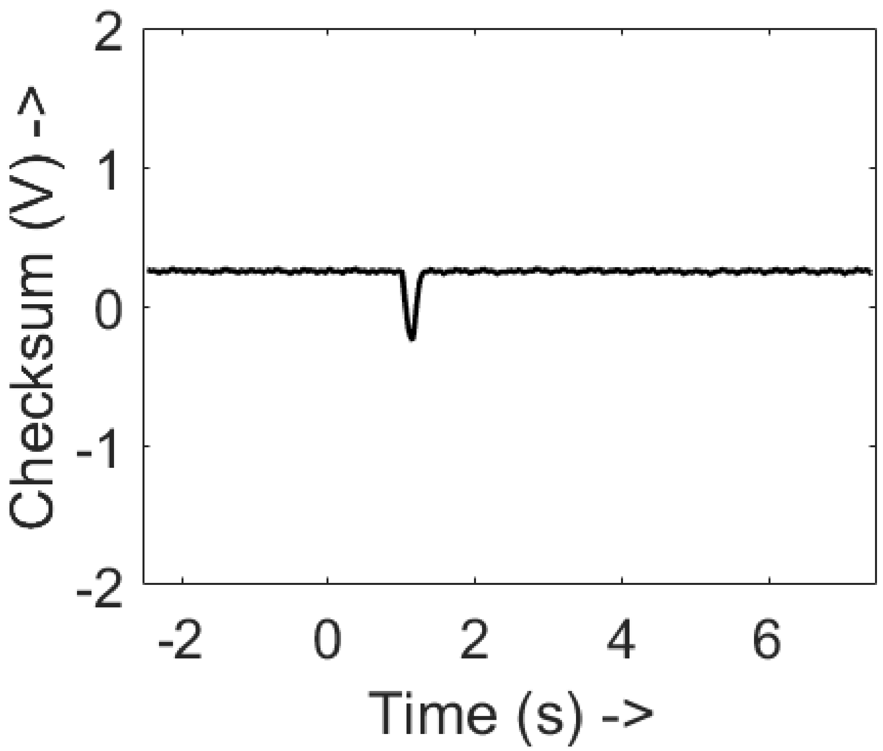


Figure 3.5: Analog checksum signal in presence of blocked rotor

3.1.5 Real-Time Parametric Error Compensation

In this study, the steady state value of ‘checksum error’ was used to detect and compensate for error. The error compensation process is based on the following steps:

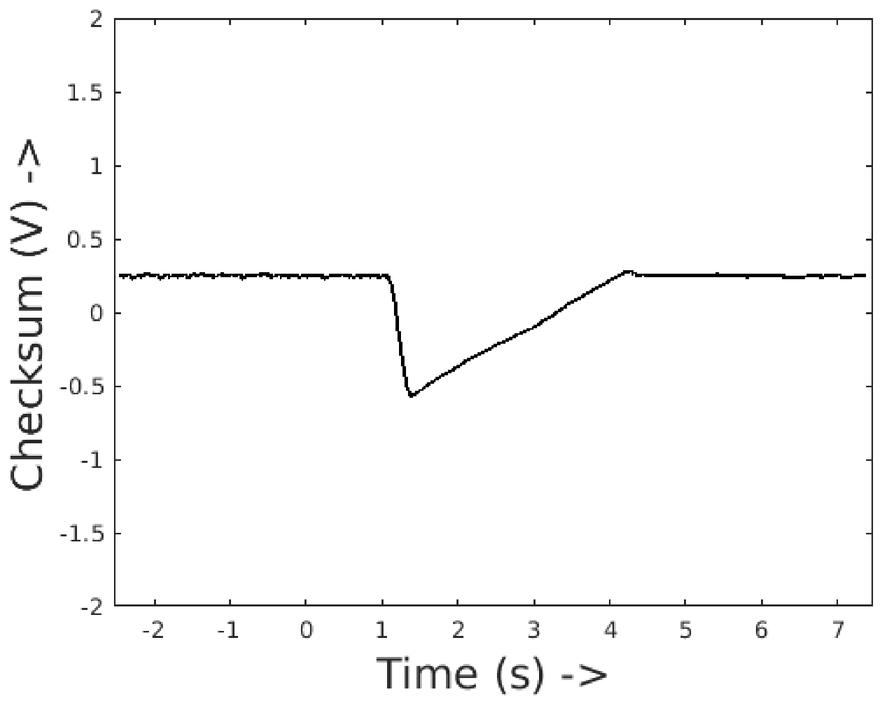


Figure 3.6: Analog checksum signal in presence of supply transient

Tuning Procedure

Analysis of the state space equations of the DC motor reveal that the sensitivity of the error signals e_1 and e_2 to perturbations in armature resistance is orders of magnitude smaller than that to perturbations in the other electro-mechanical parameters like torque constant, or back-emf constant. This is a fundamental property of the system state equations for the DC motor considered and to facilitate our study, the armature resistance R_a was measured independently. Hence, the error compensation involves measuring 3 metrics – e_1, e_2 and R_a. The tuning procedure involves 3 steps which are described in detail in the following subsections:

1. Measure e_1, e_2 and R_a.
2. Compute the deviations from nominal values - Δe_1, , Δe_2 and ΔR_a. Classify systems as pass/fail if the deviations exceed precomputed limits.

3. Formulate the tuning metric $m_e = \Delta e_1^2 + \Delta e_2^2 + \Delta R_a^2$ with appropriate normalizations and apply optimum controller gain K_{opt} as $K_{opt} = f(m_t)$ where $f(.)$ denotes the functional mapping from the tuning metric to the optimum gain.

Pass/Fail Classification

The nominal limits of Δe_1, , Δe_2 and ΔR_a are determined by introducing a $3-\sigma$ perturbation of 6% in the electro-mechanical parameters of torque constant K_e, back-emf constant K_b and coefficient of viscous friction B_1 as well as the armature resistance R_a. It is assumed that the system is calibrated for perturbations in armature inductance L_a and moment of inertia J from the nominal design values during deployment. Perturbations in L_a and J are negligible compared to the changes in the previously mentioned electro-mechanical parameters during on-field deployment.

100 systems were created in simulation by perturbing 4 parameters of K_e, K_b, B_1 and R_a with a Gaussian distribution with $\sigma = 2\%$ from the nominal values. 4 system specifications namely – 2% settling time, overshoot percentage, rise time and steady state error for the angular velocity ω were considered in this work. The nominal ranges of the specifications as well as the error metrics are determined from this experiment.

Systems are classified as 'pass' or 'fail' depending on the error metrics. For a system, a system is called 'fail' if any of the 3 error metrics Δe_1, , Δe_2 and ΔR_a are larger than the allowable nominal deviations $\Delta e_{1(opt)}$, , $\Delta e_{2(opt)}$ and $\Delta R_{a(opt)}$ (determined from the aforementioned experiment),and it is caleed 'pass' if all the error metrics fall within the acceptable limit. Similarly, a system is categorized as 'good' if all 4 specifications fall within the nominal specification limits or categorize systems as 'bad' if at least one of the specifications falls outside the acceptable nominal range. Among the 'bad' systems, if a search for the optimum controller gain for tuning the system into a 'good' one is performed and all 4 specifications can be restored within their nominal limits, such systems as reclassified as 'tunable'. 'Bad' systems which have experienced catastrophic parametric perturbations cannot be tuned with any controller gain and hence are not tunable.

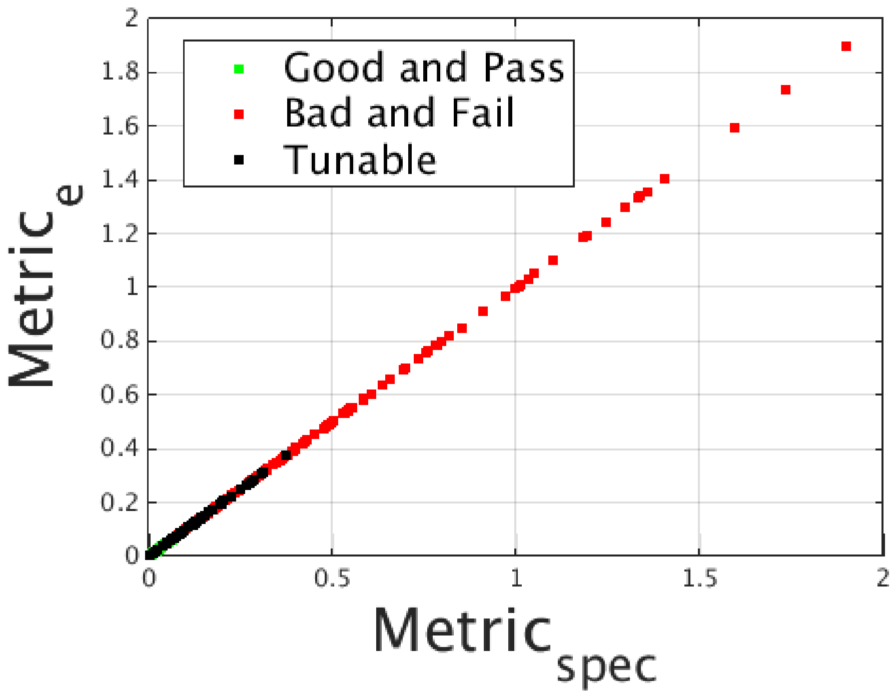


Figure 3.7: Correlation between normalized specification metric m_s and normalized error metric m_e

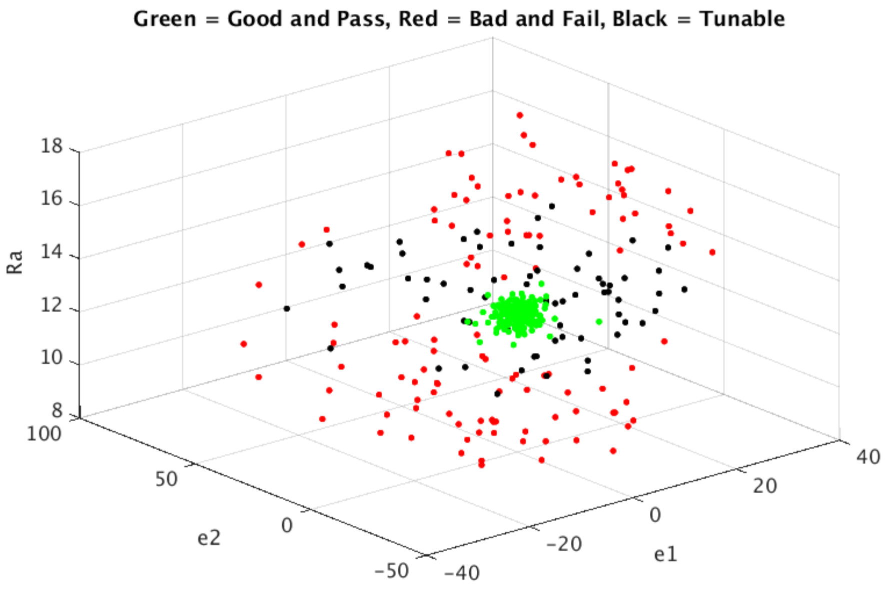
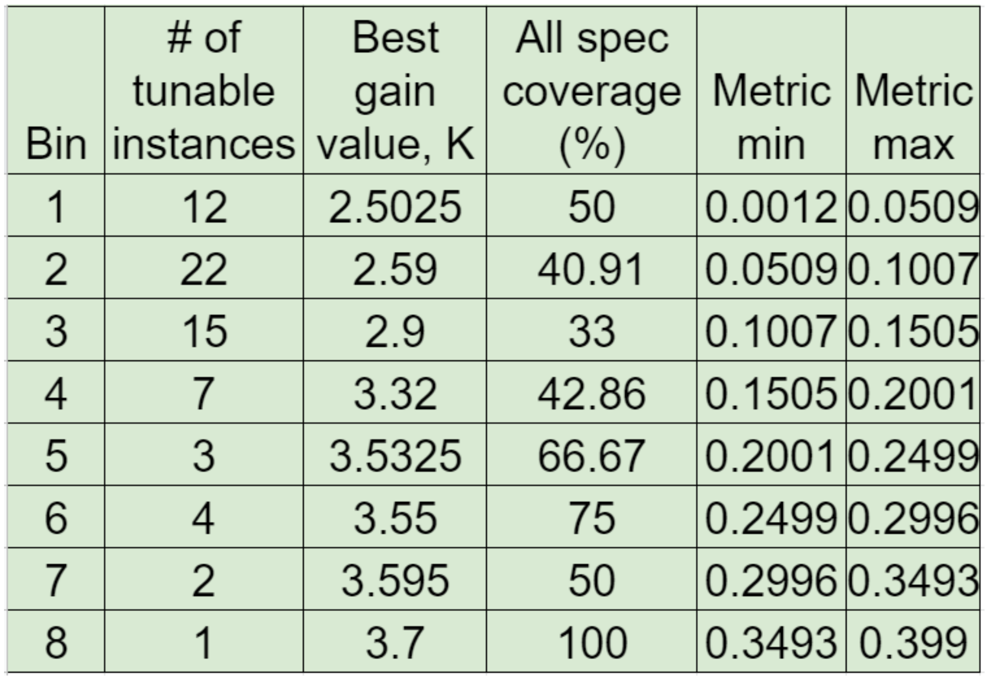


Figure 3.8: 3-d scatter plot showing the distribution of 'good', 'bad' and 'tunable' systems according to 'pass'/'fail' classification criteria on the $e_1 - e_2 - R_a$ axes

The pass/fail classification test based on the proposed tuning metric m_e is able to differentiate between 'good' and 'bad' systems since there exists a high correlation between the error metric m_e and the normalized specification metric $m_s = \Delta s_1^2 + \Delta s_2^2 + \Delta s_3^2 + \Delta s_4^2$, where $s_i, i = 1, 2, 3, 4$ are the different specifications mentioned previously. This correlation is shown in Figures 3.7 and 3.8. In Figure 3.7, the good systems are located very close to the origin and are not discernible. Here, as the parameters of the given system lies close to its nominal value, the error metric as well as the specification metric will lie very close to zero. That means the zero error results in compliance of all the specification of the system. On the other hand, as the parameter value deviates from its nominal one, different specifications of the system also deviates from its nominal value. This was the rational to select the aforementioned definition for the specification and error metric. Figure 3.8 shows a 3-d scatter plot on the $e_1 - e_2 - R_a$ axes where all the systems can be identified clearly.



Bin	# of tunable instances	Best gain value, K	All spec coverage (%)	Metric min	Metric max
1	12	2.5025	50	0.0012	0.0509
2	22	2.59	40.91	0.0509	0.1007
3	15	2.9	33	0.1007	0.1505
4	7	3.32	42.86	0.1505	0.2001
5	3	3.5325	66.67	0.2001	0.2499
6	4	3.55	75	0.2499	0.2996
7	2	3.595	50	0.2996	0.3493
8	1	3.7	100	0.3493	0.399

Figure 3.9: Compensation Look-up table (LUT) for DC Motor Control: The rightmost columns show the quantization limits of tuning metric specifying each bin uniquely. The specification coverage indicates the percentage of systems that are tuned by application of the optimal gain for that bin

Optimum Controller Gain Determination

400 instances of DC motor speed control systems were created in simulation where in 200 systems, 4 perturbed parameters K_e, K_b, B_1 and R_a follows a Gaussian distribution of $\sigma = 2\%$ from the nominal values. For the remaining 200 systems, the aforementioned parameters were perturbed with a uniform distribution of 30%. This distribution results in a set of systems where the 4 system specifications do not lie within the nominal range resulting in bad systems.

For each of the 'bad' systems, an exhaustive search for optimum controller gain was performed for tuning the systems such that all specification are restored within acceptable nominal boundaries. Here the gain was increased and it was checked whether the system meets all the specifications. If the specification is met, then the gain is the optimum gain. If not, the gain is again increased to see if this meets the specification. This was continued until the system becomes unstable. Systems for which an appropriate gain was obtained to tune the specifications are re-categorized as 'tunable' systems. For all such 'tunable' systems, the tuning metric m_e was computed. From simulation of a large number of systems it had been observed that the tuning metric stays within a particular range of values. All the tunable systems were grouped into 8 bins according to uniform quantization of tuning metric value. An optimum controller gain K_{opt} for each bin was selected in a way such that the number of tuned systems meeting all the system specifications is maximized for the corresponding bin by applying the same gain K_{opt}. This may result in a few systems in each bin for which application of the optimal gain of that bin does not tune the system. However, on an average, maximum number of systems in that bin will be tuned with the optimal gain parameter. The number of bins is determined by the tradeoff between the granularity of tuning process and the computational effort required to search for optimal controller parameters for each bin. In this work, an exhaustive search was performed for finding the optimal controller gain. Further computational savings can be achieved by more sophisticated techniques. Figure 3.9 shows the optimum controller gains obtained for

8 bins along with the tuning metric values specifying each bin limit and the specification coverage of the systems lying in each bin. Here, as an example, it is observed that bin 2 contains 22 tunable instances out of which our technique was able to tune 9 (40.91%) by selecting a K_{opt} of 2.59.

Performance of Error Compensation Procedure

The real-time error compensation involves measuring the three metrics of Δe_1, , Δe_2 and ΔR_a for a system in real-time. Computing the deviations Δe_1, , Δe_2 and ΔR_a from their respective nominal values will classify the systems as 'pass' and 'fail'. No compensation is performed on DC motor systems which are classified as 'pass' on the basis of the tuning metric m_e. For systems classified as 'fail', the tuning metric is computed and the appropriate bin in which the corresponding tuning metric value falls, is determined in real-time. If the tuning metric value does not fall into any of the precomputed bins, the compensation is not possible and offline tests are required to diagnose the faults for permanent fixes. However, the tuning metric value belonging to a particular bin's range indicates tunability and the corresponding optimal controller gain for that bin is applied to the DC motor in real-time for compensation.

In our simulation of 400 systems, 29 systems are incorrectly classified as 'pass' based on the tuning metric value. These false negative systems are those which doesn't meet at least one specification and still no compensation is applied to these systems, having passed the classification test. On the other hand, 8 systems out of 400 are 'good' systems and incorrectly classified as 'fail'.

Figure 9 shows the histogram plot of all 400 systems before performing real-time compensation. It can be observed that a significant number of systems remain outside the nominal boundaries for each of the specifications. The purpose of compensation procedure is to apply the appropriate controller gain and restore maximum systems within the acceptable nominal limits. Figure 10 shows the same histogram distribution of all 400 systems

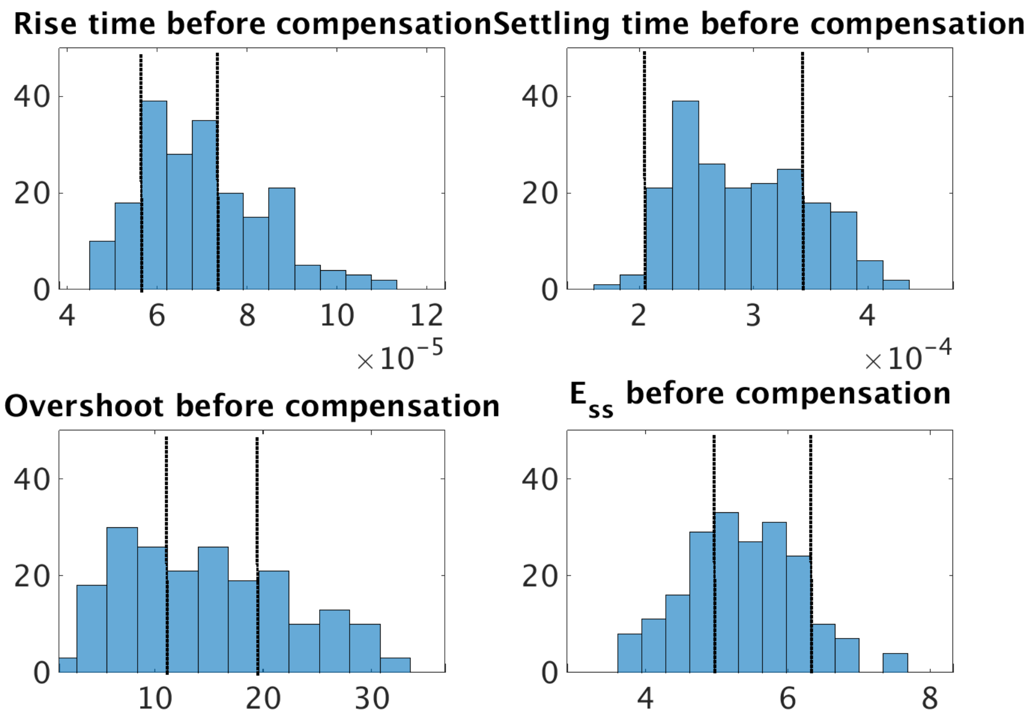


Figure 3.10: Histogram plots of rise time, settling time, overshoot percentage and steady state error (E_{ss}) before compensation. Vertical lines indicate the nominal range of each specification

after applying real-time compensation.

66 out of 400 systems were determined to be tunable based on tuning metric value (as shown in the Look-up-Table of Figure 3.9). After compensation, 30 of these 66 systems are tuned such that all their specifications are restored, yielding a compensation efficiency of 45.45%. These are the systems which were going to be identified as faulty ones without any compensation. Only this approach was able to compensate them and hence they can be used as a good system. This efficiency can further be improved by employing more sophisticated controllers like PID control allowing more degrees of control parameters and by increasing the number of bins. This compensation efficiency has to be considered with the fact that the proposed methodology provides a real-time compensation for the first time, unlike prior works which required offline model estimation methods for optimal control.

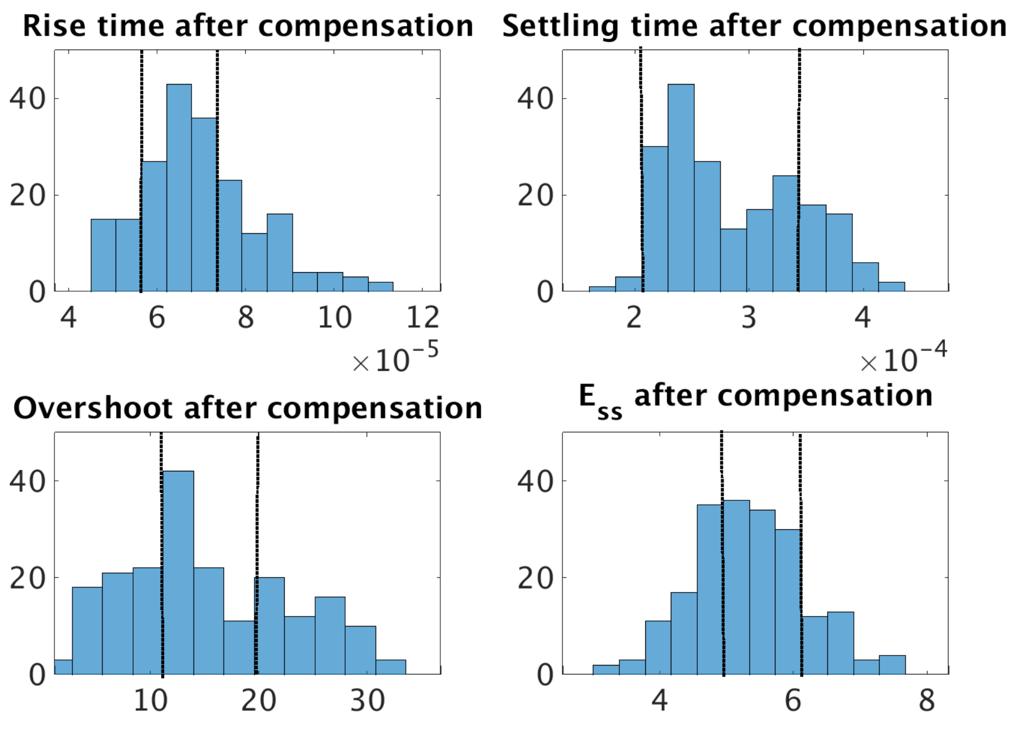


Figure 3.11: Histogram plots of rise time, settling time, overshoot percentage and steady state error (E_{ss}) after compensation. It can be observed that more number of systems have been included within the nominal specification limits, validating the efficacy of real-time compensation

3.1.6 Summary

A real time error detection scheme was introduced for speed control of a DC motor in this paper. The error detection scheme was able to detect the target electrical and mechanical faults. The error detection scheme was very simple in nature yet it was very effective in detecting the faults. The methodology has been demonstrated experimentally and it is found to be in good match with the simulation. Future work will involve the error detection methodology for field controlled DC motor speed control system which is non-linear in nature and also on the correction of error for armature controlled DC motor speed control system.

3.2 On-Line Error Diagnosis, and Compensation Linear State Variable Circuits and Systems Using Time-Domain Checksum Observers

3.2.1 Introduction and Key Contribution

In the past, circuit and system engineers have assumed that the underlying hardware which is used to actuate a linear system and its control algorithms as well as the relevant sensors and actuators are correct by design [10, 11, 16, 18, 122, 123]. However, the sensors and actuators as well as different components of the circuit and system go through wear and tear over time and gradually degrade in performance resulting in realistic failure events which can lead to loss of Quality of Service (QoS) and eventually system failure. From the perspective of application-level safety, it is imperative that such anomalies be diagnosed in real-time with high accuracy before the onset of failures resulting in catastrophic accidents.

In the past, there has been work on detection of faults using system models derived from off-line and real-time experiments [117–121]. In some of these works, the system model is estimated using off-line experiments (system parameter measurements). In other research, the system and its model are both stimulated with the same input in real time and the plant output is compared against the model output. Any discrepancies between the two are flagged as failures. However, it is difficult to perform failure diagnosis in real-time using a model of the system alone, as this requires the use of computationally intensive model parameter estimation algorithms.

While error detection can be performed using a variety of techniques [36, 39, 45, 48, 94, 111, 112], error correction for a state variable system is a much harder problem. Typically, error detection is followed by diagnosis and once diagnosis is complete, error compensation is performed via error feedback [36, 39] or adaptation of the feedback control law of the system [94]. It was shown in [39] that the steady state values of multiple checksums are required for real-time diagnosis.

Key Contributions

The key contributions of this research [79] are as follows:

1. In this work it is demonstrated for the first time, that the transient response of the checksum error to dynamic input stimulus in a checksum-encoded linear state variable system contains diagnostic information about the values of multi-parameter perturbations that causes the system to behave differently from its normal behavior. In theory, the transient response of even a single checksum can be used to uncover a wealth of multi-parameter diagnostic information about multi-parametric deviations within the system.

2. This information can subsequently be used to significantly improve both the speed and accuracy of estimation-driven multi-parameter diagnosis algorithms as well as diagnosis-driven control law correction algorithms for linear real-time state variable systems. An algorithm for detection, diagnosis and correction of multiple failure events at different points in time is developed.

3.2.2 Proposed Approach: Error Diagnosis, and Correction

As explained in the previous section, the error due to a failure can be detected in real-time comparing 'checksum error' signal with respect to precomputed threshold. However, the diagnosis and correction has scope of improvement. In this section, the idea for error correction is presented using the checksum error signal. As observed from previous studies, the crossing the 'checksum error' signal over a precomputed threshold would detect the failure. Additionally, it is also observed that the time-domain property of checksum error can be used to diagnose multi-parametric deviations in the plant parameters of a linear control system. The signal waveform of the checksum error for a faulty system depends on the input stimulus. Different input stimuli excite multi-parameter variations in different ways, resulting in dissimilar checksum error signals. Conversely, the same input stimulus may produce different error signals in systems with different failure conditions. The diagnosis of the source of failure is performed with a two-step process (pre-deployment and post-deployment) which are explained below:

Pre-deployment Step: Training of Checksum Error

The input stimulus, along with the corresponding checksum error response across a multitude of multi-parameter failure conditions is used to train a regression function to predict failed plant parameter values from observations of the same. The training procedure is depicted in Figure 3.12.

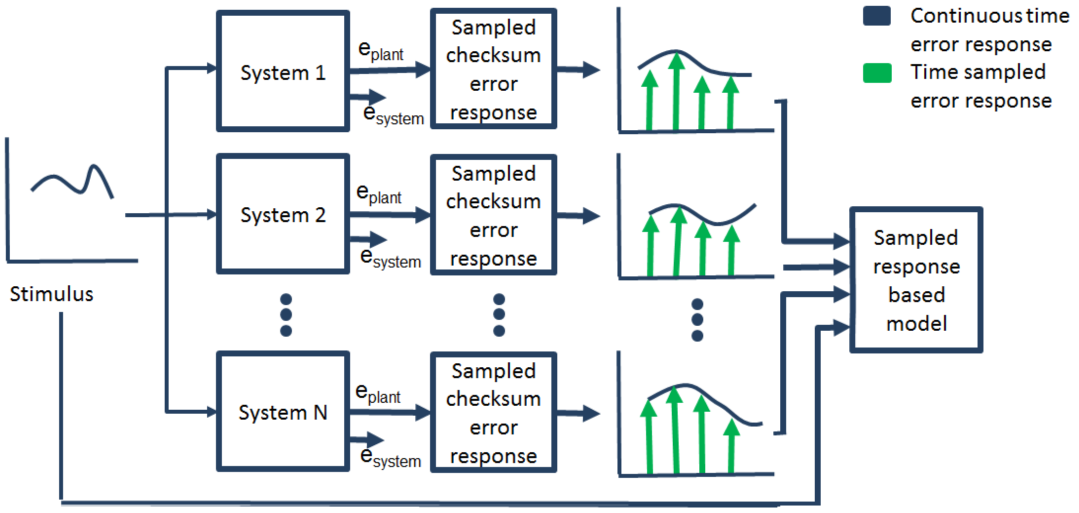

Continuous time error response
Time sampled error response
Stimulus
System 1
System 2
System N
e_plant
e_system
Sampled checksum error response
Sampled response based model

Figure 3.12: Training of checksum error response for diagnosis

As shown in Figure 3.12, a family of N parameter-perturbed systems (plant parameters) is stimulated by the same input and the checksum response is captured by digitizing the checksum error waveform. This experiment is repeated for diverse sets of input stimuli. A regression based model is trained with samples of the checksum error and stimuli as inputs and the perturbed plant parameter set of the system as output. This trained model is later used for coarse plant parameter estimation of the linear system.

Post-deployment Step: Diagnose and Correct

In this section, the idea for error correction using the checksum error signal is presented. The system architecture of a regular control system is shown in Figure 3.13 and the flow-chart of the proposed methodology for error correction is shown in Figure 3.14. In Figure 3.13, the 'set point' represents the operating point of the system which the system is designed to track with assistance from the controller. An additional checksum, namely the "system checksum" was introduced for error correction. In Figure 3.13, the 'plant checksum' is created by only encoding the state equations of the plant, P and the 'system checksum' is implemented by encoding the state equations of both the plant P and the controller, K (the latter is updated upon failure detection to enable detection of future failure events)

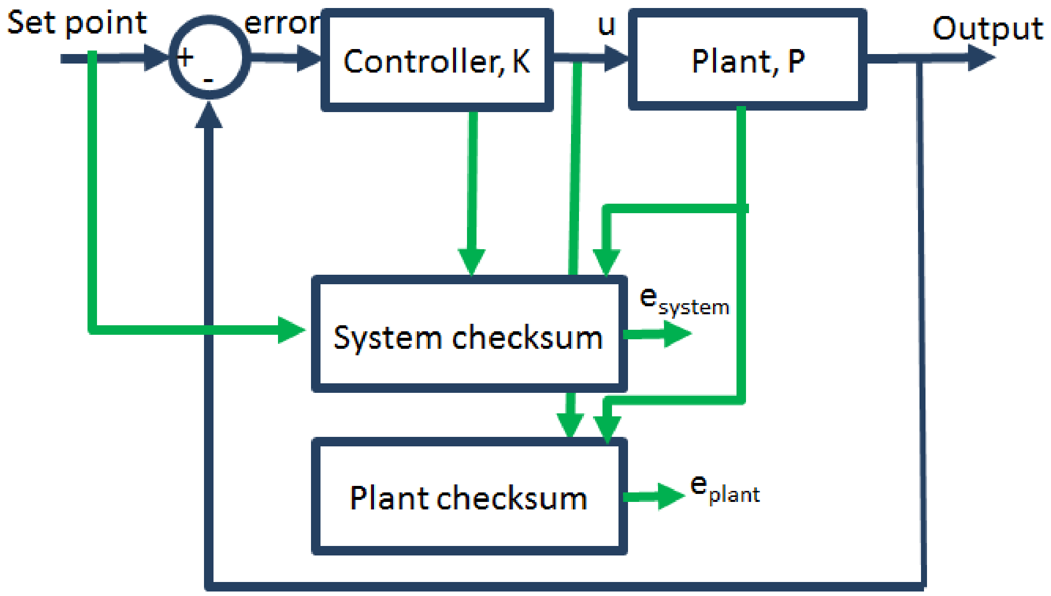


Figure 3.13: System architecture for error correction

It must be noted that, the plant may experience several parametric failure events over the course of time, and the precomputed model described in Step 1 becomes invalid if the 'plant checksum' is updated after failure detection. For this reason, the same reference 'plant checksum' over time was maintained. This allows diagnosis of more than one parametric failure event while one additional checksum, the 'system checksum' is required to detect

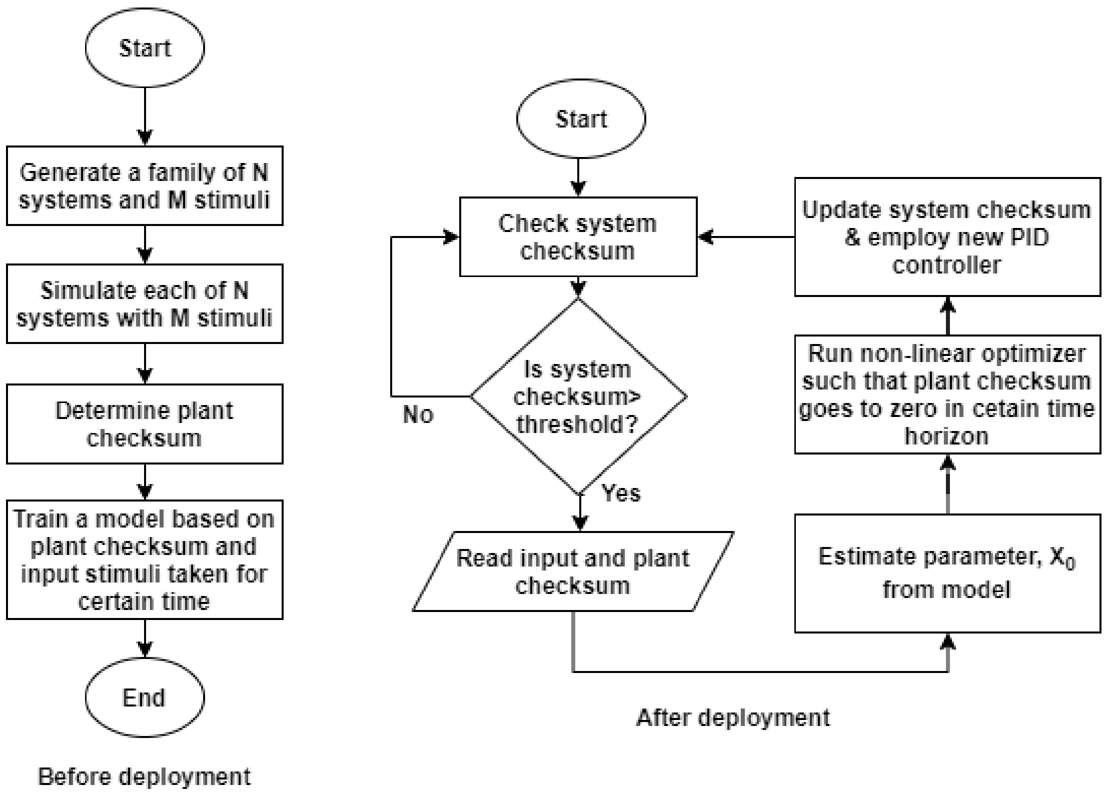

Start
Generate a family of N systems and M stimuli
Simulate each of N systems with M stimuli
Determine plant checksum
Train a model based on plant checksum and input stimuli taken for certain time
End
Before deployment
Start
Check system checksum
Is system checksum> threshold?
No
Yes
Read input and plant checksum
Estimate parameter, X0 from model
Run non-linear optimizer such that plant checksum goes to zero in cetain time horizon
Update system checksum & employ new PID controller
After deployment

Figure 3.14: Error correction flow chart

more than one failure event across time and is updated after each such event. The 'system checksum' triggers the correction scheme, shows the status of the controller updates and allows the checksum error to return to zero when the controller update is completed. In a nominal system, both the plant and system checksum errors are ideally zero (lie within specified certain thresholds in the presence of measurement noise). In the event of parametric failure, both of these checksum errors, namely $e_{plant}(t)$ and $e_{system}(t)$ are non-zero. When $e_{system}(t)$ is greater than a predefined threshold and hence triggers the correction scheme, the parameters of the modified plant (after multi-parameter failure) are estimated from the input stimulus and $e_{plant}(t)$ using the previously learned regression model. Let the estimated parameter set from the model be X_0. After obtaining the initial parameter estimate, a non-linear optimization is performed in real-time to determine the best estimate for the plant parameter set by minimizing $e_{plant}(t)$ taking X_0 as the initial starting point for the search in the parameter space of the plant, P. It must be noted that from the time the system checksum crosses the threshold indicating parametric failure, the plant inputs as well the checksum error response are digitized and recorded for use as input to the regression model described earlier. It is imperative that the error compensation be performed as rapidly as possible. The estimated parameters X_0 from the regression function, help the real-time optimization to converge quickly. After obtaining the revised parameter estimate, the controller, K is updated in such a way that the system specifications are restored. At the same time, the 'system checksum' is also updated with the estimated plant parameter values such that $e_{system}(t)$ is forced to zero and can trigger the error detection/correction scheme again in case of future parametric failure.

3.2.3 Test Cases Overview

Test Case I: DC Motor Control

DC motor speed control system has been considered as the first test case for this study. The system dynamics and check constructions are already explained in Section 3.1.2, and hence is skipped here for the sake of brevity.

Test Case II: Generator Connected to Infinite Bus

A generator connected to infinite bus is the second test case for our proposed approach. The generator system connected to an infinite bus is expressed as a linear state variable system whose representation is given as:

$$\begin{aligned}\frac{d|E_a'|}{dt} &= \frac{E_{fd}}{T_{d0}} + \frac{1-k_3}{k_3 T_{d0}}|V_\infty|\cos\delta - \frac{E_a'}{k_3 T_{d0}}\\ \frac{d\delta}{dt} &= p\\ \frac{dp}{dt} &= \frac{P_m}{M} - Dp - P_g(|E_a'|,\delta)\end{aligned} \tag{3.8}$$

where, $P_g(|E'_a|, \delta)$ and k_3 have following expression.

$$\begin{aligned} P_g(|E'_a|, \delta) &= \frac{|E'_a||V_\infty|}{x'_{dm}} \sin\delta + \frac{|V_\infty|^2}{2}\left(\frac{1}{x'_{qm}} - \frac{1}{x'_d m}\right) \sin 2\delta \\ k_3 &= \frac{x'_d + x_L}{x_d + x_L} \end{aligned} \tag{3.9}$$

Here, E_{fd} = Field voltage, T_{d0} = Rotor time constant, δ = Power angle, x_d = Direct axis reactance, x'_d = Direct axis transient reactance, x_q = Quadrature axis reactance, x_L = Line reactance, P_m = Mechanical power input, P_g = Electrical power output, $x_{qm} = x_q + x_L$, and $x'_{dm} = x_d + x_L$. All the quantities have the unit of per unit (p.u.), except power angle whose unit is in degrees. The state vector of this system is given as: $x = [|E'_a|, \delta, p]^T$ and input to this system consist of the field voltage and mechanical power which is supplied to the generator.

Equation 3.10 shows that the generator connected to an infinite bus is a non-linear system by nature. Hence, the checksum is implemented on a linearized version of the model at its operating point. The linearized version of the system at its operating point is given below in equation (7):

$$\dot{x} = \begin{bmatrix} -\frac{1}{k_3 T_{d0}} & \frac{(1-k_3)|V_\infty|}{k_3 T_{d0}} & 0 \\ 0 & 0 & 1 \\ \frac{1}{M}\frac{\partial P_g(|E'_a|,\delta)}{\partial |E'_a|} & \frac{1}{M}\frac{\partial P_g(|E'_a|,\delta)}{\partial \delta} & -\frac{D}{M} \end{bmatrix} x + \begin{bmatrix} \frac{1}{T_{d0}} & 0 \\ 0 & 0 \\ 0 & \frac{1}{M} \end{bmatrix} u \tag{3.10}$$

where, $x = [|E'_a|, \delta, p]^T$, and $u = [E_{fd}, P_m]^T$. In this way, the linearized version of generator connected to infinite bus can be represented in matrix form as $\dot{x} = Ax + Bu$. From this representation, the checksum circuit was implemented by selecting a coding vector $CV = [\alpha_1, \alpha_2, \alpha_3]^T$.

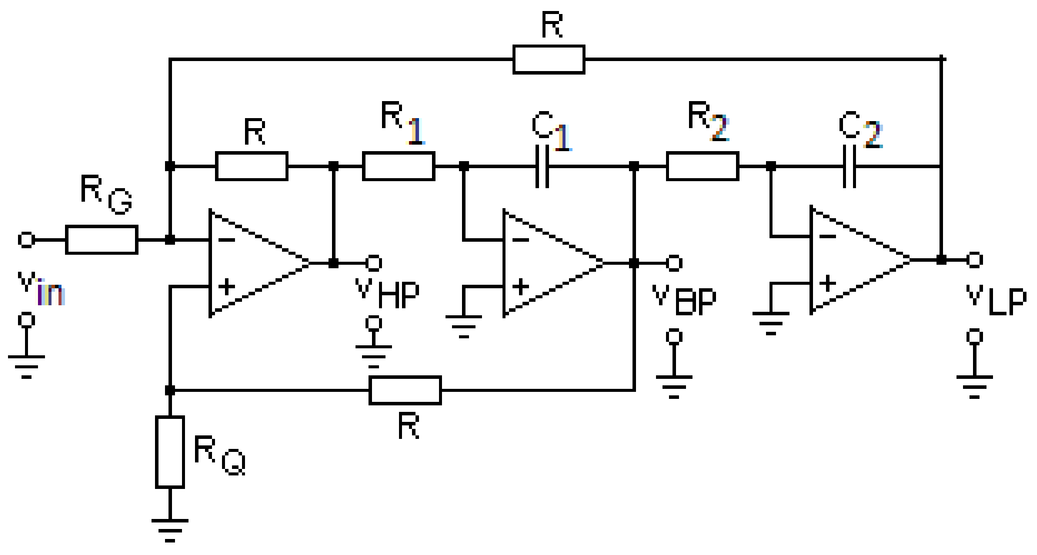


Figure 3.15: Biquad filter

Test Case III: Analog Biquadratic Filter

An analog biquadratic filter (also known as biquad filter) is considered as the third test case and is shown in Figure 3.15. The biquad filter is expressed in state space form by:

$$\dot{x} = \begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} x + \begin{bmatrix} 0 \\ b_2 \end{bmatrix} u \tag{3.11}$$

where, $a_{11} = -\left(2 + \frac{R}{R_G}\right) \frac{R_Q}{R_Q + R_1} \frac{1}{R_1 C_1}$, $a_{12} = \frac{1}{R_1 C_1}$, $a_{22} = -\frac{1}{R_2 C_2}$, $b_2 = \frac{R}{R_G} \frac{1}{R_1 C_1}$. And, $x = [v_{BP},\, v_{LP}]^T$, $u = v_{in}$. From the representation given in equation 3.11, the checksum circuit can be implemented by selecting a coding vector $CV = [\alpha_1,\, \alpha_2]^T$.

3.2.4 Result: Real-Time Diagnosis of System Parameters

DC motor with control

1. *Simulation*: The system along with its error detection scheme was simulated in MATLAB. The faults injected in the system were multi-parameter perturbations in coeffi-

cient of armature reactance L_a, moment of inertia J, torque constant K_e, and armature resistance R_a. Injection of faults changes the state-space representation of the system and generates a non-zero checksum error signal $e(t)$ indicating the presence of parametric perturbations in the system.

2. *Model training*: After selecting a proper coding vector, a family of 400 parameter-perturbed DC motor control systems was created. The parameters were varied with a uniform distribution of 30% and stimulated with unit step signals. The checksum error signals $e_{plant}(t)$ for each of the systems were sampled at 100 samples per second. Among the 400 systems, 200 instances were selected in random to train a Multivariate Adaptive Regression Spline (MARS) model with the perturbed parameter set as target output. After training, a MARS model was obtained with 14 basis functions.

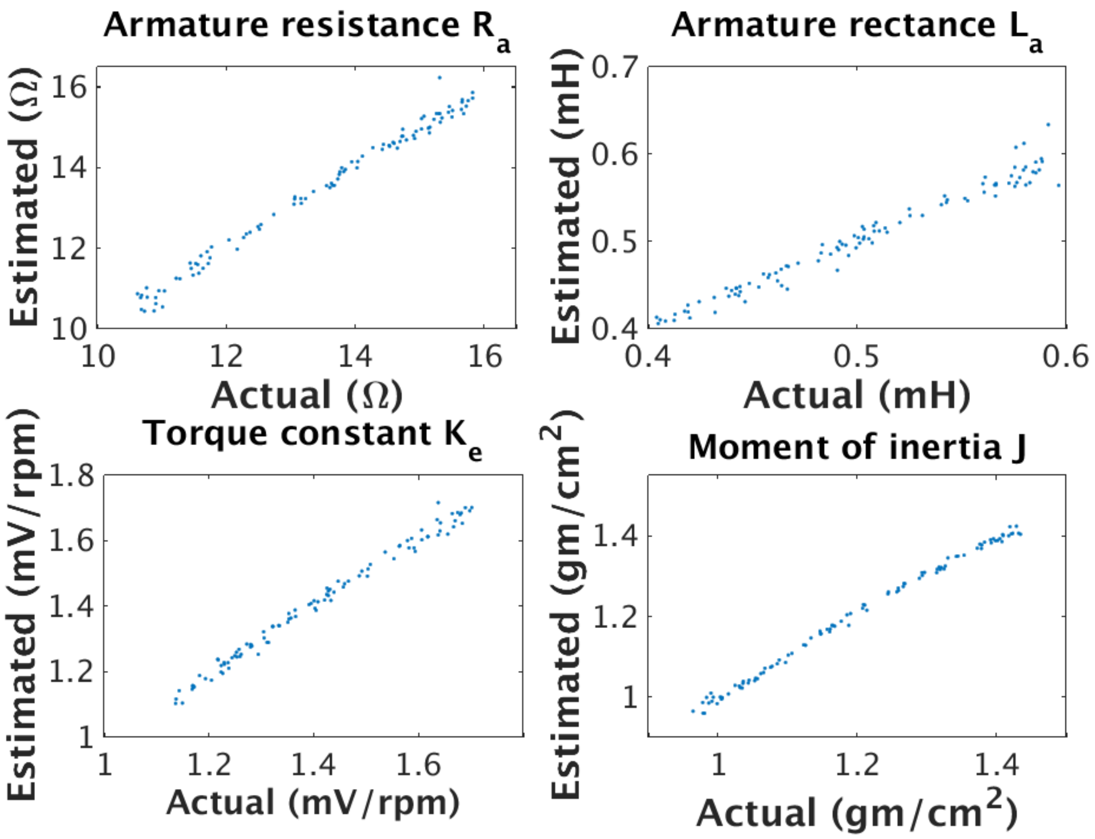


Figure 3.16: Estimated vs Actual parameter of DC motor with control

3. *Model validation*: The trained model is tested for validation by the remaining 200 in-

stances of systems. The graph of estimated parameters obtained by the MARS model vs actual parameters are shown in the Figure 3.16. It clearly shows that the proposed approach of using only one checksum can accurately estimate the parameter of the DC motor control system.

Generator connected to infinite bus

1. *Simulation*: As the second test case, a generator connected to infinite bus was considered. It was also simulated in MATLAB. Multi-parameter perturbations were introduced by varying direct axis reactance X_d, quadrature axis reactance X_q, direct axis transient reactance X_{dp} and rotor time constant T_{d0}. The checksum was implemented on the linearized version of the system when input mechanical power gradually increased from 0 p.u. to 0.8 p.u as a ramp signal.

2. *Model training*: The model for this system was trained in the same way like the previous case. 400 instances of generator connected to infinite bus system were created. After simulation, the checksum signal $e_{plant}(t)$ was captured by sampling at 100 samples per second. 200 instances are selected in random to train a MARS model with 8 basis functions.

3. *Model validation*: The trained model was tested for validation by the remaining 200 instances of generator systems. The graph of estimated parameters obtained by the MARS model vs actual parameters is shown in the Figure 3.17. As can be seen from the obtained graph, the model is able to estimate direct axis reactance, quadrature axis reactance, direct axis transient reactance and rotor time constant with very high accuracy. This again demonstrates that temporal information of only checksum can be used for parameter prediction.

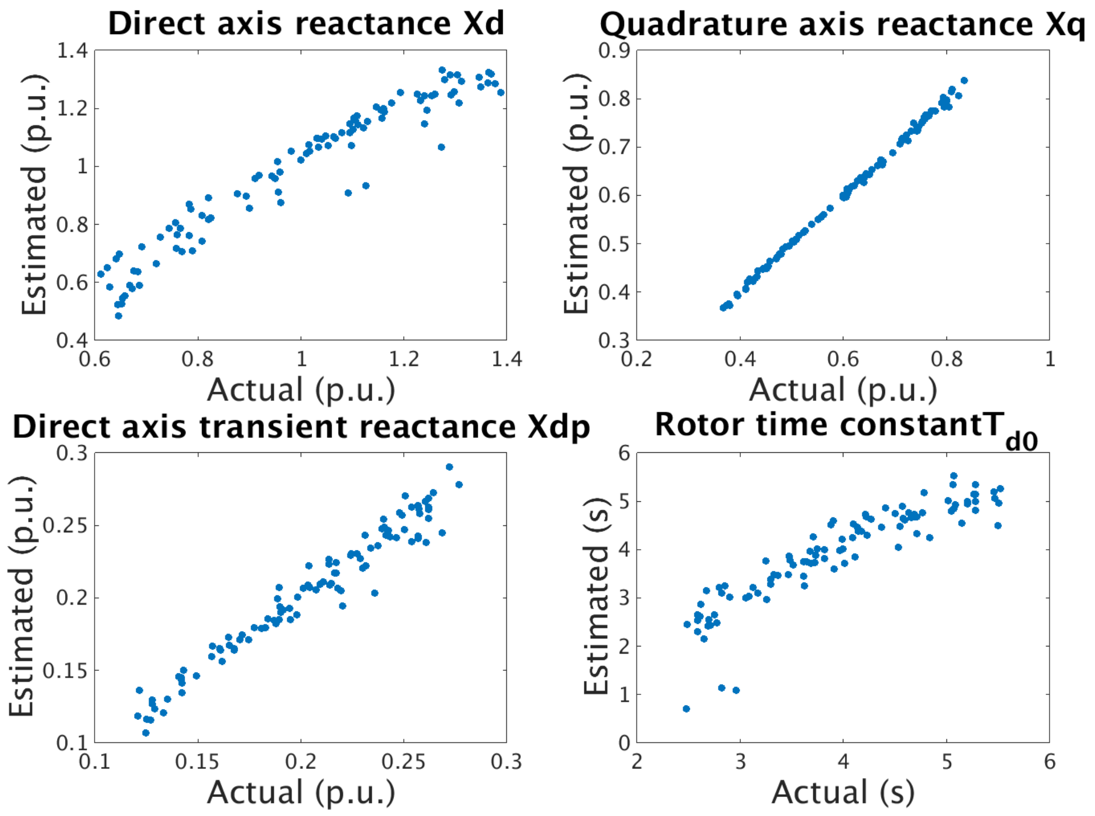


Figure 3.17: Estimated vs Actual parameter of generator connected to infinite bus

Biquadratic filter

1. *Simulation*: The biquadratic filter was simulated as the third test case to demonstrate the applicability of the proposed approach to analog circuits as well. This system was also simulated in MATLAB along with error detection. A family of 500 biquad filters was created by varying its R_1, R_2, C_1, and C_2 values from a uniform distribution of 40% variation. Introduction of these variation injects faults in the biquad filters in its central frequency and quality factor and by this way, parametric faults are injected in the system. Here, only these four parameters were varied to show the proof of concept and it can be very extended to all the parameters of the system.

2. *Model training*: The system was excited with single-tone sinusoidal input signal and the checksum error signal $e_{plant}(t)$ was recorded for all the circuit instances. After computing the checksum error of all the filters, the amplitude ratio and phase lag

of this error signal with respect to the input signal, were determined. Taking the amplitude ratio and phase lag of 250 instances as input, a Support Vector Machine (SVM) [124] based learner was trained. ‘Radial Basis Function’ was used as the kernel function to train the SVM classifier model.

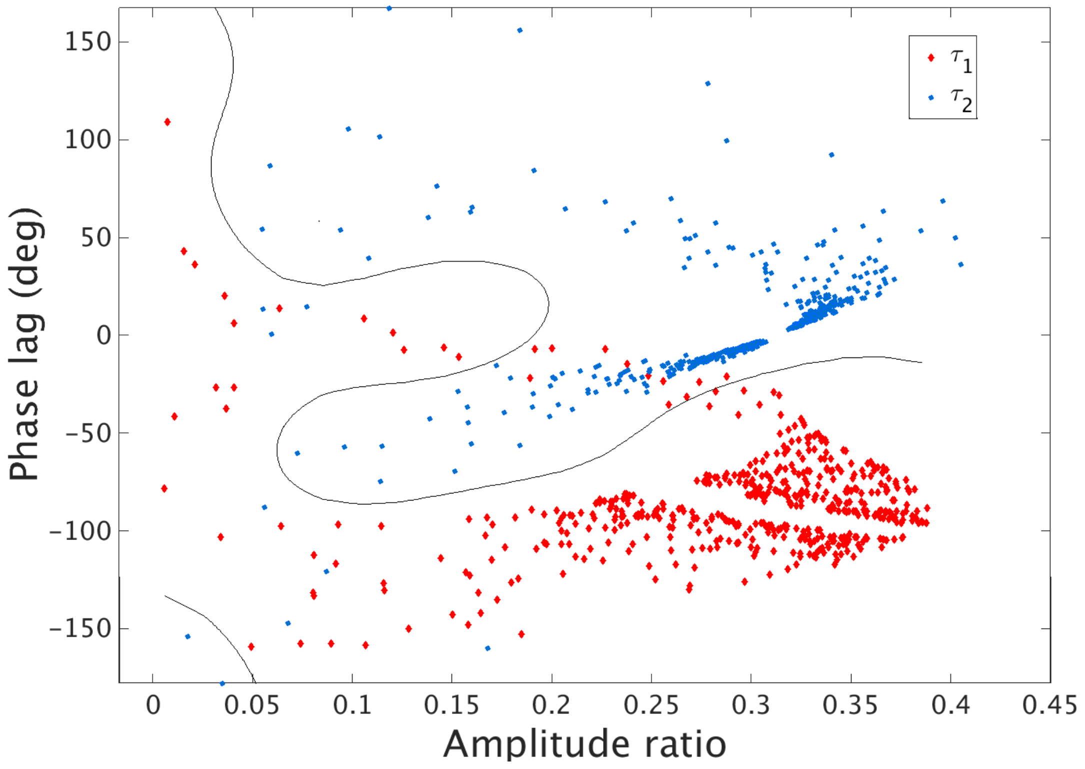


Figure 3.18: SVM classification of time constants of biquad filter

3. *Model Validation*: The trained model was tested for validation by the remaining 250 instances of biquad filters. The resulting plot was shown in Figure 3.18. As can be observed from the graph, the SVM classifier was successfully able to create a distinct boundary to classify whether the fault is because of $\tau_1 = R_1C_1$ or $\tau_2 = R_2C_2$. The misclassification rate observed for this classifier is 2.8%. These three test cases clearly demonstrate the viability of the proposed approach to diagnose the source of faults from the temporal information of only one checksum error using any suitable regression/learning algorithm.

3.2.5 Result: Real-Time Error Correction

A detailed description of error correction approach for a linear state variable system is already provided in Section 3.2.2 and in this section, the result obtained using this approach on a DC motor control system with PID controller is discussed.

System architecture

For correction of parametric faults, two checksum signals are introduced in the DC motor speed control system namely, 'plant checksum' and 'system checksum' as shown in Figure 3.13. The 'system checksum' is being used to trigger the correction scheme whereas the 'plant checksum' would be used to estimate the parameter from model as well as to perform nonlinear optimization.

Pre-deployment learning

Before the deployment stage, a family of 128 systems has been created with a PID controller to attain a nominal %overshoot (%OS) of 5% and rise time (T_r) of $100\mu s$. Each of those systems was simulated with 16 stimuli and their $e_{plant}(t)$ has been recorded for a certain time horizon. A neural network based model has been trained using $e_{plant}(t)$ and its corresponding stimuli as the input and the plant parameter as the target output. For the error diagnosis results presented in Section 5.4.1, a MARS model was trained for prediction purposes. Here, a neural network based model was trained as any appropriately designed machine learning algorithm would be a candidate for this purpose.

Post-deployment optimization

Plots of ω, $e_{plant}(t)$, and $e_{system}(t)$ of one system, which has %OS= 1.1% but $T_r = 102\mu s$ which violates nominal specification, are shown in Figure 3.19 and post-deployment correction is explained using these plots. At the post-deployment stage, the error correction scheme checks the system checksum, $e_{system}(t)$ at all times. As soon as the magnitude

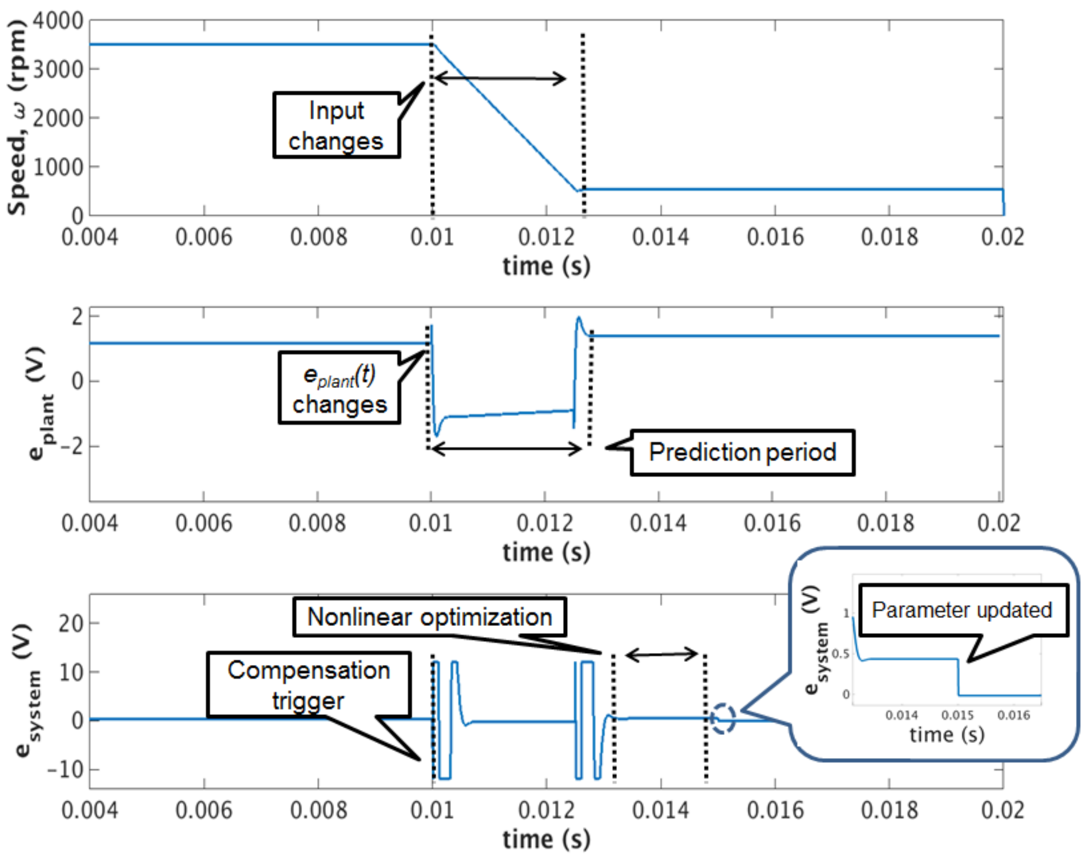


Figure 3.19: Error correction of DC motor system

of $e_{system}(t)$ becomes higher than a pre-defined threshold indicating parametric failures, $e_{plant}(t)$ and the input stimuli are recorded and used as inputs to the model learned at pre-deployment stage. In this example, $e_{system}(t)$ of this system crosses the threshold just before 10 ms. The compensation scheme records the value of $e_{plant}(t)$ and input stimulus from 10 ms to 12.5 ms and predict the parameters from model. Using the output of this model as the initial condition, a 'Levenberg-Marquardt' algorithm [125, 126] based non-linear optimization is performed in real-time from 12.5 ms to fine tune the parameter estimation. Because of using the initial search point, the optimization completed very quickly at 15 ms. The system checksum and the PID controller of the system is updated after the optimization is completed at 15 ms using the final predicted parameter set that drives $e_{system}(t)$ to zero again as shown in Figure 3.19 and updates the controller to restore system specifications to nominal value of %OS= 3.1% and $T_r = 99.6\mu s$. This demonstrates the viability of error correction scheme.

3.2.6 Summary

A real time parametric diagnosis approach for the linear state variable system has been proposed in this work. The approach has been applied to three test cases and they have been able to diagnose the system parameters with high accuracy. Error correction was demonstrated for a DC motor with PID controller. These examples represent the viability of this proposed approach. Future work will involve demonstration of error correction on other complex linear and nonlinear systems operating under arbitrary failure mechanisms.

CHAPTER 4
DETECTION, DIAGNOSIS AND COMPENSATION OF CONTROL PROGRAM, SENSOR AND ACTUATOR FAILURES IN NONLINEAR SYSTEMS USING HIERARCHICAL STATE SPACE CHECKS

4.1 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 [127], 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 [19] for 2016 shows that a self-driving car *failed about every 3 hours due to hardware or software malfunction*. Other examples abound [20–22]. The most recent Boeing incident [20, 21], 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*.

4.2 Key Contributions

The key contributions of this research [81, 84, 85] are as follows:

1. A novel *hierarchical error checking methodology for autonomous systems is developed* in this research that allows errors to be detected with ultra low latency and high sensitivity across the different components of an autonomous robotic system. The effects of subsystem errors on system function are detected by system level checks that ensure high coverage of internal failure mechanisms.

2. The technique allows *errors in sensors and actuators of the system to be diagnosed with low diagnostics granularity* using a matrix of checks across the different levels of the design.

3. The methodology allows *diagnosis and correction of multiple simultaneous failures* in different subsystems while allowing assessment of the impact of such failures on overall system performance. The contribution in individual subsystem recovery is given below:

4. *Control Program Errors:* There has been limited attention to recovery from errors in control program execution on a digital processor [128, 129]. 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.

5. *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.

6. *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.

4.3 Preliminaries: Quadcopter and Brushless DC Motor Models and Control

4.3.1 State Variable System: Overview

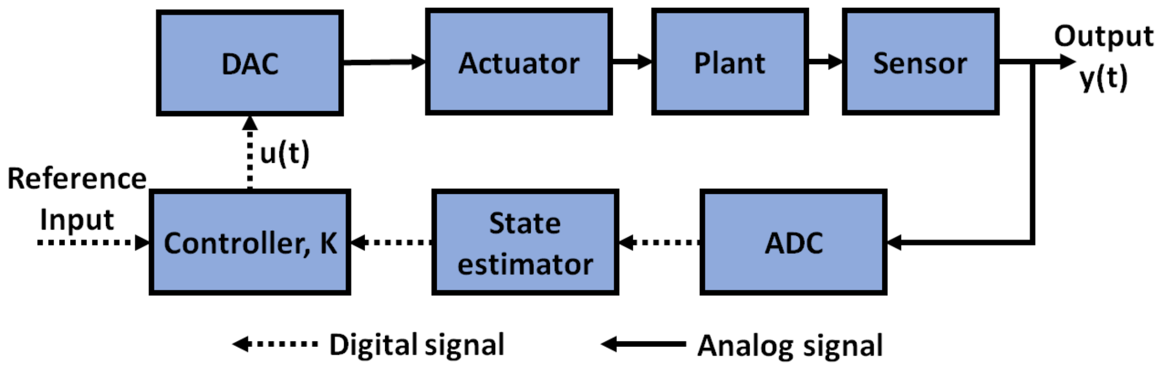


Figure 4.1: A nonlinear state variable system

Figure 4.1 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.

$$\dot{s}(t) = f(s(t), u(t)) + w(t) \tag{4.1}$$

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,

$$z(t) = h(s(t), u(t)) + v(t) \tag{4.2}$$

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 [130], Lyapunov based controller design [131] or on a reinforcement learning (RL) based controller [132]). 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.

4.3.2 Quadcopter Overview

As an example test case of a nonlinear state variable system, quadcopter control is considered in this work which has become a defacto standard to assess the performance of different research on nonlinear state variable autonomous systems [133–137] and which also is used for different applications [138–140]. 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 ϕ, θ and ψ represent the roll, pitch and yaw angle in the body frame of the quadcopter (see Figure 5.15). 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 [141]:

$$\begin{aligned} [\ddot{x}, \ddot{y}, \ddot{z}]^T &= [0, 0, -mg]^T + RT_B + F_D \\ [\ddot{\phi}, \ddot{\theta}, \ddot{\psi}]^T &= I^{-1}(\tau - \omega \times (I\omega)) \end{aligned} \tag{4.3}$$

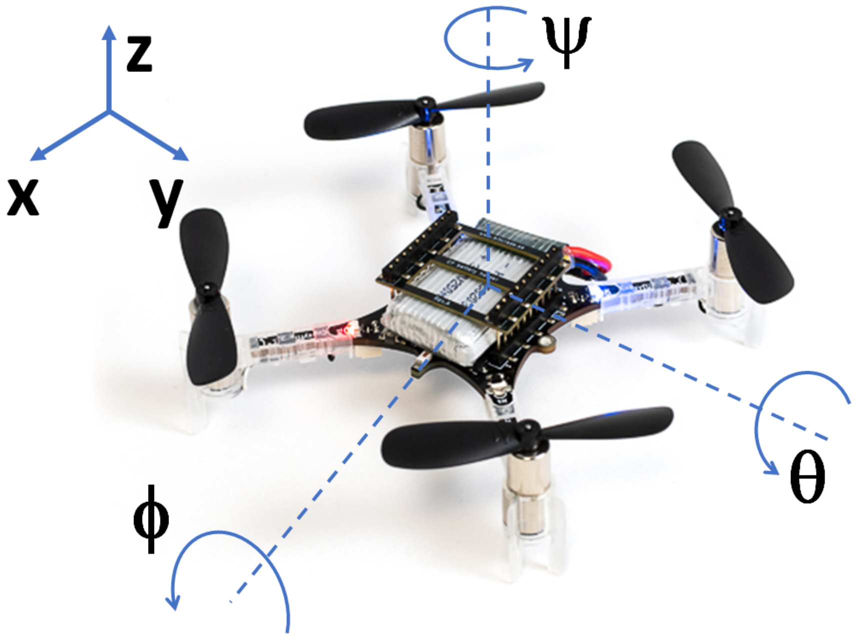


Figure 4.2: An Example Quadcopter System (adopted from [142])

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

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

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, Ω_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. The simplified expression for k is given in the following which establishes the relation between thrust generated by a motor-propeller pair with the motor speed [143]:

$$k = 18.5944745 \times 10^{-12} d^{3.5} \sqrt{pitch} \tag{4.4}$$

where, d = the propeller diameter, and $pitch$ = propeller pitch. Applying appropriate thrust from each motor changes the dynamics of the system, and hence control the quadcopter in the desired way.

4.3.3 Actuator Overview: Brushless DC (BLDC) Motor

For actuation, the use of brushless DC (BLDC) motors has been investigated in this work [144]. 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 [145]:

$$\mathrm{A}_{act} =$$

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

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

where, $x_{act}(t) = [I_{a,act}, I_{b,act}, I_{c,act}, \omega_{act}, \theta_{act}]^T$ and $u_{act}(t) = [V_{as}, V_{bs}, V_{cs}, T_l]^T$ are the motor state and input respectively. R_s, λ, J, B_f and P are the stator resistance per phase,

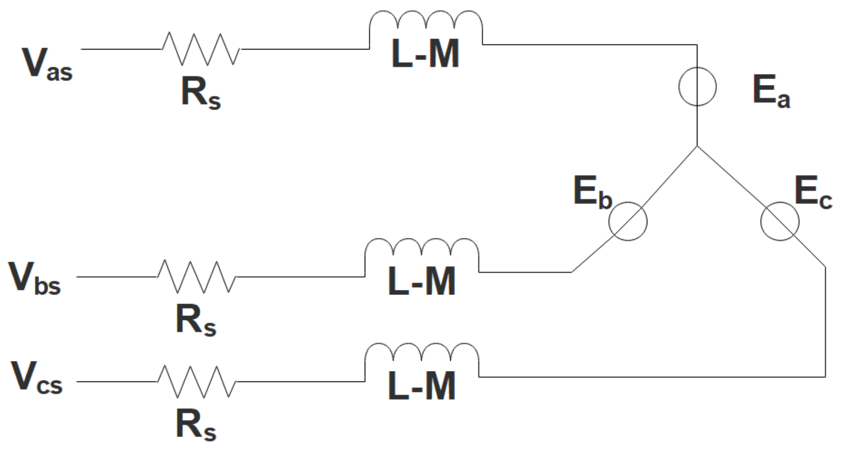


Figure 4.3: Brushless DC motor equivalent circuit (adopted from [145])

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 [145] and are related by:

$$\hat{f}_b(\theta) = \hat{f}_a(\theta - 120°), \hat{f}_c(\theta) = \hat{f}_a(\theta + 120°) \tag{4.5}$$

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.

4.3.4 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 [146]. For the quadcopter system, the system dynamics is defined in subsection 4.3.2. We have the following as dynamics for roll, pitch and yaw angles [147]:

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

where, $[U_2, U_3, U_4]^T$ are individual components of the external torque vector τ and J_{TP} is angular momentum. All other quantities are explained in subsection 4.3.2. In the nominal case, the values of ϕ, θ, ψ and ω 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:

$$\ddot{z} = -g + \cos\phi\cos\theta\frac{U_1}{m} \tag{4.6}$$

where, $U_1 = k(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$ and other terms are described in subsection 4.3.2. Similar to the design of previous PID controller, the values of ϕ and θ 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 [147].

4.4 Hierarchical Checking Approach

4.4.1 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).

4.4.2 Checking methodology: state space checks

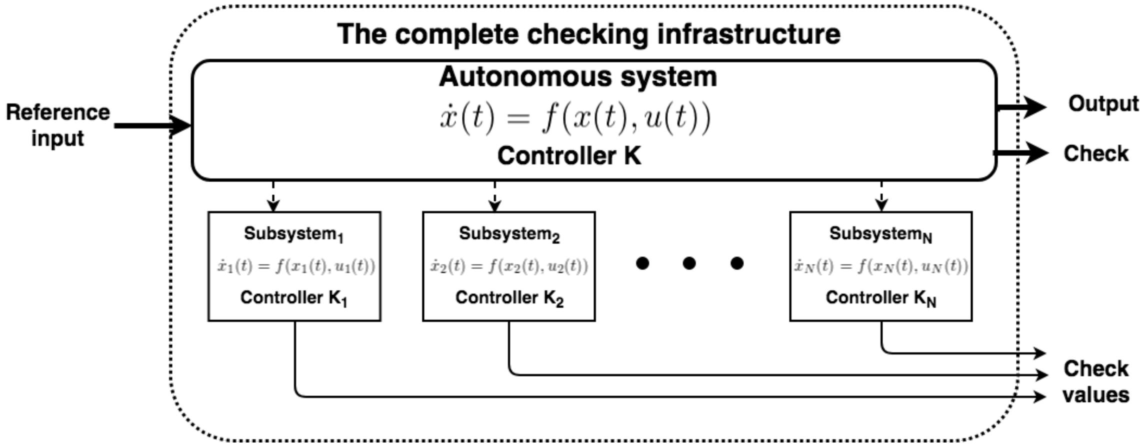

The complete checking infrastructure
Autonomous system
$\dot{x}(t) = f(x(t), u(t))$
Controller K
Reference input
Output
Check
Subsystem$_1$
$\dot{x}_1(t) = f(x_1(t), u_1(t))$
Controller K$_1$
Subsystem$_2$
$\dot{x}_2(t) = f(x_2(t), u_2(t))$
Controller K$_2$
Subsystem$_N$
$\dot{x}_N(t) = f(x_N(t), u_N(t))$
Controller K$_N$
Check values

Figure 4.4: Block diagram

In this subsection, we describe how hierarchical checks are employed to detect and diagnose multiple faults in quadcopter subsystems. A block diagram of an autonomous system is shown in Figure 4.4. Here, the system is depicted as a hierarchical interconnection of subsystems, which may have their own controllers. Figure 4.4 shows a block diagram of the proposed hierarchical error checking methodology. At the highest level, the observable states of the autonomous system and its controller K are monitored and checked via the mechanism described in Figure 4.6. The states of each subsystem$_i$ and their respective controllers K_i are checked hierarchically using the mechanism of Figure 4.6. Respective check signals $check_i$ are generated as shown in Figure 4.4. Such a hierarchical decomposition of checks allows ease of failure diagnosis down to individual subsystems while allowing multiple failures to occur simultaneously without compromising detectability.

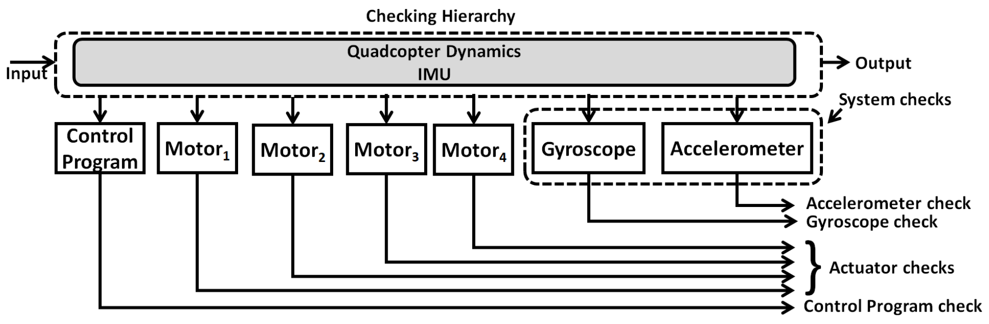


Figure 4.5: Hierarchical checking methodology

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 4.5. 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 4.1 summarizes the diagnosis strategy.

Table 4.1: Diagnosis summary

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

Sequential model based state checking

Figure 4.6 shows the generation of the implemented state space based check for a nonlinear state variable system [83]. Each column within the dashed block of Figure 4.6 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 [83]. 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 $s(t-1), s(t-2), \ldots, s(t-t_W)$ 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 [148] which is explained in

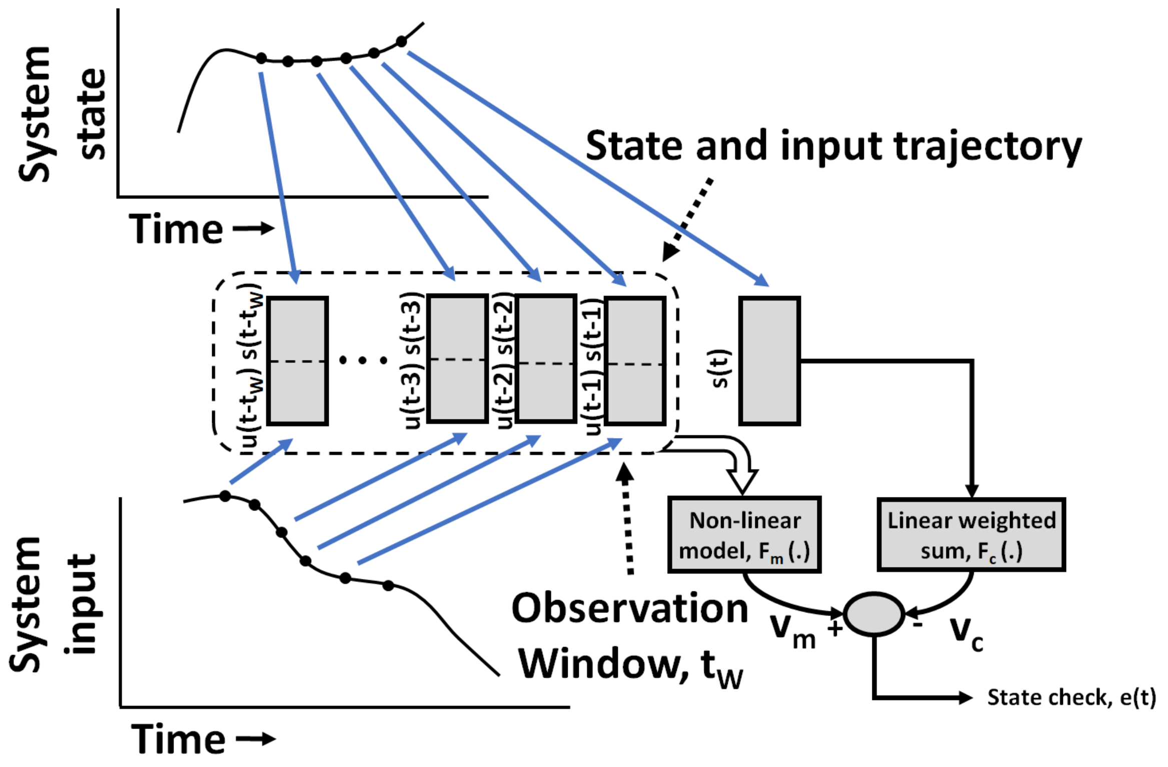


Figure 4.6: Computation of state space based check

subsection 4.4.2.

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 4.6) 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 [148] and are based on feedback of the output of a perceptron to its input, and they have been successfully used in various applications [149–156]. 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)* [157], and is used in this research. A block diagram of an LSTM cell is shown in Figure 4.7 where important 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

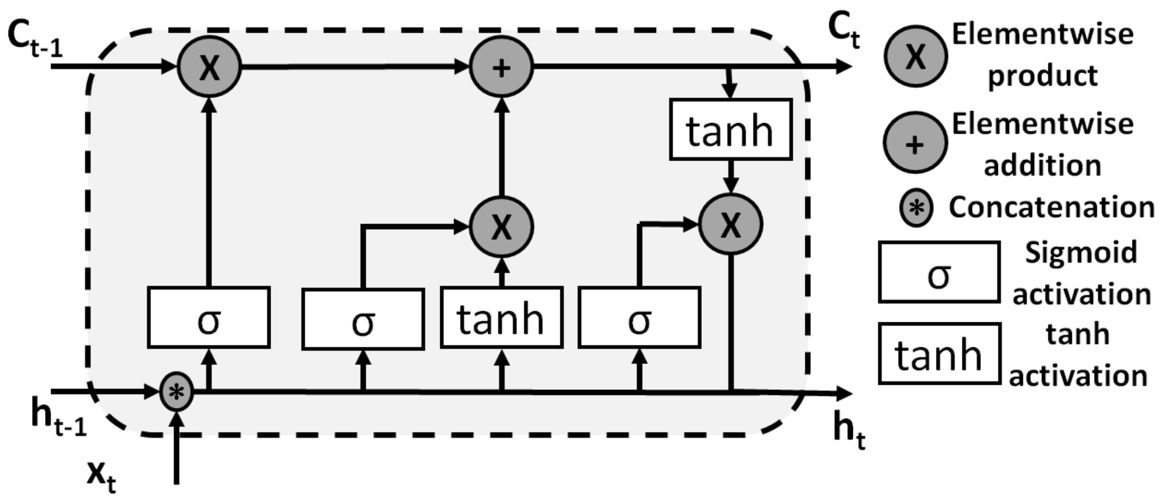


Figure 4.7: Long Short Term Memory Block

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 [157].

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 4.8 shows 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 4.8 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

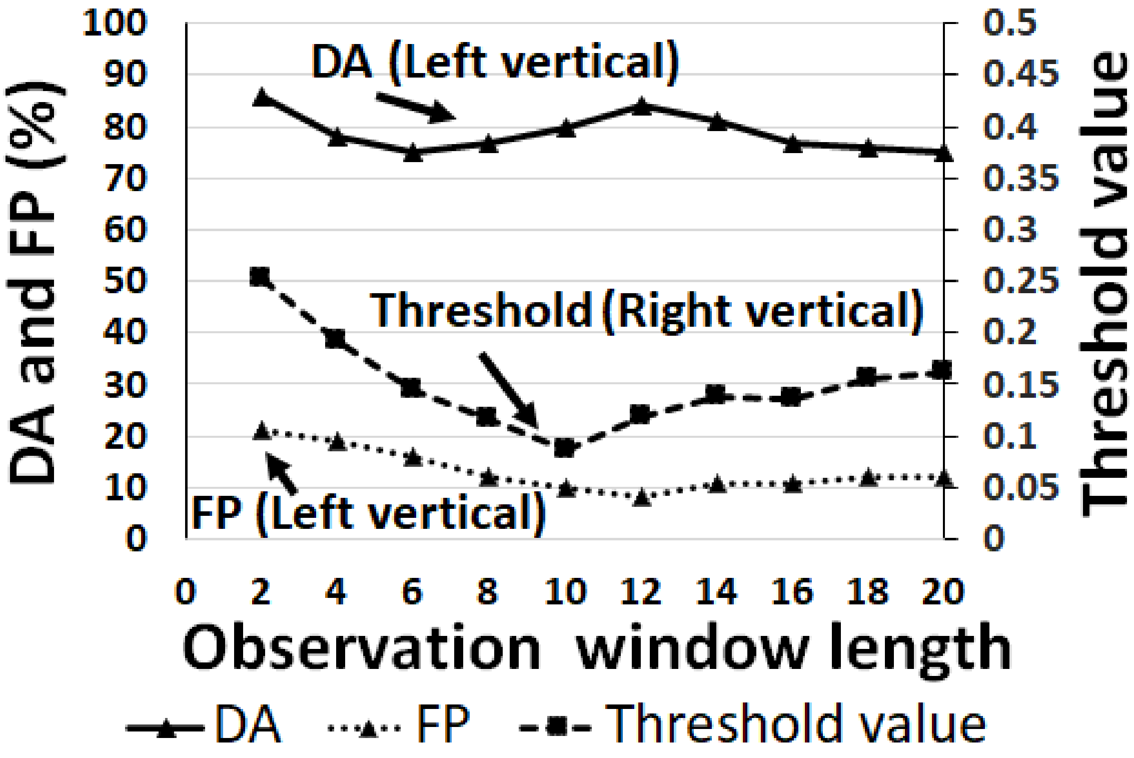


Figure 4.8: Trade-off between observation window and accuracy for $\sigma^2_{noise} = 0.05$

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 4.8, 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.

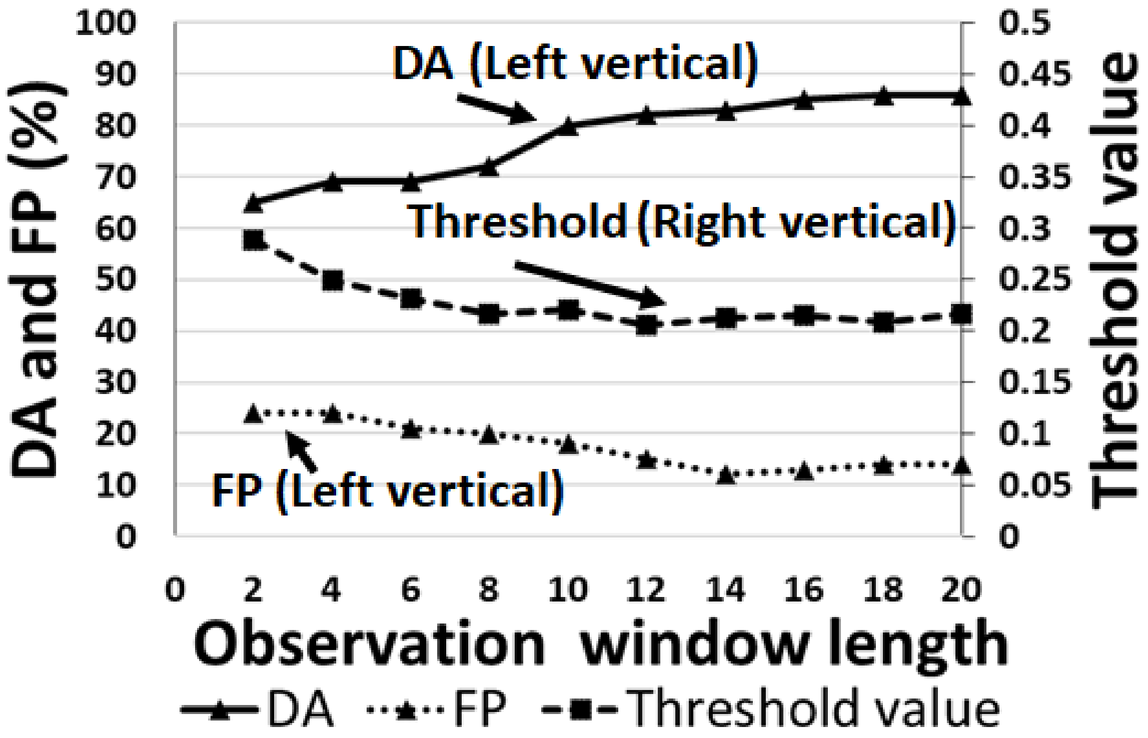


Figure 4.9: Trade-off between observation window and accuracy for $\sigma^2_{noise} = 0.25$

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 4.9, 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 4.8, it is observed that for prediction window length of 10, the $DA \geq 80\%$ and $FP \leq 10\%$ 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.

4.4.3 Hierarchical check infrastructure for the quadcopter system

According to the study conducted in Section 4.4.2, 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.

Control program check

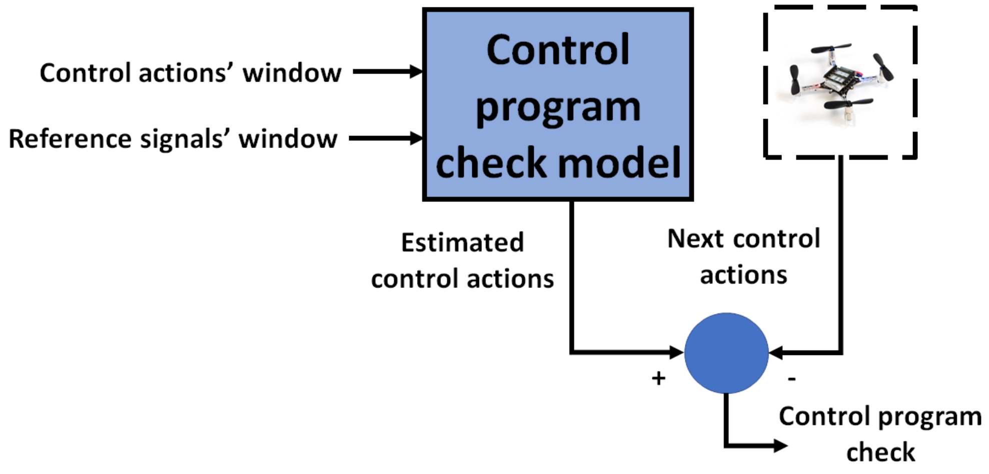

Control actions' window
Reference signals' window
Control program check model
Estimated control actions
Next control actions
+
-
Control program check

Figure 4.10: 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 4×1.

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 3×1. So, the control program check's inputs and outputs sizes are 7×1 and 4×1 respectively. If the ensemble control actions are denoted by $U(t)$ $(= [U_1, U_2, U_3, U_4]^T$, defined in subsection 4.3.4), then the control program check, chk_{ctrl} is defined as $chk_{ctrl} = U(t)_{predicted} - U(t)_{actual}$, and it is of size 4×1.

Sensor check

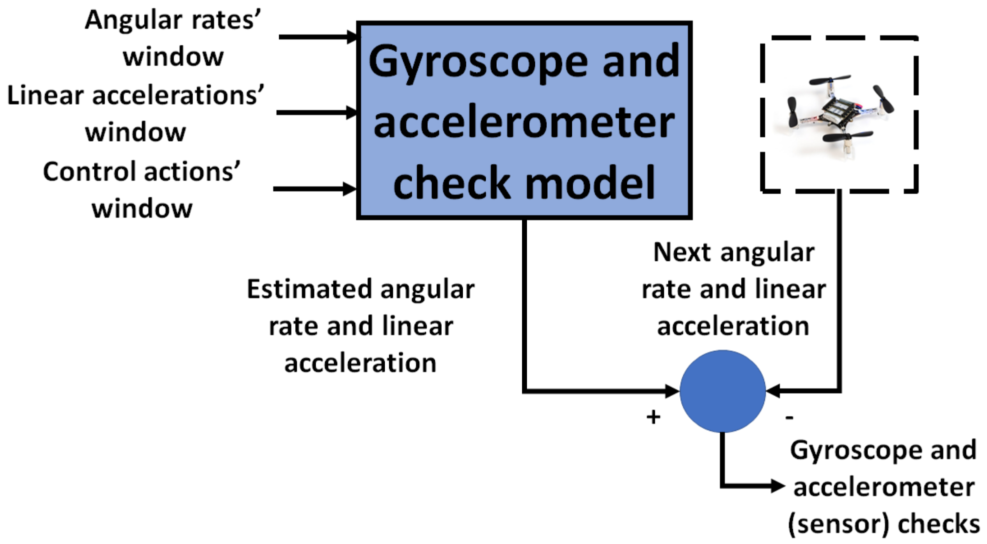

Angular rates' window
Linear accelerations' window
Control actions' window
Gyroscope and accelerometer check model
Estimated angular rate and linear acceleration
Next angular rate and linear acceleration
+
-
Gyroscope and accelerometer (sensor) checks

Figure 4.11: Sensor (gyroscope and accelerometer) 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 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 3×1. In this work, they have been merged together to share model resources efficiently, and hence they are considered as one unit.

Actuator check

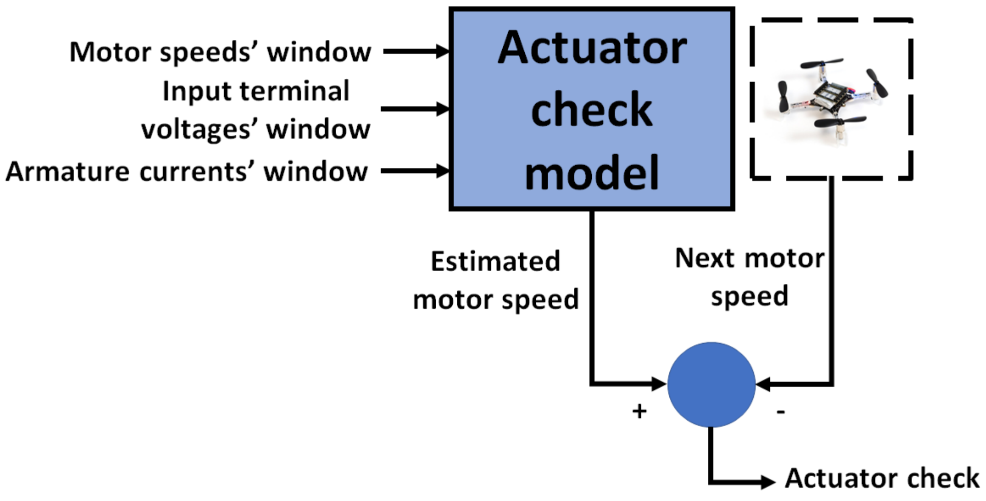


Figure 4.12: Actuator check

For the actuator check, the LSTM based machine learned model receives the windows of motor speed Ω, 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.

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

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' [158]. 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.

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 μs 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.

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 4.4.2, 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 [159] is performed to estimate the changed parameter (line 22 in Algorithm 1), p which is defined as:

$$\begin{aligned} p^* &= \arg\min_p \left(\|chk_{observed}(t_i) - chk(act(p, t_i))\|_2\right) \\ &\text{subject to, } 0 < t_i \leq 55 \end{aligned} \tag{4.7}$$

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 [146]. 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).

4.4.5 Simulation Experimental Result

A quadcopter with the following design parameters were considered: $L = 0.3\,\mathrm{m}$, $r = 0.1\,\mathrm{m}$, $m = 1.2\,\mathrm{kg}$, and $b = 0.0245$, propeller diameter $d = 10\,\mathrm{inch}$, and propeller pitch $= 4.5\,\mathrm{inch}$. 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 μs 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 5.6.7). 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 [84].

Error Diagnosis

We will briefly discuss the error diagnosis performance in this section. Figures 4.13 and 4.14 show the check error plots when an alpha particle strike fault is introduced in the computation core of the system at time $t = 1.5$ sec.

As can be observed these plots, the sensor check. which is the higher level check, produces a non-zero signal at time $t = 1.5$ sec representing the existence of the failure. Once the failure is detected, one can investigate Figures 4.14a and 4.14b. Here, it is observed that the control check produces a non-zero signal, and actuator check is producing zero signal. From these plots, it can be concluded that the failure was introduced at Control program execution. Similar type of experiments are performed for failures in other blocks and Table

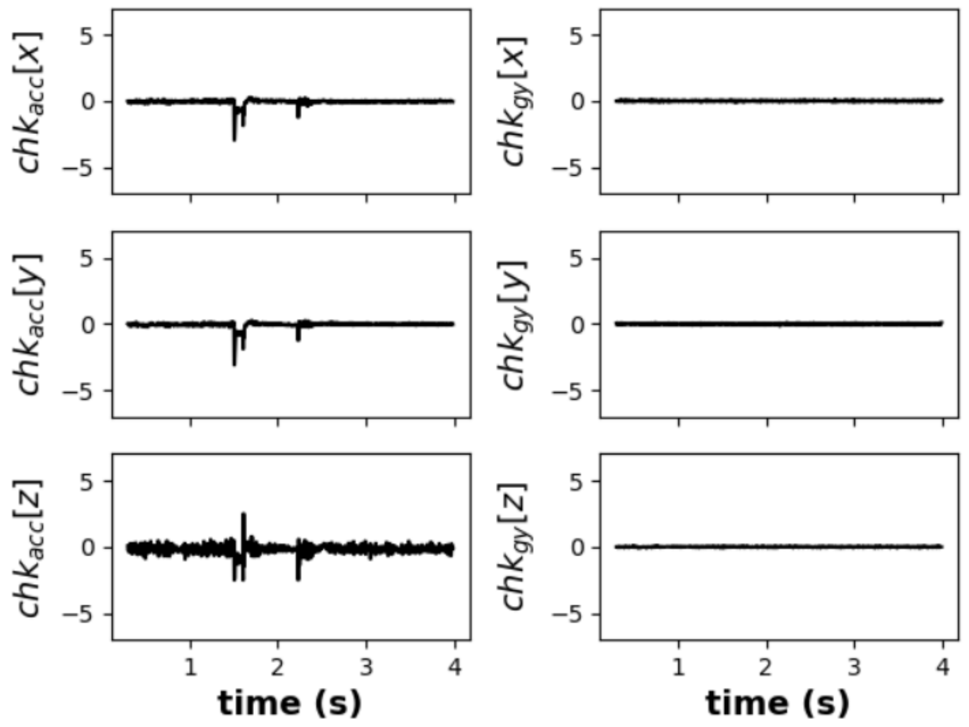
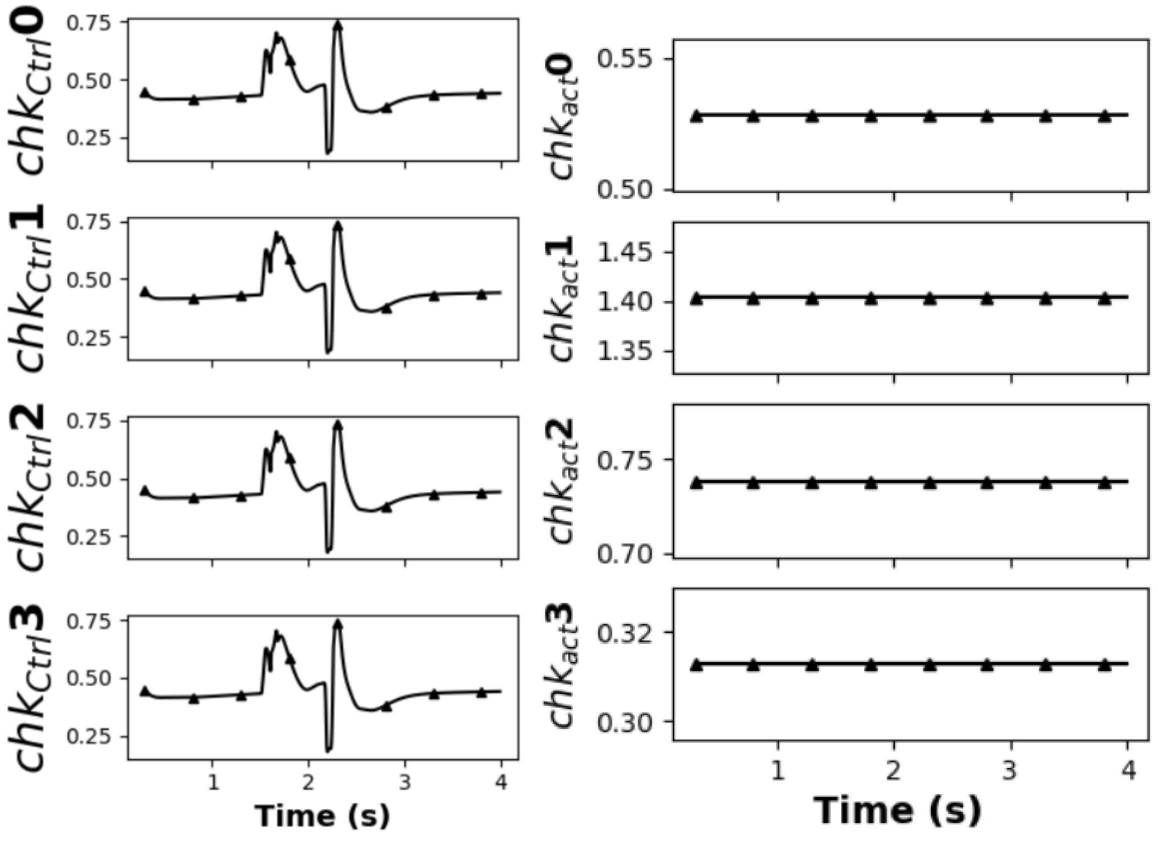


Figure 4.13: Sensor check in presence of control program failure

4.1 was constructed.

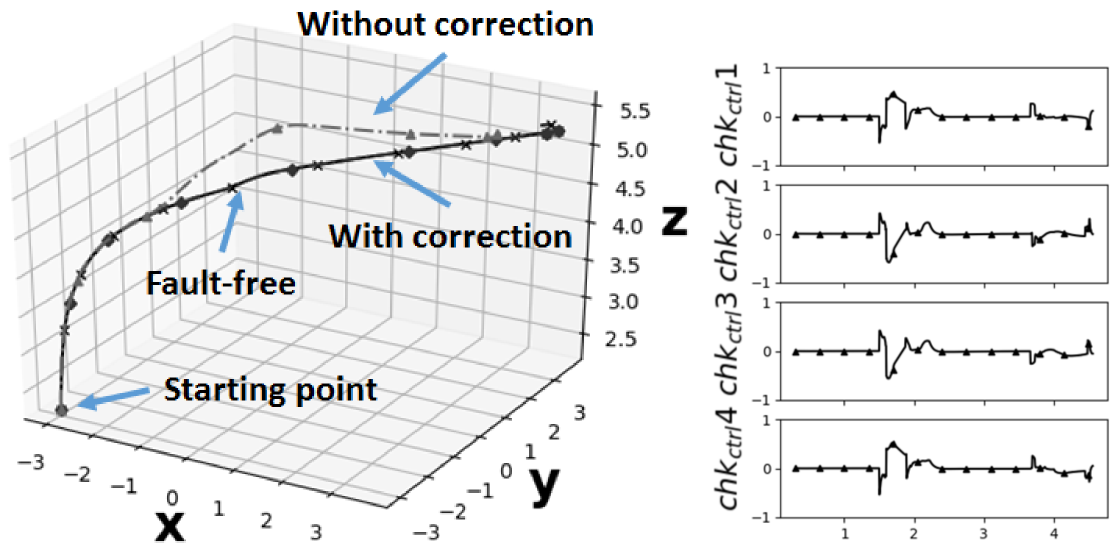
Errors in Execution of control Program

Figure 4.15 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 4.15a, 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 4.15b, the faults have been 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 4.2. This shows the improvement and hence the efficacy of the proposed approach.



(a) (b)

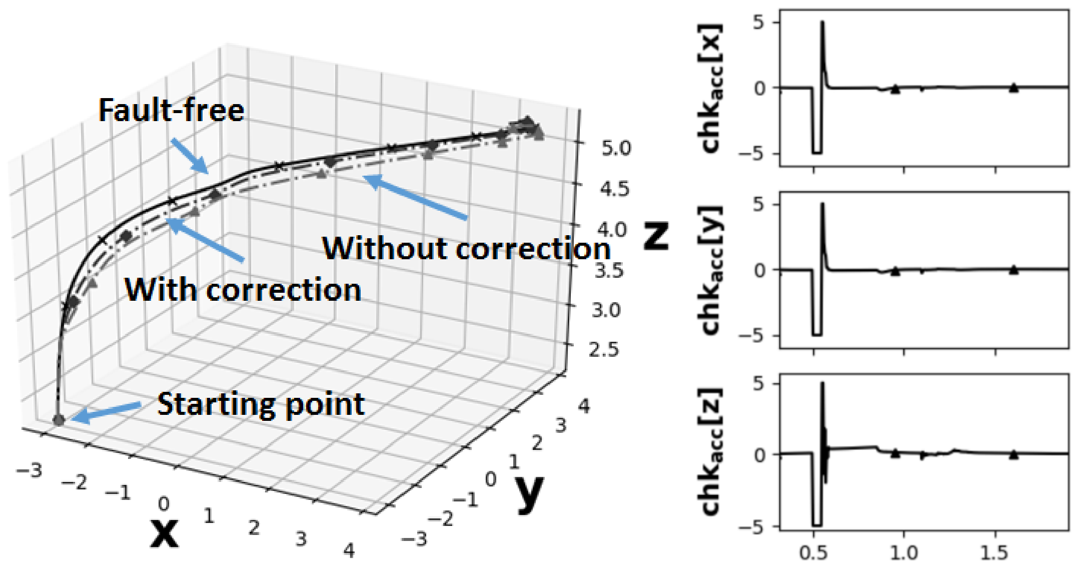
Figure 4.14: a) Control check for control program fault, and b) actuator check plot in presence of control program failure



(a) (b)

Figure 4.15: a) Trajectories of quadcopter for control program fault, and b) Corresponding control program check plot

Sensor Transient Errors: Accelerometer



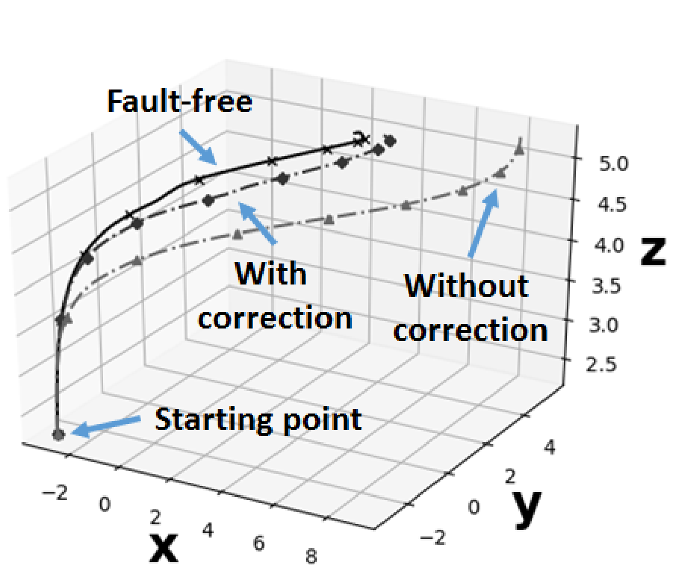
(a) (b)

Figure 4.16: a) Trajectories of quadcopter for accelerometer transient fault, and b) Corresponding accelerometer check plot

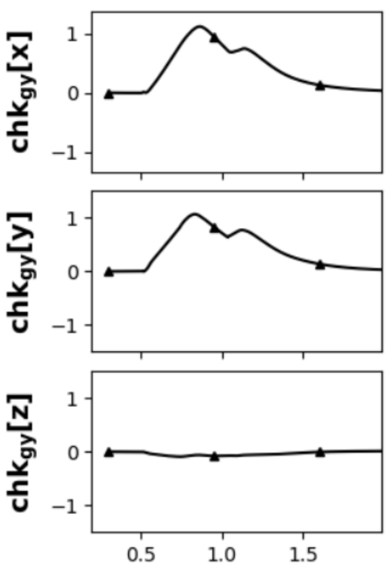
Proposed approach was applied to transient error injected in accelerometer sensor as 2nd experiment. A transient error was injected at $t = 0.5$ sec, as a form of spurious bit-flips, at accelerometer sensor reading which changed the trajectory of the quadcopter as observed in Figure 4.16a. It can also be observed from Figure 4.16b 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 4.2.

Sensor Transient Errors: Gyroscope

Figure 4.17 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 4.17b, and the trajectory is changed to a great extent when no correction is performed. However, after applying the proposed methodology, it was



(a)



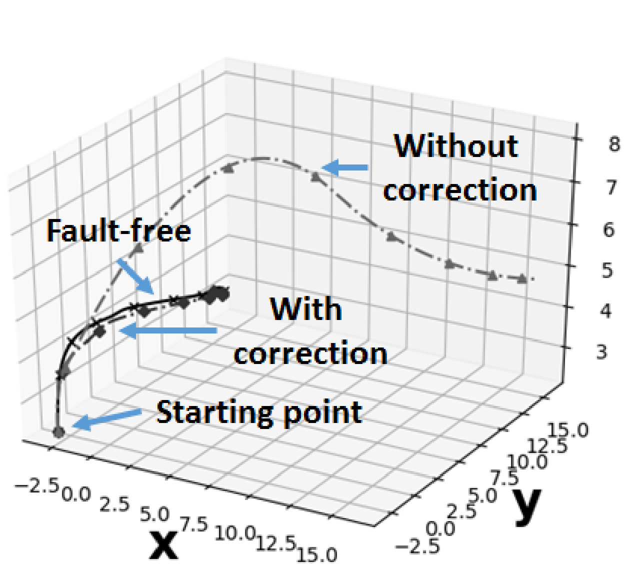
(b)

Figure 4.17: a) Trajectories of quadcopter for gyroscope transient fault, and b) Corresponding gyroscope check plot

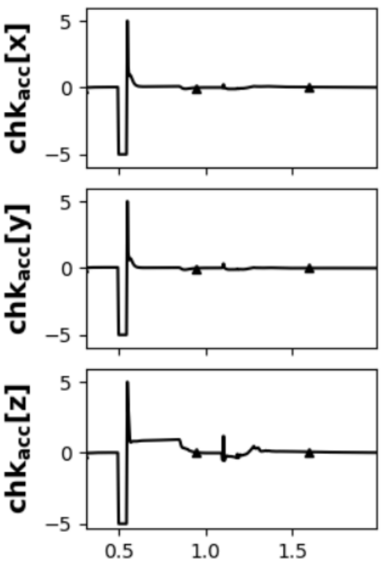
possible to improve the trajectory of the system. The comparison of v_1 and v_2 are given in Table 4.2. As can be seen from these trajectories and the table, the corrected trajectory was very close to reference one.

Parameter Deviations in Sensors: Accelerometer

Figure 4.18 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 4.18a. 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 4.2. Here, the machine learning models were trained under closed loop configuration where the system behavior was moni-



(a)



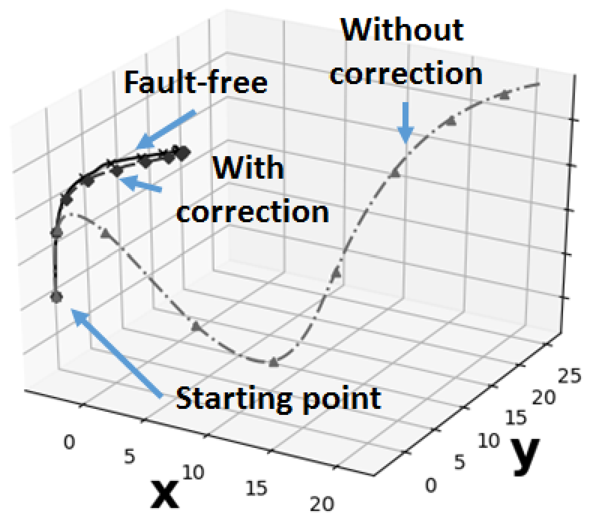
(b)

Figure 4.18: a) Trajectories of quadcopter for accelerometer parametric deviation, and b) Corresponding accelerometer check plot

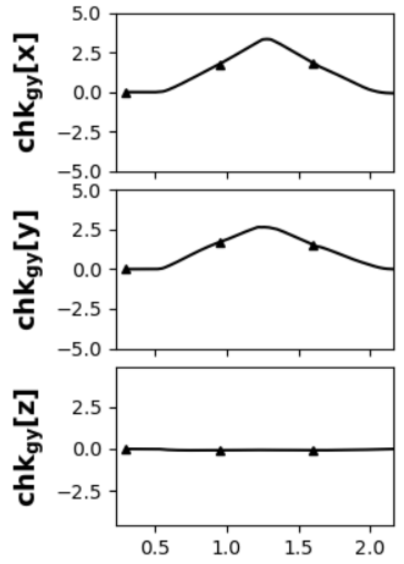
tored and observed behavior was utilized to train the model. During the deployment phase, the behaviors were validated with respect to learned model. In this situation, they are operating in the same manner like in the training phase. Additionally, this corrected information is processed further in the system and these information are used in future time-steps. In this way, they are working in closed loop.

Parameter Deviations in Sensors: Gyroscope

We investigated parametric variation of gyroscope as the fifth study which is shown in Figure 4.19. 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 4.2.



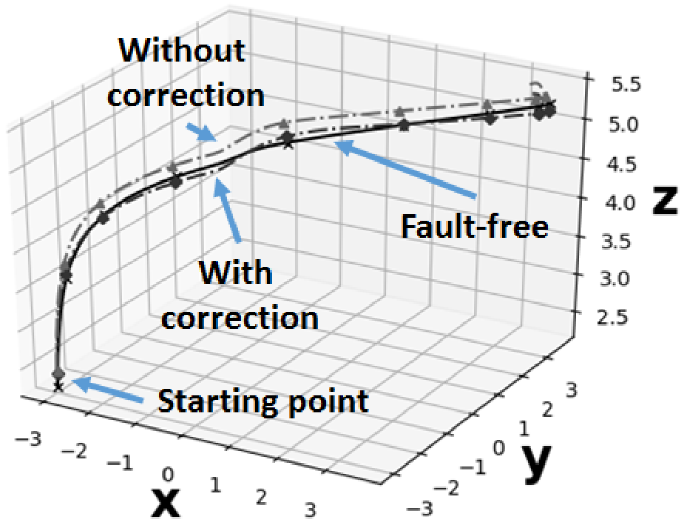
(a)



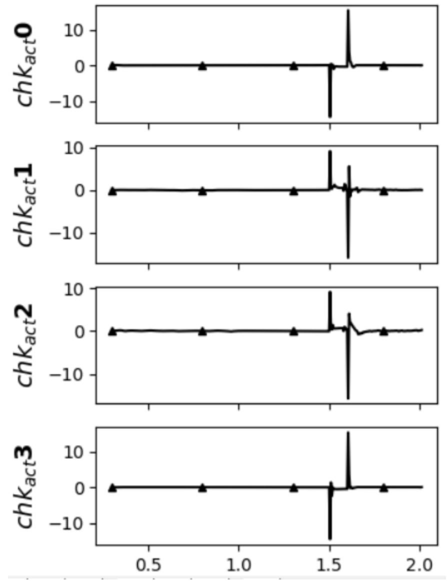
(b)

Figure 4.19: a) Trajectories of quadcopter for gyroscope parametric deviation, and b) Corresponding gyroscope check plot

Parametric Deviations in Actuators



(a)



(b)

Figure 4.20: 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 4.20b) which detects and diagnoses the source of the fault. An optimization as described in Equation 5.38 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 4.20. As seen from Table 4.2, the L2 norm of trajectory improves when corrective action is employed.

Summary of Experimental Results

Table 4.2: 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 4.2 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, the proposed approach was able to correct different kinds of failures with very high accuracy which clearly shows the efficacy of the approach.

4.4.6 Hardware Experimental Result

We implemented the proposed checking methodology on a '*Crazyflie 2.1*' [142] 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 5.7. The quadcopter communi-

Table 4.3: Hardware configuration of Crazyflie 2.1

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

cates 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 cy-

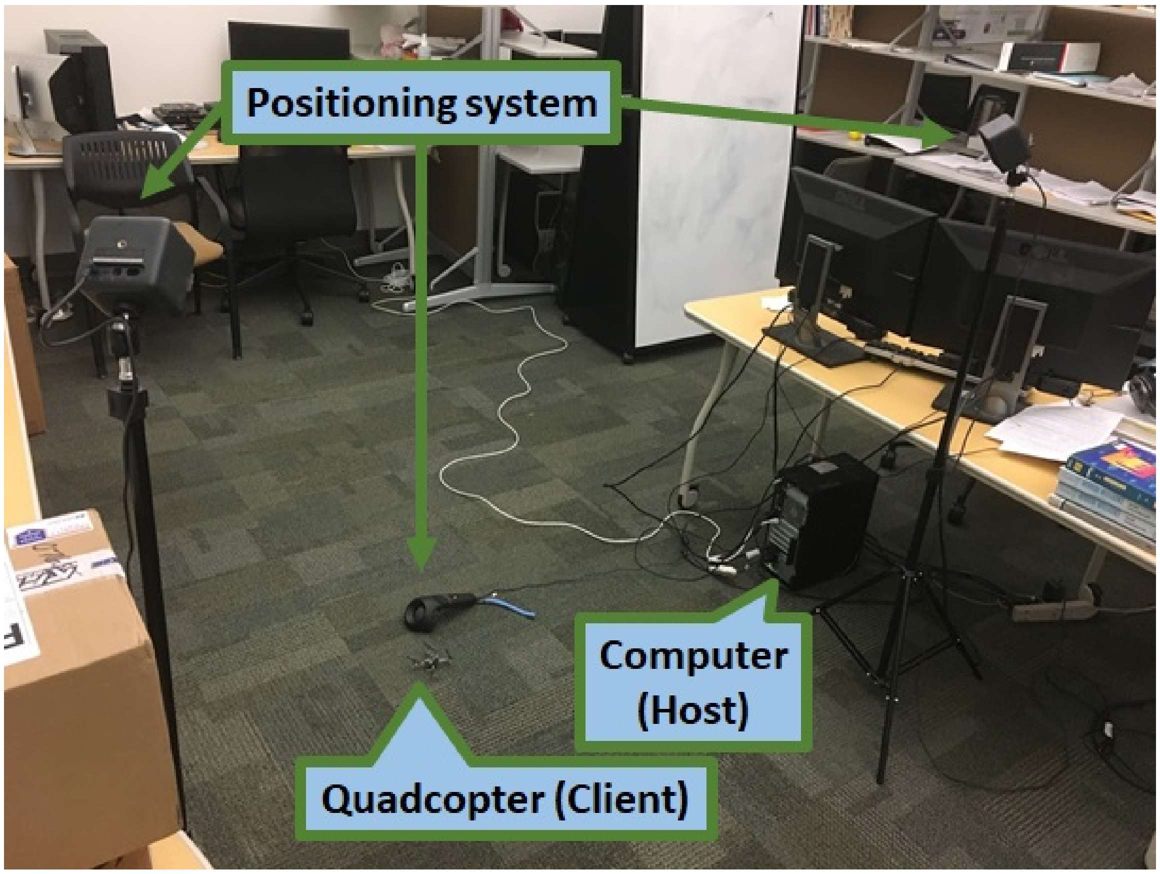


Figure 4.21: Hardware setup

cle available, and for this reason, model and code optimization for our proposed approach was necessary which is described below:

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) [160] which has less parameter (less computation overhead and simpler) with almost same accuracy, and has use in multiple different applications which includes sentiment analysis [161], stock market and financial institution [162, 163], energy dissipation [164], security attacks [165] and biomedical [166] and speech applications [167]. 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 4.4.2 to come up with the optimum model (the average mean square error of estimation was less than 5×10^{-3}), and finally, a generalized model architecture given in Figure 4.22 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.

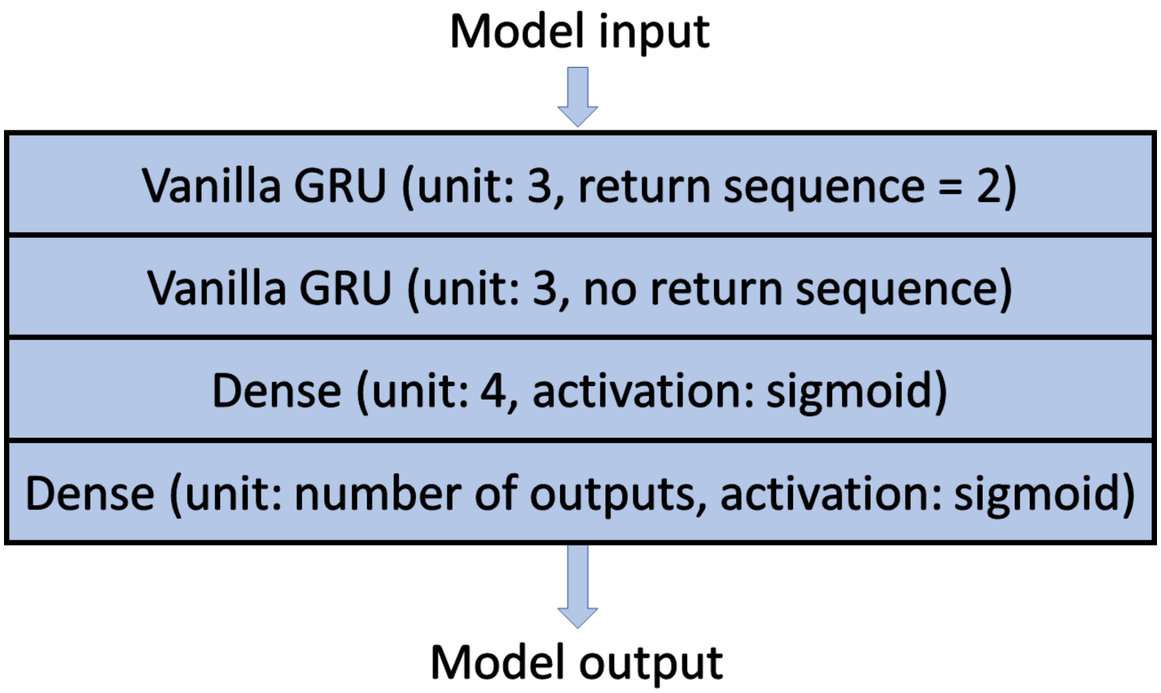

Model input
Vanilla GRU (unit: 3, return sequence = 2)
Vanilla GRU (unit: 3, no return sequence)
Dense (unit: 4, activation: sigmoid)
Dense (unit: number of outputs, activation: sigmoid)
Model output

Figure 4.22: 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:

Control program fault

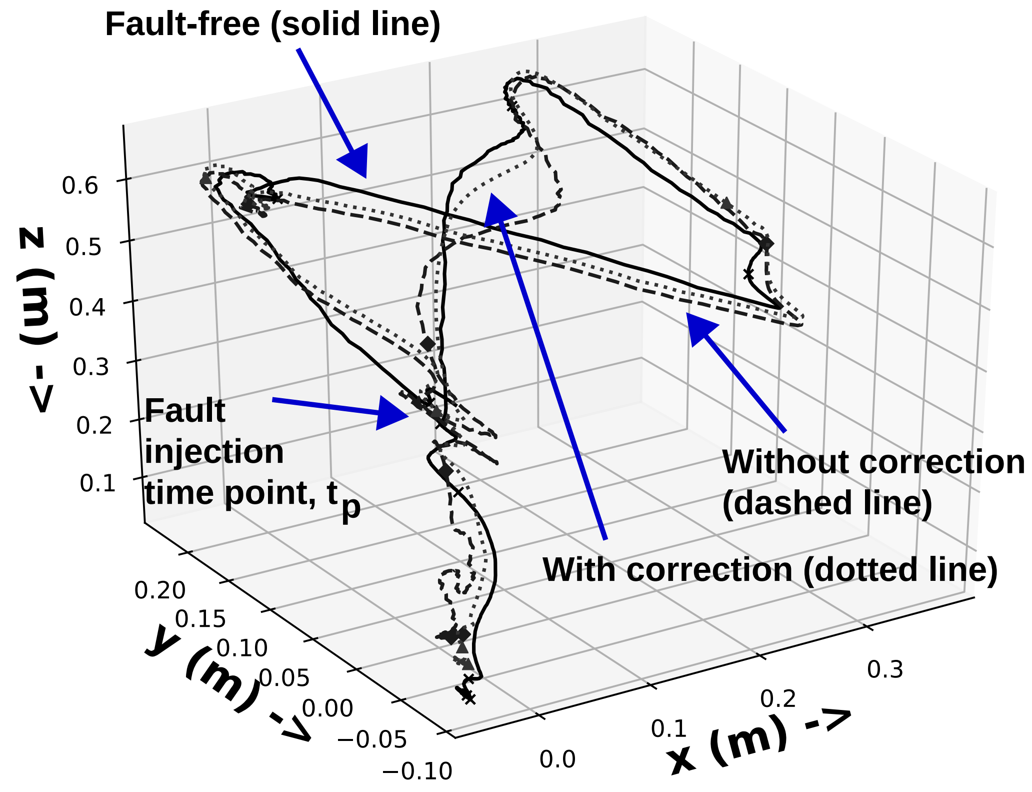


Figure 4.23: Quadcopter trajectory in presence of control program fault

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 5.35. These plots show that after '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.

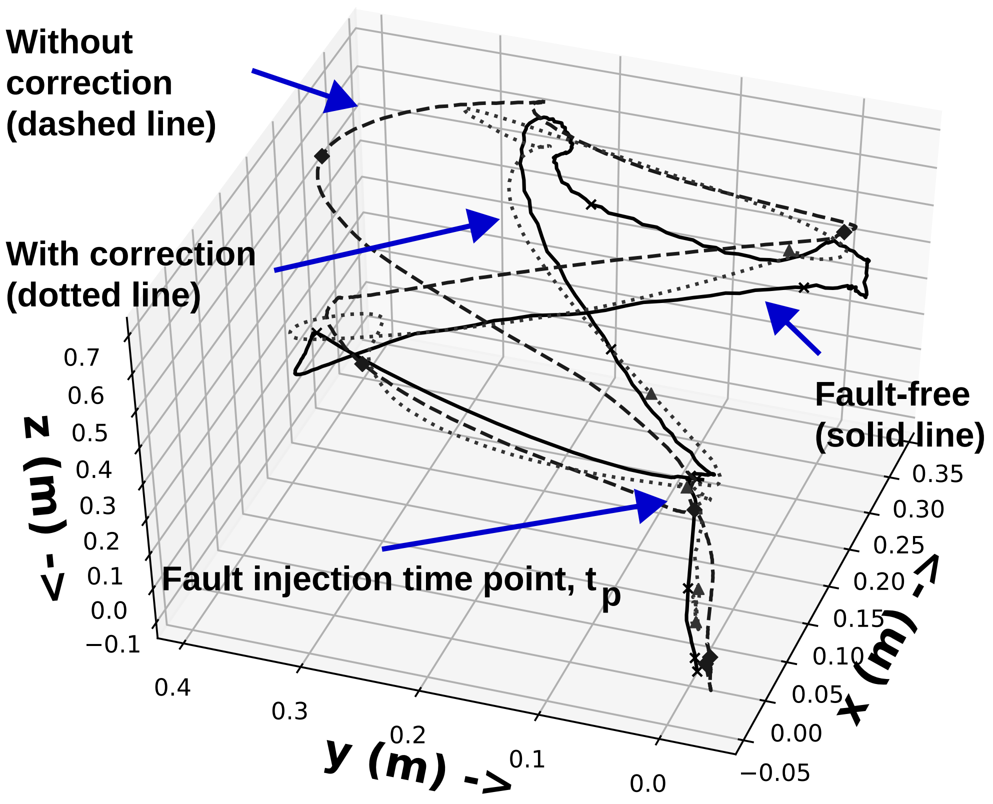


Figure 4.24: Quadcopter trajectory in presence of sensor fault

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 4.24. 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.

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.

During the deployment, we performed an experiment (see Figure 5.32) 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 4.26 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

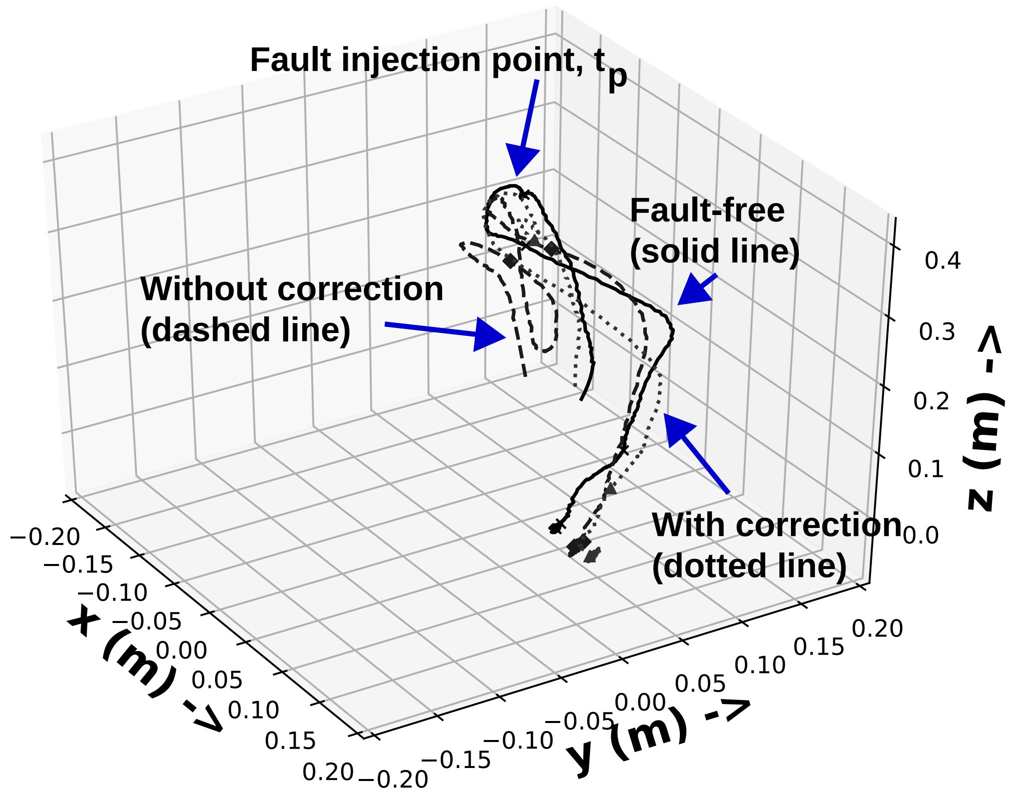


Figure 4.25: Quadcopter trajectory in presence of actuator fault

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 4.26). 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

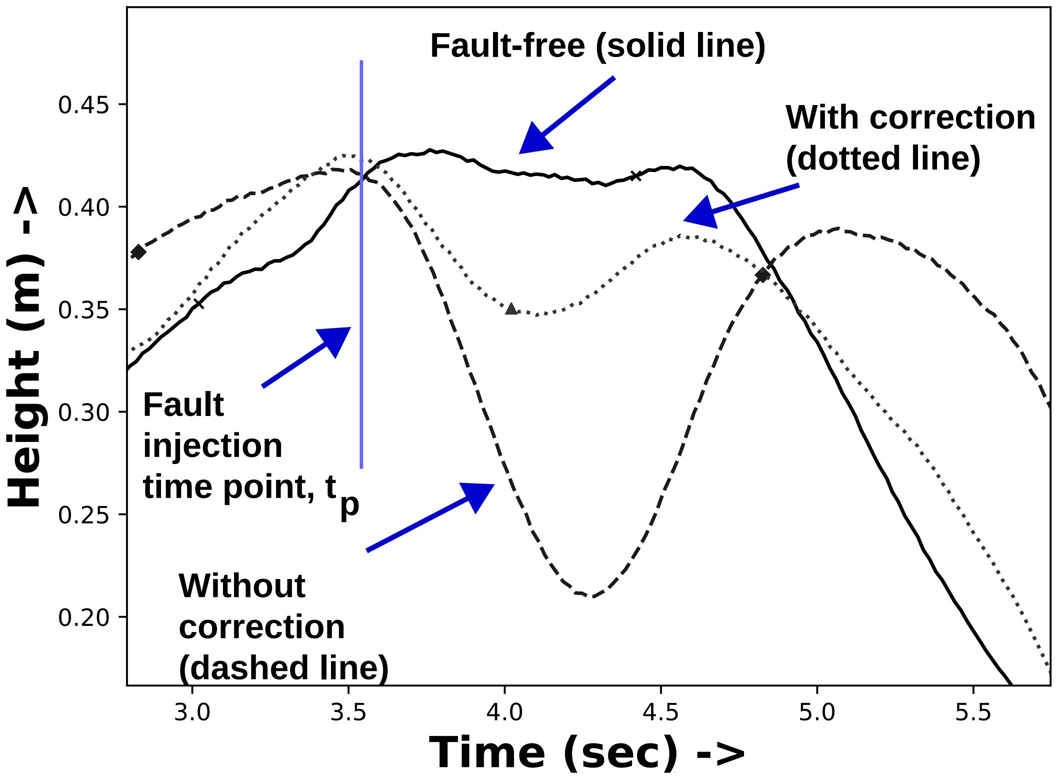


Figure 4.26: Controller compensation effect observed from height due to actuator fault parameter, the behavior of the system was improved.

4.4.7 Summary

In this work, 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.

CHAPTER 5
CONCURRENT ERROR DETECTION IN EMBEDDED DIGITAL CONTROL OF NONLINEAR AUTONOMOUS SYSTEMS USING ADAPTIVE STATE SPACE CHECKS

5.1 Introduction and Key Contributions

The increased deployment of autonomous vehicles and drones has raised questions regarding the safety and trustworthiness of such systems [168]. Risks in the real-time operation of such nonlinear systems can arise from errors and failures in sensors, actuators and runtime execution of control software [82, 169] due to field stress, part wearout and hostile operating conditions. Such risks, if manifested during real-time system operation can result in abnormal behavior of the system or that of its constituent subsystems (electrical, electro-mechanical, sensors, actuators and relevant controllers). Abnormal behaviors which are *different from nominal system function* are broadly called *anomalies*, and need to be detected with high coverage and low latency to enable quick recovery. We are particularly concerned with applications such as aerial drone maneuvers and subsystems of autonomous vehicles where the *latency of anomaly detection must be extremely small* (fractions of a second) to minimize the risk of accidents. In this context, the key contributions of this research are the following:

5.1.1 Key Contributions

The key contributions of this research are provided below:

1. A new methodology for checking anomalous behaviors in subsystems (sensors, actuators, state estimation algorithms and control program execution) of *nonlinear autonomous systems using encoded Extended Kalman Filter (EKF)*, is developed. The

method is scalable to a variety of nonlinear systems, incurs minimal computation overhead and latency and requires no machine learning as compared to [82]. For the first time, state encoding based checks are introduced *both for checking the state transition as well the covariance matrix computation steps* of the Extended Kalman Filter.

2. As opposed to prior checking schemes with *time-invariant coding schemes and fixed detection thresholds*, the state transition behavior changes from one time step to the next in generalized nonlinear systems (as given by the Jacobian of the state transition function). For this reason, prior encoding techniques *cannot be directly applied to Extended Kalman filter (EKF)*. This fundamental problem is solved in this research by employing a *time-varying encoding scheme with variable detection thresholds*. This further allows the checks to adapt to changing performance sensitivities to error/failure conditions under diverse maneuvers of autonomous systems (e.g. hovering vs. sharp turns for quadcopters), achieving higher error coverage than possible with prior techniques with significantly lower computational overhead. Results on two test cases are demonstrated: a quadcopter and a steer-by-wire system for autonomous vehicles.

3. A machine learning facilitated error diagnosis and correction framework is developed. Error at sensor is compensated by performing online parameter estimation and correction. Errors at actuator parametric failure and controller were compensated by reconfiguring the controller by reinforcement learning and by restoring last safe action, respectively.

5.2 Preliminaries: Extended Kalman Filter

Figure 5.1 shows the block diagram of a nonlinear state variable system. Let us assume that the system has n states $x(t)$, m outputs $y(t)$, p inputs $u(t)$, and q measurements $z(t)$.

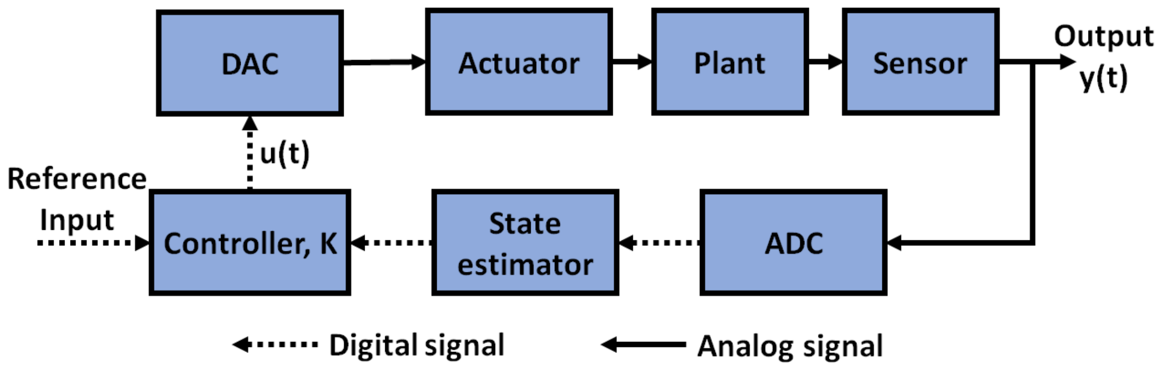


Figure 5.1: A nonlinear state space control system

i.e. $x(t) \in \mathcal{R}^n$, $y(t) \in \mathcal{R}^m$, $u(t) \in \mathcal{R}^p$, and $z(t) \in \mathcal{R}^q$. The plant behavior of this system is expressed in the form of an ordinary differential equation:

$$\dot{x}(t) = f(x(t), u(t)) + w(t) \tag{5.1}$$

where the vectors $x(t) \in \mathcal{R}^n$ and $u(t) \in \mathcal{R}^p$ represent the system states and inputs, respectively. The function $f(.)$ defines the system dynamics and $w(t) \in \mathcal{R}^n$ represents zero-mean process noise. The output $y(t) \in \mathcal{R}^m$ of the plant is given by,

$$y(t) = h(x(t), u(t)) + v(t) \tag{5.2}$$

where $h(.)$ represents an output mapping and $v(t) \in \mathcal{R}^q$ represents zero-mean measurement noise. For both linear and nonlinear state variable systems, the input $u(t)$ is computed by an external controller K that strives to maintain system performance under dynamically changing plant conditions. In this work, we assume that the states and measurements follow Gaussian statistics. Check computation for states and measurements with non-Gaussian statistics is addressed in [170].

In general, due to limited observability of all the internal states of the plant, a recursive state estimator such as the Extended or Unscented Kalman filter is used to estimate the plant states in the presence of measurement noise. In partially observable nonlinear state variable systems, the Extended Kalman Filter (EKF) [171] is used to estimate the observable as well as the unobservable states of the system in the presence of measurement noise under the assumption that the system states have Gaussian distributions. The EKF assumes a system model of the form of Equations 5.1 and 5.2. The nonlinear functions $f(.)$ and $h(.)$ are linearized at discrete time instants k and are denoted by the Jacobian matrices $F_k \in \mathcal{R}^{n\times n}$ and $H_k \in \mathcal{R}^{m\times n}$, respectively. The estimated state and covariance matrices are represented by $\hat{x} \in \mathcal{R}^n$ and $P \in \mathcal{R}^{n\times n}$, respectively. $Q_k \in \mathcal{R}^{n\times n}$ and $R_k \in \mathcal{R}^{q\times q}$ represent covariance matrices for the process and observation noise, respectively. $z_k \in \mathcal{R}^q$ is a vector of measurements performed on the system and corresponds to discretized values of the measured outputs of the system $y(t) \in \mathcal{R}^m$ of Figure 5.1. In the following, $\hat{x}_k^- \in \mathcal{R}^n$ and $P_k^- \in \mathcal{R}^{n\times n}$ represent the state estimate and covariance matrix at time instant k based on observations up to time instant $k-1$. Similarly $\hat{x}_k \in \mathcal{R}^n$ and $P_k \in \mathcal{R}^{n\times n}$ represent corresponding estimates based on observations up to time instant k.

The EKF works in a two major part at every time instant k [172], namely - prediction and correction. During the *prediction* as the Step 1, the states and covariance matrix are predicted by

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}, u_k) \\ P_k^- &= F_k P_{k-1} F_k^T + Q_k \end{aligned} \tag{5.3}$$

on the basis of knowledge of $\hat{x}_k^-$, u_k, F_k, P_k^- and Q_k. The predicted state and covariance matrices are denoted by x_k^{prior} and P_{prior}, respectively.

In the *update* as Step 2, the measurements z_k are used to determine the state residue as

$$\tilde{y}_k = z_k - h(\hat{x}_k^-) \tag{5.4}$$

Once the residue is computed, the covariance residue S_k and the Kalman gain K_k are computed in Step 3 as

$$\begin{aligned} S_k &= H_k P_k^- H_k^T + R_k \\ K_k &= P_k^- H_k^T S^{-1} \end{aligned} \tag{5.5}$$

Using these quantities, the state and covariance matrix are updated using the following equations in Step 4:

$$\begin{aligned} \hat{x}_k &= \hat{x}_k^- + K_k \tilde{y}_k \\ P_k &= (I - K_k H_k) P_k^- \end{aligned} \tag{5.6}$$

5.3 Proposed Approach: State Encoding Based EKF Checks

5.3.1 Encoding property of matrix: an overview and its application to EKF

We review algorithm based encoding of matrices as first described in [25, 26, 35]. Consider the information matrix A, where $A \in \mathcal{R}^{m\times n}$. Let us define two vectors, namely $CV_1 \in \mathcal{R}^{1\times m}$ and $CV_2 \in \mathcal{R}^{1\times n}$. The column, row, and full weighted checksum matrix A_c, A_r, and A_f, of the matrix A are defined as [25]:

$$\begin{aligned} A_c &= \left[\begin{array}{c} A \\ \hdashline CV_1.A \end{array}\right] \\ A_r &= \left[\begin{array}{c:c} A & A.CV_2^T \end{array}\right] \\ A_f &= \left[\begin{array}{c:c} A & A.CV_2^T \\ \hdashline CV_1.A & CV_1.A.CV_2^T \end{array}\right] \end{aligned} \tag{5.7}$$

As shown in [25], it can be seen that each of these weighted checksum matrices has its separate information matrix and check vectors which can be used to ensure fault tolerance.

The matrix is stored in its full weighted checksum matrix format and is manipulated in the full, row, or column weighted checksum matrix format, on the matrix. After computations are performed, the information sub-matrix can be readily extracted from the full weighted checksum encoded matrix. There are five matrix operations which preserve the weighted checksum property: addition, multiplication, LU decomposition, transpose, and product of a matrix with a scalar [25]. They are as follows:

1. If $A + B = C$,then $A_r + B_r = C_r$, $A_c + B_c = C_c$, and $A_f + B_f = C_f$
2. If $AB = C$,then $A_cB = C_c$, $AB_r = C_r$, and $A_cB_r = C_f$
3. If $sA = C$, where s is a scaler, then $sA_r = C_r$, $sA_c = C_c$, $sA_f = C_f$
4. If $A^T = C$, then $A_r^T = C_r$, $A_c^T = C_c$, $A_f^T = C_f$
5. If $A = LU$, then $A_f = L_cU_r$

Considering these definitions and properties, different aspects of the EKF framework can be encoded for rapid error detection. During the prediction step, predicted state estimate $\hat{x}$, and predicted covariance estimate $P_{k|k-1}$ are encoded by:

$$\begin{aligned} \hat{x}_{k-1,c} &= \left[\begin{array}{c} \hat{x}_{k-1} \\ \hdashline CV.\hat{x}_{k-1} \end{array}\right] \\ P_{k,f}^- &= \left[\begin{array}{c:c} P_k^- & P_k^-.CV^T \\ \hdashline CV.P_k^- & CV.P_k^-.CV^T \end{array}\right] \end{aligned} \tag{5.8}$$

Where, CV is an appropriately sized row vector, called *Coding Vectors*, containing non-zero elements.

Next, we show how the state and covirance matrices can be encoded to detect computation errors in the Equations 5.3, 5.4, 5.5 and 5.6 of the EKF algorithm as discussed before.

The quantities of update steps (Equations 5.4, 5.5, and 5.6) are augmented in the following way:

$$
\begin{aligned}
S_{k,f} &= \left[\begin{array}{c:c} S_k & S_k.CV \\ \hdashline CV.S_k & CV.S_k.CV^T \end{array}\right] \\
K_{k,f} &= \left[\begin{array}{c:c} K_k & K_k.CV_2^T \\ \hdashline CV_1.K_k & CV_1.S_k.CV_2^T \end{array}\right] \\
\hat{x}_{k,c} &= \left[\begin{array}{c} \hat{x}_k \\ \hdashline CV.\hat{x}_k \end{array}\right] \\
P_{k,f} &= \left[\begin{array}{c:c} P_k & P_k.CV^T \\ \hdashline CV.P_k & CV.P_k.CV^T \end{array}\right]
\end{aligned} \tag{5.9}
$$

where, CV_1 and CV_2 are row vectors having size equal to the number of states and measurements, respectively.

For error detection, we propose to encode the Jacobian F_k by including an extra check state by introducing a time-varying coding vector CV_k, defined as $CV_k = [\alpha_{1k}, \alpha_{2k}, \dots, \alpha_{nk}]$, where n is the number of states of the system (the mechanism for determining the coding vector CV_k is discussed later). We augment the Jacobian of the system F_k and the system state estimate $\hat{x}_k$ in the following manner:

$$
\begin{aligned}
\hat{x}_{k,aug} &= \left[\begin{array}{c} \hat{x}_k \\ \hdashline C_k \end{array}\right] \\
F_{k,aug} &= \left[\begin{array}{c:c} F_k & 0 \\ \hdashline CV_k F_k & 0 \end{array}\right]
\end{aligned} \tag{5.10}
$$

where, CV_k is the coding vector at time k and C_k is defined as $C_k = CV_k x_k$, the inner product of CV_k and x_k. As the states' estimates can be for prior and posterior, the quantity

C_k also have two representations: $C_k^{pre} = CV_k x_k^{pre}$, and $C_k^{post} = CV_k x_k^{post}$. Following is the expanded form of the augmented jacobian $F_{k,aug}$:

$$F_{k,aug} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 \\ \vdots & \vdots & \vdots & \vdots & \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 \\ c_1 & c_2 & \dots & c_n & 0 \end{bmatrix} \tag{5.11}$$

Where, $c_i = CV_k.col_i$, and col_i is the ith column of system jacobian F_k.

After the augmentation, the linearized state variable system takes the following form:

$$\begin{bmatrix} \dot{\hat{x}}_k \\ \hdashline \dot{C}_k \end{bmatrix} = F_{k,aug} \begin{bmatrix} \hat{x}_k \\ \hdashline C_k \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \\ \bar{b}_1 & \bar{b}_2 & \dots & \bar{b}_p \end{bmatrix} u \tag{5.12}$$

Where, $\bar{b}_i = CV_k.col_{Bi}$, and col_{Bi} is the ith column of linearized input matrix. As the state vector is augmented, the covariance matrix also takes modified form where we have one additional row and additional column given by $CV.P$ and $P.CV^T$, respectively (P can be both predicted or updated covariance matrix). Further simplification can be performed on these expressions as showed below. The ith element of $CV.P$ is given by:

$$\begin{aligned}(CV_k.P)_i &= \sum_{j=0}^{n} \alpha_j cov(x_i, x_j) \\ &= \sum_{j=0}^{n} cov(x_i, \alpha_j x_j) \\ &= cov(x_i, \sum_{j=0}^{n} \alpha_j x_j) \\ &= cov(x_i, C_k)\end{aligned} \tag{5.13}$$

Similar simplification can also be performed for $P.CV^T$. With these simplifications, the fully encoded covariance matrix takes the form:

$$P_{aug} = \begin{bmatrix} var(x_1) & cov(x_1, x_2) & \dots & cov(x_1, C_k) \\ cov(x_1, x_2) & var(x_2) & \dots & cov(x_2, C_k) \\ \vdots & \vdots & \vdots & \vdots \\ cov(x_1, C_k) & cov(x_2, C_k) & \dots & var(C_k) \end{bmatrix} \tag{5.14}$$

From these representations, it is clear that the augmented system contains all the computations for the canonical EKF, and hence, can be considered as a filter augmented with a check state. We propose two checks, namely the state check and the variance check (explained below) by reusing these EKF computations. The overhead can vary from $\mathcal{O}(1/k)$ to 100% where k is the number of system states, and this depends on the composition of the Jacobian. In rest of this work, we denote the state estimates $\hat{x}_k^-$ and $\hat{x}_k$ as x_k^{prior} and x_k^{post} and covariance matrices P_k^- and P_k as $P_k{}^{prior}$ and $P_k{}^{post}$, respectively. Additionally, we assume x_k be the vector of states $x_k = [s_{1k},\ s_{2k},\ ..,\ s_{ik},\ ..,\ s_{nk}]^T$, and the variance of the state s_{ik} be σ_{ik} for both prior and posterior quantities. These notations are used to deduce the proposed checks in following subsections.

5.3.2 Algorithmic State Check

In the computation flow of the EKF, the check state at time instants k, C_k^{post} and the state estimates x_k^{post} are computed during update steps. The response of a state variable system is highly correlated in time in the presence of bounded inputs, and hence, for a nominal system, the quantities C_k^{post} and $CV_k.x_k^{post}$ are equal (limited by modelling inaccuracy and noise statistics). As a result, the quantity $e_{x,a}(k) = C_k^{post} - CV_k.x_k^{post}$ is very small. However, this argument is not true when the system experiences failure. Here, the quantity $C_k^{post} - CV_k.x_k^{post}$ exceeds a threshold (discussed later), which can be used in real-time to detect the presence of failures or errors in the system, and we propose this as the 'Algorithmic state check'.

5.3.3 State Update Check

Similar to the 'Algorithmic state check', the states are computed during predict (x_k^{prior}) and update (x_k^{post}) at a time instant k, steps along with corresponding check states (C_k^{prior} and C_k^{post}). The response of a state variable system is highly correlated in time in the presence of bounded inputs, and hence, for a nominal system, the quantities C_k^{prior} and C_k^{post} are highly correlated as well. As a result, the quantity $e_{x,u}(k) = C_k^{prior} - C_k^{post}$ is very small (limited by modeling inaccuracies and noise). However, this argument is not true when the system experiences failure. Here, the quantity $C_k^{prior} - C_k^{post}$ exceeds a time-varying threshold (discussed later), which can be used in real-time to detect the presence of failures or errors in the system, and we propose this as the 'State update check'.

5.3.4 Statistical Variance Check

The 'Statistical Variance Check' is constructed using the covariance matrix which is given in Equation 5.14. From the properties of random variables, the variance of the updated check state at time instants k, C_k^{post} (defined earlier as $C_k^{post} = CV_k x_k^{post}$), can be expressed as:

$$var(C_k^{post}) = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post}) \tag{5.15}$$

As can be observed and proved using properties of random variables, both $var(C_k^{post})$ and $\Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post})$ are equal (limited by noise and modelling inaccuracy) in nominal operation. However, they will differ significantly when the system experiences anomalies as changes in the state trajectories impact state covariance values. Hence, we propose the following as the definition of a statistical variance check $e_{v,a}(k)$:

$$e_{v,a}(k) = var(C_k^{post}) - \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post}) \tag{5.16}$$

5.3.5 Variance Update Check

The ‘statistical variance Check’ is also constructed using the covariance matrix which is given in Equation 5.14. Using the representation of $var(C_k)$, we define a quantity v_k^* for both prior and posterior covariance matrices as follows:

$$\begin{aligned} v_k^{prior} &= var(C_k^{prior}) = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{prior}, s_{jk}^{prior}) \\ v_k^{post} &= var(C_k^{post}) = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post}) \end{aligned} \tag{5.17}$$

As can be observed, v_k^{post} and v_k^{prior} will agree during nominal operation of the system (limited by noise and modelling inaccuracy). However, they will differ significantly when the system experiences anomalies as changes in the state trajectories impact state covariance values. Hence, we propose the following as the definition of a variance check $e_{v,u}(k)$ which is followed by mathematical analysis:

$$e_{v,u}(k) = v_k^{prior} - v_k^{post} \tag{5.18}$$

All four quantities namely - ‘algorithmic state check’, ‘state update check’, ‘statistical variance check’, and ‘variance update check’ can be used to detect failures and they can be proved using properties of random variables.

We propose following theorem with proof to establish this.

Theorem 5.3.1. *Consider a nonlinear system with system states x which receives input u. Further assume that, the system states are following Gaussian distribution. The 'Algorithmic state check' $e_{x,a}(k) = C_k^{post} - CV_k.s_k^{post}$ is zero under nominal condition and 'State update check' $e_{x,u}(k) = C_k^{prior} - C_k^{post}$ is bounded by $e_{x,u}(k)_{nominal}$, where $e_{x,u}(k)_{nominal} = max(\sum_{i=0}^{n} \alpha_{ik} K_k(z_k - h(\hat{s}_k^-)), \forall k$*

Proof. From the earlier discussion, for the case of 'Algorithmic state check', the quantity C_k^{post} is defined as $CV_k.s_k^{post}$ and their relationship is defined in Equation 5.12. From the definition of C_k^{post}, the quantities C_k^{post} and $CV_k.s_k^{post}$ will agree and $e_{x,a}(k)$ will be zero.

For the 'State update check', $e_{x,u}(k)$ is defined as $e_{x,u}(k) = \sum_{i=0}^{n} \alpha_{ik}(s_{ik}^{prior} - s_{ik}^{post})$. Under nominal operation of the system, different quantities of the EKF follows the behavior described in Subsection 5.2. According to these relations, the state update check can be simplified to:

$$\begin{aligned} e_{x,u}(k) = C_k^{prior} - C_k^{post} &= \sum_{i=0}^{n} \alpha_{ik}(\hat{s}_k^- - \hat{s}_k) \\ = \sum_{i=0}^{n} \alpha_{ik} K_k \tilde{y}_k &= \sum_{i=0}^{n} \alpha_{ik} K_k (z_k - h(\hat{s}_k^-)) \end{aligned} \tag{5.19}$$

Lets define, $e_{x,u}(k)_{nominal} = max(\sum_{i=0}^{n} \alpha_{ik} K_k(z_k - h(\hat{s}_k^-)), \forall k$. Under nominal condition of system operation, the operation of EKF is also nominal and the state residuals $\tilde{y}_k = z_k - h(\hat{s}_k^-)$ are bounded. This makes the 'State update check' also bounded and the bound is defined by $e_{x,u}(k)_{nominal}$. □

Corollary 5.3.1.1. *Errors in the Kalman filter computations that produce check error values for the 'algorithmic state check' and 'state update check' larger than their respective thresholds will be detected by the proposed methodology.*

Proof. To set up the premise of the proof, we look at different computation steps (Step 1-4 in Section 5.2) of Extended Kalman Filter and observe how they are changed in presence faults. As seen before, the predicted covariance estimate P_k^- subsequently changes the other quantities one after another. Let us consider the predicted covariance estimate (Step 1) is changed in following way:

$$P_{k,corrupt}^- \rightarrow P_{k,nominal}^- + \Delta P_{k-1} \tag{5.20}$$

where, ΔP_{k-1} represents the faulty value (zero in absence of any error). Because of this change covariance estimate, the residual covariance S_k (Step 2 and 3) will also change in the following:

$$S_{k,corrupt} \rightarrow S_{k,nominal} + \Delta S \tag{5.21}$$

After simplification, the expression of ΔS becomes (Step 3),

$$\Delta S = H_k \Delta P_{k-1} H_k^T \tag{5.22}$$

Similarly, the expression of Kalman gain in presence of fault becomes (Step 3),

$$\begin{aligned} \Delta K_k &= K_{k,corrupt} - K_{k,nominal} \\ &= (P_k^- + \Delta P_{k-1}) H_k^T (S_k + \Delta S)^{-1} - P_k^- H_k^T S_k^{-1} \\ &= P_k^- H_k^T (S_k + \Delta S)^{-1} + \Delta P_{k-1} H_k^T (S_k + \Delta S)^{-1} \\ &- P_k^- H_k^T S_k^{-1} \end{aligned} \tag{5.23}$$

We use Woodbury matrix identity [173] to compute the inverse of corrupted residual covariance. This relation states that if there are two matrices A and B, then inverse of

$(A+B)$ can be computed by the following:

$$(A+B)^{-1} = A^{-1} - \underbrace{\frac{1}{1+g} A^{-1} B A^{-1}}_{\tilde{B}\,(let,)} \tag{5.24}$$

where, $g = trace(BA^{-1})$. Using this relation to compute the inverse of residual covariance and after simplification, we obtain the following:

$$\Delta K_k = \Delta P_{k-1} H_k^T S_k^{-1} + (P_k^- + \Delta P_{k-1}) H_k^T \tilde{\Delta}_S \tag{5.25}$$

where, $\tilde{\Delta}_S = -\frac{1}{1+g_s} S_k^{-1} \Delta S S_k^{-1}$, and $g_s = trace(\Delta S S_k^{-1})$ Similarly corrupted version of updated covariance matrix (Step 4) will change to:

$$\begin{aligned}
& P_{k,corrupted} \\
& = (I - (K_k + \Delta K_k) H_k)(P_k^- + \Delta P_{k-1}) \\
& = \underbrace{(I - K_k H_k) P_k^-}_{P_{k,nominal}} + \\
& \underbrace{(I - K_k H_k)\Delta P_{k-1} - \Delta K_k H_k P_k^- - \Delta K_k H_k \Delta P_{k-1}}_{\tilde{\Delta}_P\,(let,)} \\
& = P_{k,nominal} + \tilde{\Delta}_P
\end{aligned} \tag{5.26}$$

Similarly the change in states becomes (Step 4),

$$\begin{aligned}
\hat{s}_{k,corrupted} &= \hat{s}_k + (K_k + \Delta K_k)\tilde{y}_k \\
&= \underbrace{\hat{s}_k + K_k \tilde{y}_k}_{\hat{s}_{k,nominal}} + \underbrace{\Delta K_k \tilde{y}_k}_{\tilde{\Delta}_s\,(let,)} \\
&= \hat{s}_{k,nominal} + \tilde{\Delta}_s
\end{aligned} \tag{5.27}$$

With these aforementioned relations, the algorithmic state check $e_{x,u}(k)$ can be simplified to:

$$e_{x,a}(k) = CV_k.\tilde{\Delta}_s \tag{5.28}$$

which crosses the bound for algorithmic state check. Similarly, the state update check, $e_{x,u}(k)$ is expressed as:

$$e_{x,u}(k) = \sum_{i=0}^{n}(\alpha_{ik}(\hat{s}_k^- - \hat{s}_k)) = \sum_{i=0}^{n}(\alpha_{ik}K_k\tilde{y}_k) \tag{5.29}$$

Reusing the expression of corrupted K_k and after further simplification,

$$\begin{aligned} e_{x,u}(k) &= e_{x,u}(k)_{nominal} + \sum_{i=0}^{n}(\alpha_{ik}\Delta K_k\tilde{y}_k) \\ &> e_{x,u}(k)_{nominal} \end{aligned} \tag{5.30}$$

Now, the first term of the aforementioned equation represents bound due to nominal operation (due to modelling inaccuracy, noise etc.). And, in presence of failure, the second term of the last equation would become non-zero. From this analysis, it is shown how errors in computation of different steps in EKF are detected in algorithmic state check and state update check. □

From corollary 5.3.1.1, it can be seen that the error in covariance matrices (step 1) will be detected as this error will create non-zero elements in ΔP_{k-1}. Similarly, sensor, actuator and control program failure will be detected by ‘state update check’ due to incorrect residue (step 2), out of bounds systems states (Step 4), and out-of-distribution control action respectively (Step 3). On the other hand, $e_{x,a}(k)$ will detect failure in sensor and actuator through system residue (Step 2) and state estimates (Step 4).

Theorem 5.3.2. *The ‘statistical variance check’* $e_{v,a}(k) = var(C_k^{post}) - \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post})$ *is zero under nominal operation, and ‘variance update check’* $e_v(k) = v_k^{prior} - v_k^{post}$, *is bounded by* $e_{v,u}(k)_{nominal}$*, where* $e_{v,u}(k)_{nominal} = max(\sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}((P_k^-)_{ij} - (P_{k,nominal})_{ij}), \forall k$ *and* P_{ij} *represents element of* P *at* i*th row and* j*th column.*

Proof. From the properties of random variables, the variance of C_k is given by:

$$var(C_k^{post}) = var(\sum_{i=0}^{n}\alpha_{ik}s_k^{post}) = \sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post}) \tag{5.31}$$

As can be observed from the definition of 'statistical variance check', $var(C_k^{post})$ and $\sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post})$ are equal which makes $e_{v,a}(k)$ zero under nominal condition.

The 'statistical variance check' $e_{v,u}(k)$ is dependent on v_k^{prior} and v_k^{post} which are defined at Equation 5.17. Under nominal operation, different computations of the EKF follows the behavior described in Section 5.2, and the expression of 'statistical variance check' $e_{v,u}(k)$ becomes,

$$e_{v,u}(k) = \sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}((P_k^-)_{ij} - (P_k)_{ij}) \tag{5.32}$$

Lets define, $e_{v,u}(k)_{nominal} = max(\sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}((P_k^-)_{ij} - (P_{k,nominal})_{ij}), \forall k$. Under nominal condition of system operation, the operation of EKF is also nominal and the state residuals covariance matrices are very close to one another. Hence their element wise difference is bounded. This makes the 'variance update check' also bounded and the bound is defined by $e_{v,u}(k)_{nominal}$. □

Corollary 5.3.2.1. *Errors in the Kalman filter computations that produce check error values for the 'statistical variance check' and 'variance update check' larger than their respective thresholds will be detected by the proposed methodology.*

Proof. In Theory 5.3.1.1, we showed how the deviation in predicted covariance matrix introduces changes other quantities in the EKF computaion, which will be used in this proof. The expression of $var(C_k)$ in presence of failure becomes:

$$var(C_k) = var(\sum_{i=0}^{n} \alpha_{ik}(s_{ik} + (\tilde{\Delta}_s)_i)) \tag{5.33}$$

From the properties of random variables, the variance of sum of random variables A and B is $var(A+B) = cov(A,A) + cov(B,B) + 2cov(A,B)$. Using this relation, $var(\sum_{i=0}^{n} \alpha_{ik}(x_{ik} + (\tilde{\Delta}_x)_i))$ can be expressed as sum of $\sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{ik}\alpha_{jk}cov(s_{ik}, s_{jk})$ and variance of corrupted terms $(\tilde{\Delta}_s)_i$ and covaiance of $(\tilde{\Delta}_s)_i$ and s_{ik}. From this analysis, it can be shown that the first term corresponds to bound and the second and third terms will cross the bounds in presence of failure.

In presence of failure, the variance update check, $e_{v,u}(k)$ becomes:

$$\begin{aligned} e_{v,u}(k) &= (\sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{ik}\alpha_{jk}cov(s_{ik}^{prior}, S_{jk}^{prior})) \\ &- (\sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post})) \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{ik}\alpha_{jk}(cov(s_{ik}^{prior}, s_{jk}^{prior}) - cov(s_{ik}^{post}, s_{jk}^{post})) \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{ik}\alpha_{jk}((P_k^-)_{ij} - (P_k)_{ij}) \end{aligned} \tag{5.34}$$

In presence of failure, the expression of P_k is corrupted and, as shown before, takes the following form:

$$P_{k,corrupted} = P_{k,nominal} + \tilde{\Delta}P \tag{5.35}$$

Using this relation and after further simplification, the expression of $e_{v,u}(k)$ becomes,

$$\begin{aligned} e_{v,a}(k) &= \sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}((P_k^-)_{ij} + (\Delta P_{k-1})_{ij} \\ &- (P_{k,nominal})_{ij} - (\tilde{\Delta P})_{ij}) \\ &= e_v(k)_{nominal} + \sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}((\Delta P_{k-1})_{ij} - (\tilde{\Delta P})_{ij}) \\ &> e_v(k)_{nominal} \end{aligned} \tag{5.36}$$

The first term of the aforementioned equation represents the bound which will be present in nominal form (due to modelling inaccuracy, noise etc.). And, in presence of failure, the second term of the last equation would become non-zero. Hence, the error will be detected. □

As shown in the proof of the corollary 5.3.1.1, state update check $e_{x,u}(k)$ can detect failure. In reality, it is compared against a time-varying threshold to determine whether the system is experiencing failure. The threshold needs to be selected carefully to ensure good 'precision' and 'recall'. For this reason, a statistical bound is computed to determine the threshold and a two-tail statistical test is performed on these bounds to determine whether the system is experiencing anomalies. To make this bound even tighter and to make the checks more responsive to different anomalies, the time varying coding vector CV_k is determined. We explain the time-varying threshold and coding vector using state checks in the following. However, the analysis is extended to variance checks as well.

Determination of Time-Varying Coding Vector

To allow the checks to adapt to changing performance sensitivities to error/failure conditions, the coding vector CV_k is computed as suggested by Lemma 1. Here, $CV_k = [\alpha_{1k}, \alpha_{2k}, \ldots, \alpha_{nk}]$ and $\{\alpha_{ik}\}$ is varied inversely to the variance of the state x_i at time instant k (the latter variance is obtained as a direct by-product of the EKF computations).

This reduces the variance of the check quantity $e_x(k)$ allowing a tighter detection threshold at time instant k. Example state check plots for a quadcopter system with constant (as in prior research) and variable coding vectors are shown in Figure 5.2, where a small error is introduced in the sensor reading and escapes the constant coding vector based check (left), but is detected by the time-varying coding vector based check (right). Let x_k be the vector of states $x_k = [s_{1k}, s_{2k}, .., s_{ik}, .., s_{nk}]^T$, and the variance of the state s_{ik} be σ_{ik}, then we have,

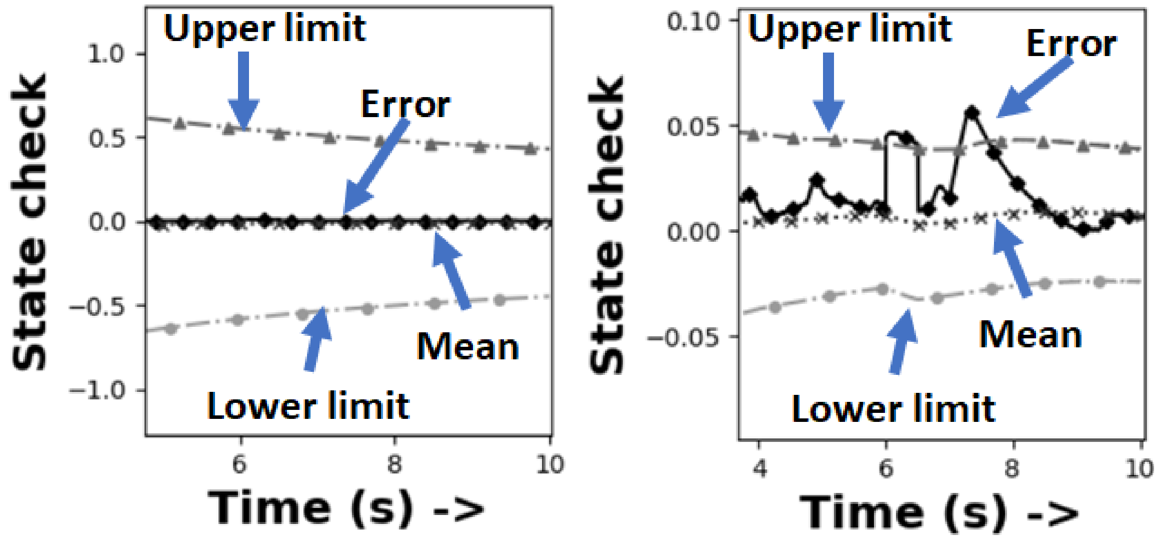


Figure 5.2: State check plot for (left) constant, and (right) variable CV for a transient sensor failure in quadcopter

Lemma 1. *Given N independent normally distributed states and elements of coding vectors α_{ik} such that the variance of each $s_{ik} > 1$, the variance of the resultant check state reduces when the coding vectors are produced by $\alpha_{ik} = 1/\sigma_{ik}$ at time step k.*

Proof. From the properties of a normal distribution, for independent and identically distributed states, the variance of $\sum_{i=1}^{N} \alpha_{ik}s_{ik}$ is the sum of the individual variances of each $\alpha_{ik}s_{ik}$, i.e. $var(C_k) = var(\sum_{i=1}^{N} \alpha_i s_{ik}) = \sum_{i=1}^{N} \alpha_{ik}^2\sigma_{ik}^2$ for each $\alpha_{ik}\sigma_{ik}$ being the standard deviation of a given $\alpha_{ik}s_{ik}$. Since each $\alpha_{ik}^2\sigma_{ik}^2 > 1$ we have $var(C_k) > N$.

When we scale the states such that $\alpha_{ik} = 1/\sigma_{ik}$ we map each s_{ik} to the unit normal distribution with unity standard deviation, giving $var(\alpha_{ik}s_{ik}) \approx 1$, and the overall check state variance as $var(C_k) = \sum_{i=1}^{N} 1^2 \approx N$. From the previous, we see that the variance is reduced. □

Accordingly, the coding vector at time step k is selected to satisfy the constraint $\alpha_{1k}\sigma_{1k} = \alpha_{2k}\sigma_{2k} = \cdots = \alpha_{ik}\sigma_{ik}$, $1 \leq i \leq n$ with $\sum_{i=1}^{n} \alpha_{ik} \leq R$, where R is selected to prevent overflow in the numerical arithmetic involved.

Computation of Time-Varying Check Threshold

Since C_k is a weighted sum of the states, a hypothesis test is performed to determine how the state check should be thresholded as a proxy for flagging errors in the system states. For a hypothesis test, we choose the simple two-tail test for normal distributions [174]. To determine the statistical bound (threshold) of the check, the trajectory of C_k^{prior} and C_k^{post} are recorded and mean and variance of the error $e_x(k)$ are determined for a certain time window. In general, the time window can be selected to be *adaptive* (smaller time-window when the system is in equilibrium and higher when performing complex operations). In this work, the length of the window was determined by the method described in [83–85]. The covariance is used to derive the variance of the error between the prior and posterior check state. Thus for the check state C_k, one has $M(C_k^{prior}, C_k^{post}) = [\sigma^2(C_k^{prior}), \sigma_{cov}^2; \sigma_{cov}^2, \sigma^2(C_k^{post})]$ as the covariance matrix. From this representation, the following relationship is used to determine the variance: $\sigma^2(C_k^{prior} - C_k^{post}) = \sigma^2(C_k^{prior}) + \sigma^2(C_k^{post}) - 2cov(C_k^{prior}, C_k^{post})$.

After the mean and variance are computed, the statistical bound of the error is determined by:

$$\mu \pm \rho\sigma \tag{5.37}$$

where, μ and σ are mean and standard deviation of the error $e_x(k)$ respectively for the window described earlier, and ρ controls the error bound beyond which system anomalies

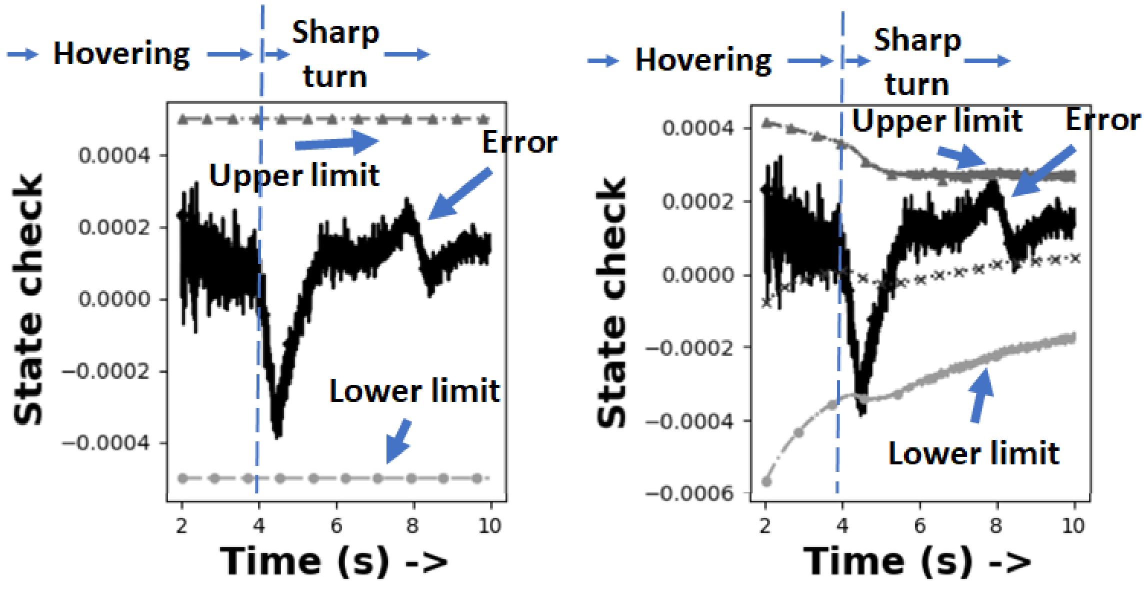


Figure 5.3: State check plot for (left) constant, and (right) variable threshold

are flagged. Note that, the quantities μ and σ are time varying in nature, whereas the quantity ρ is fixed and is determined by the method explained in 5.6.1. This results in a *time-varying threshold for the check employed.* In addition to the check proposed in this work, a simple state check can also be adapted from [175] with time-varying coding vectors and anomaly detection thresholds, to address detection of slow-changing parametric failures.

An example use of a variable check threshold is shown in Figures 5.3a and 5.3b where a quadcopter hovers starting at $t = 0$ sec and goes through a sharp turn $t = 3.95$ sec. The fixed threshold scheme was not able to detect the fault as shown in Figure 5.3a. However, the variable threshold (Figure 5.3b) adapts dynamically to the operating environment of the quadcopter, maximizing anomaly coverage while minimizing false alarms.

5.4 Resilience methodology

The diagnosis and resilience methodologies for failures at different subsystems of the autonomous system are discussed in this subsection. As shown in the Figure 5.4, after an error is detected, the source of this error is diagnosed by the method explained in subsec-

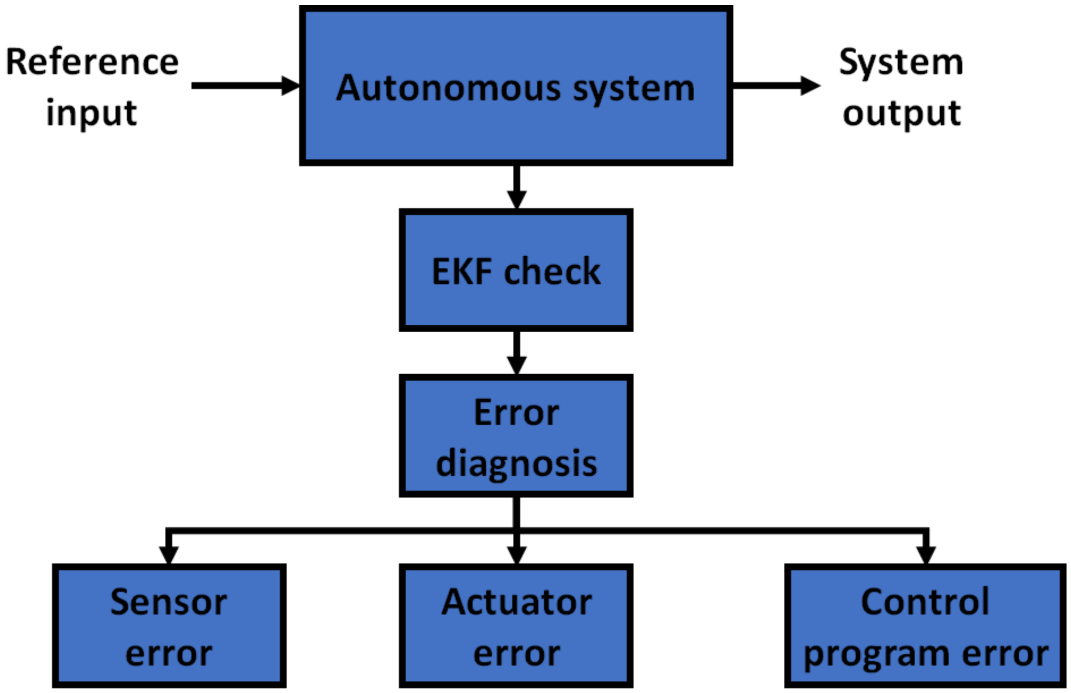


Figure 5.4: Resilience methodology flow

tion 5.4.1. Parametric fault was considered as the failure model for sensor and actuator, and transient fault was considered for control program error. The detailed diagnosis and resilience methodology for each of these failure are explained below.

5.4.1 Machine learning assisted failure diagnosis

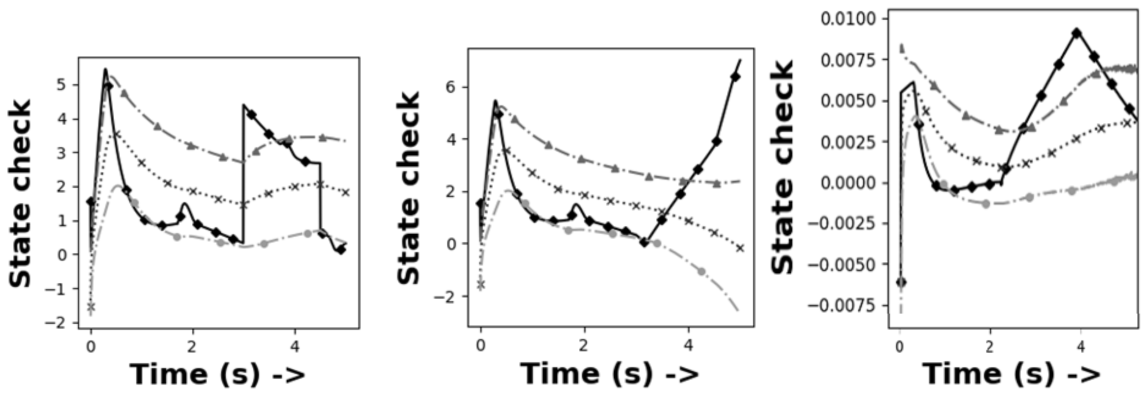


Figure 5.5: State check plot for (left) sensor, (middle) actuator, and (right) control program failure

Once the failure is detected, the next step is to diagnose the source of the failure. Sample plots for failures in different parts of the system, namely - sensor, actuator, and control program are shown in Figure 5.5. As can be observed, the time domain property of error signals are different for different failure mode and this can be used for diagnosis purpose. For this reason, a machine learning based approach is employed to classify the source of failure as shown in Figure 5.6. Here, it is assumed that the system behavior will be captured sufficiently in widely diverse mode of operations and failure to effectively employ the classifier as a diagnosis tool. In this work, a 'Support Vector Machine' [124] has been trained where the model looks at the state and variance checks of the system for different failure modes, and learns their dependency on the error. The model was trained using *Radial Basis Function* as the kernel. The model was trained on data-sets obtained from 300 simulation studies performed in diverse operating conditions and failure modes selected in random to completely capture the behavior, and the hyper-parameters of the model were optimized.

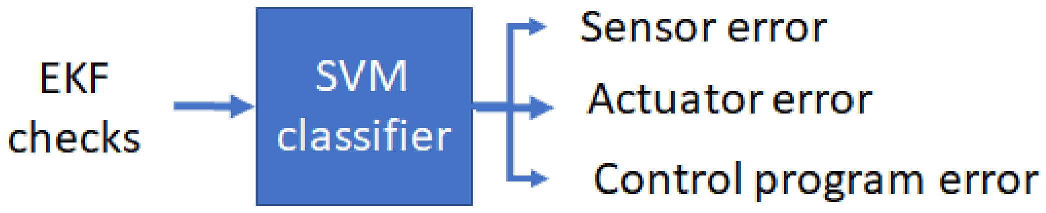


Figure 5.6: Error diagnosis block diagram

Figure 5.7 shows the performance of the trained model for different hyper-parameters. 'Classification score' has been selected as the performance metric which is defined as the fraction of data which are classified accurately (the higher the better). It is observed from this Figure that, the model hyper-paramters γ and C (γ and C are *kernel coefficient* and *regularization paramter* respectively) are varied for different regions and the classification score also varies. In the given experiments, the score varied from 67% to 100% for different combination of hyper-parameters. From these experiments, it is found that the best performance of the model is obtained when $\gamma = 0.1$ and $C = 1$ when the classification

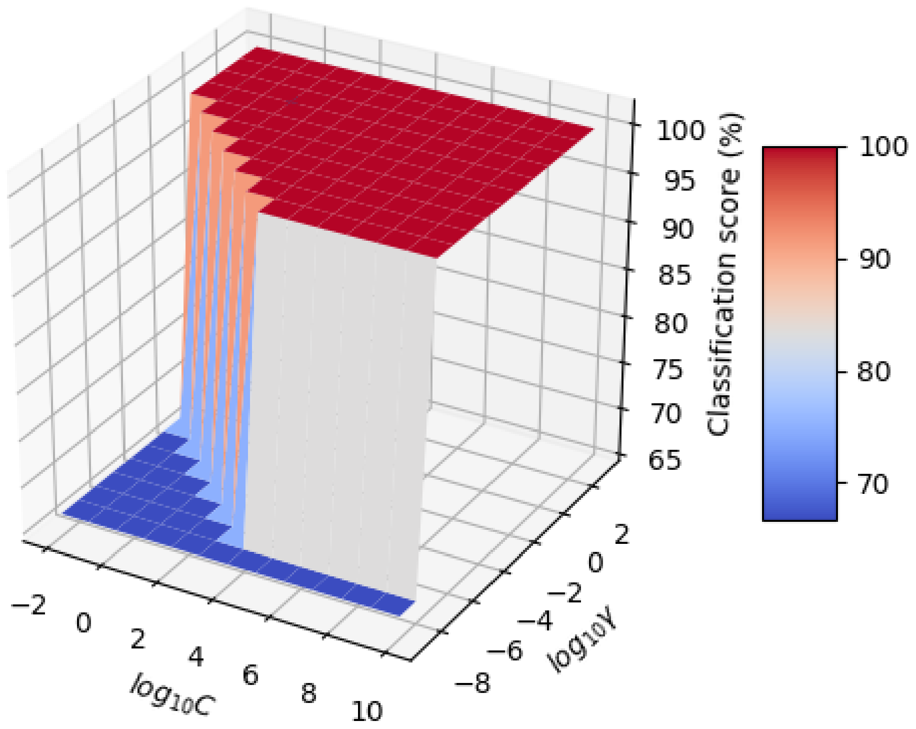


Figure 5.7: Classification score the diagnosis engine for different hyper-parameter score of 100% was obtained. Higher value of γ and C will have similar score. However, it is not recommended as this may result into overfitting.

Sensor Fault

Sensor offset and scaling faults have been considered which are the most common form of failure the sensor. For correction of this kind of failure, we propose the parameter optimization on-the-fly and apply appropriate correction in the sensor reading. Here, the fault can be detected by the checking scheme and diagnosed by the machine learning based method which are explained earlier. After the fault is diagnosed, the observed state checks are sampled for t_k time instants and the recorded quantity is defined as $chk_{observed}(t_k)$. A 'Levenberg Marquardt' [125, 126] algorithm based optimization was performed to estimate the changed parameter where the optimization problem is defined as:

$$
\begin{aligned}
p^* &= \underset{p}{\arg\min}\,(\|chk_{observed}(t_i) - chk(p,t_i)\|_2) \\
&\text{subject to, } 0 < t_i \leq t_k
\end{aligned}
\tag{5.38}
$$

where, $chk(p, t_i)$ is the check defined by sensor parameter p at time-instant t_i.

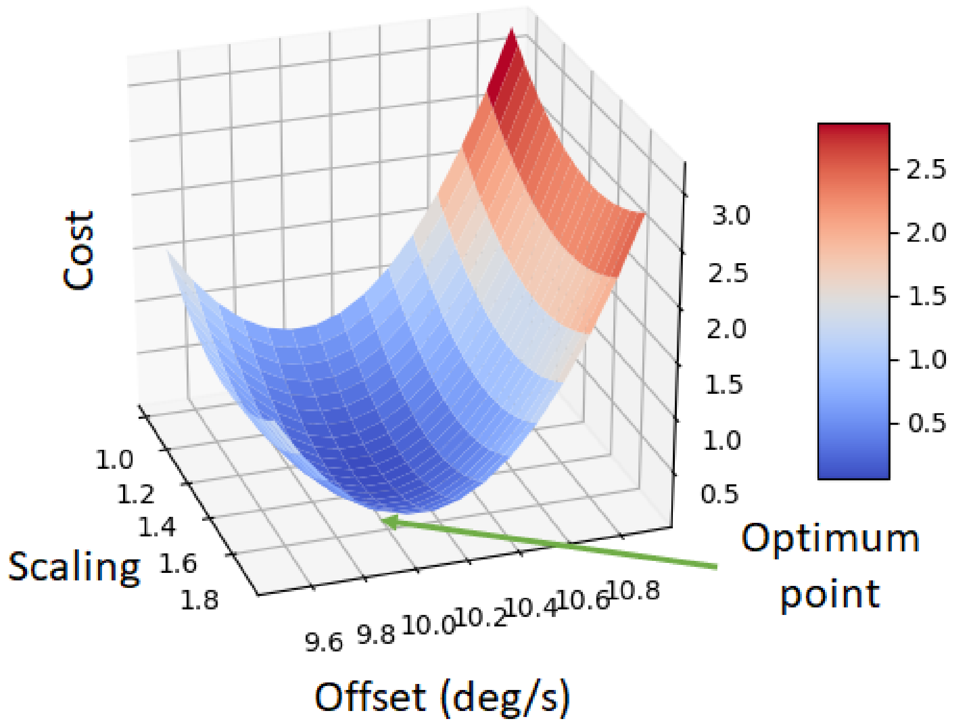


Figure 5.8: Cost function for gyroscope parametric failure

Examples of the cost function for the aforementioned optimization problem is shown in Figures 5.8 and 5.9 where parametric failures were introduced in gyroscope and accelerometer, respectively. As can be observed from these two plots, a properly tailored optimization algorithm can be used to estimate the parameter with very high degree of accuracy. For the best performance of this optimization problem, a trade-off study between relative percentage error of estimation vs. the observation window was performed. This study was performed for three different cases, namely - a) only state check developed using state estimator, b) sensor readings, c) both check and sensor readings were used. The

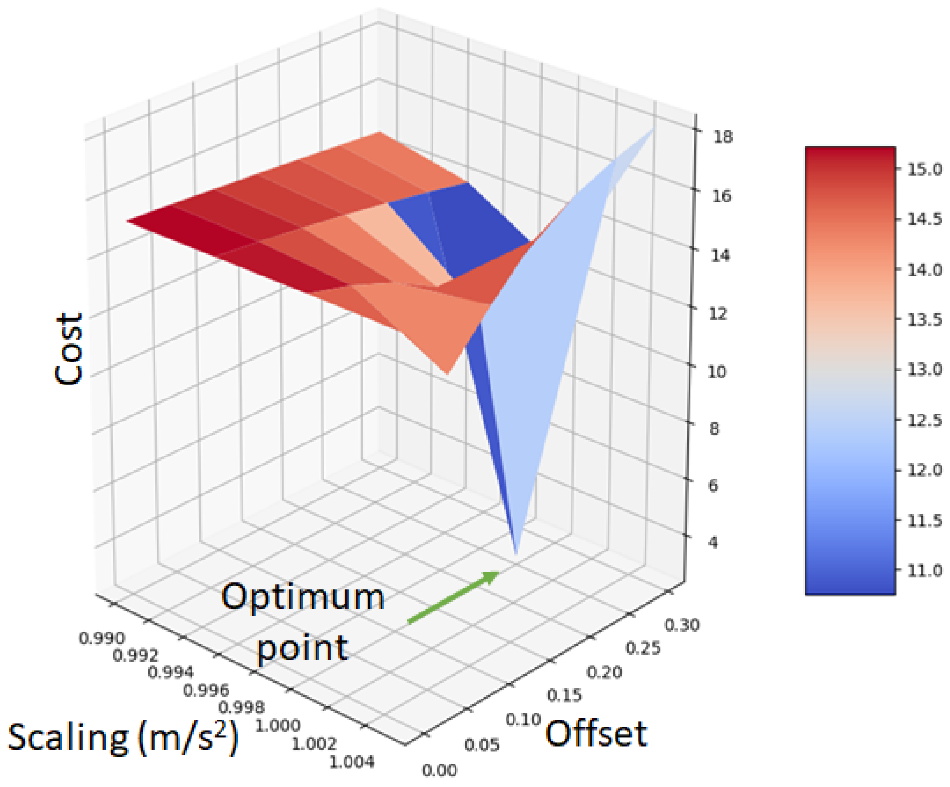


Figure 5.9: Cost function for accelerometer parametric failure

performance of these three studies are shown in Figure 5.10.

As can be observed from the Figure 5.10, the relative percentage error starts to reduce (i.e. the accuracy increases) as the observation window length t_k gradually increases. The reason for this observation is providing more observation points facilitates the optimization problem and hence better accuracy is obtained. However, the relative error goes through an optimum point and the error starts to increase after that length as the observation window is increased further. The reason for this kind of behavior is as the observation window increases too much, no new information is captured. However, increasing the window too much captures the random noises which hurts the accuracy. Hence, a saturation at relative error is observed as the observed window is extended too much.

Among the three different cases, direct comparison with sensor readings resulted into the minimum relative error (i.e. best accuracy), as in this method the sensor readings are directly used in this optimization. Comparison with the observed check results into an

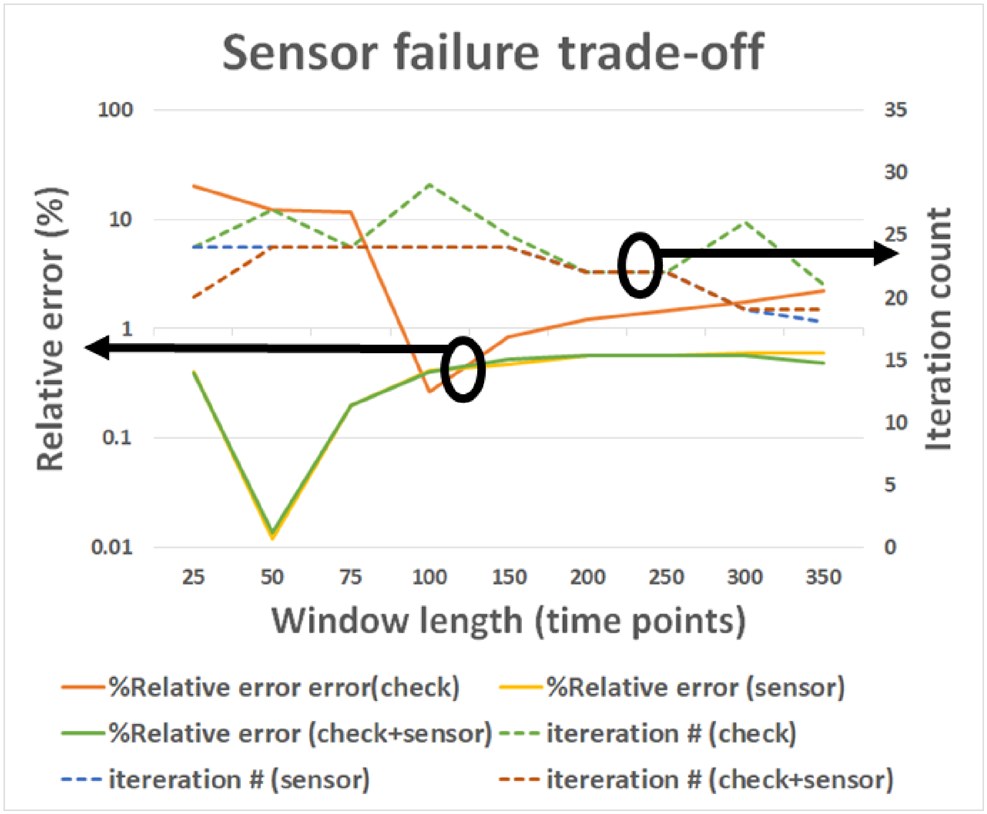


Figure 5.10: Sensor trade-off study

optimum point which is a bit larger than the same for ‘sensor comparison’ case and the accuracy is inferior as well. The reason for this behavior is the checks were constructed by encoding the states’ estimates which may introduce aliasing effects and hence inferior performance is obtained. However, having an ‘Extended Kalman Filter’ is enough for this case, but a system model (which may not be accurate all the time) is necessary for ‘sensor comparison’ method. From this discussion, we can clearly understand the dependency of accuracy on considered data (check value or sensor value or both) is used is extremely important. Additionally, if a system has n states and m sensors, and n states are encoded to produce one check, comparing sensors for optimization would incur $\frac{n}{m}$ times more overhead (in addition to model execution overhead) than comparing the checks. It may be beneficial to use check based optimization when $\frac{n}{m}$ much larger than one or when the relative error is less sensitive to the system behavior. Similarly, for a fully observable system, $\frac{n}{m} = 1$, and the overheads are comparable. From this discussion, it is observed that the appropriate approach will depend on the system. The plots also show the iteration count for three different studies and it was found that all of these approaches take almost same number of iteration. From the observed study observed from this plot, a window length of 50 (equivalent to 0.050 sec) was selected which results in an error of less than 1%. After the correction approach is applied, the check error goes down to nominal case, which is shown in Figure 5.11.

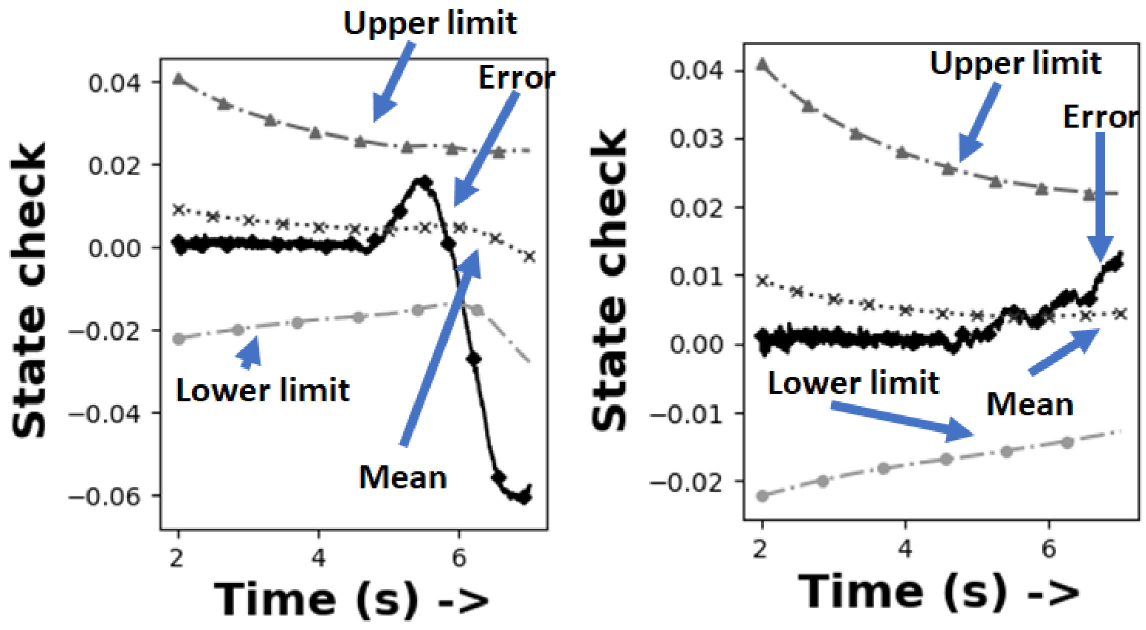


Figure 5.11: Check error plot for (left) without, and (right) with correction for sensor failure

Actuator Fault

Similar to the sensor, parametric deviation was considered as the failure model for actuator in this work.

The actuator parameter estimation accuracy is not very high because an accurate parameter estimation requires consistent persistence in actuator with properly designed input stimuli [88]. For this reason, we employ '*Multi-Armed Bandit*' based reinforcement learning algorithm [176] to find out the best control action. The algorithm excites the actuator

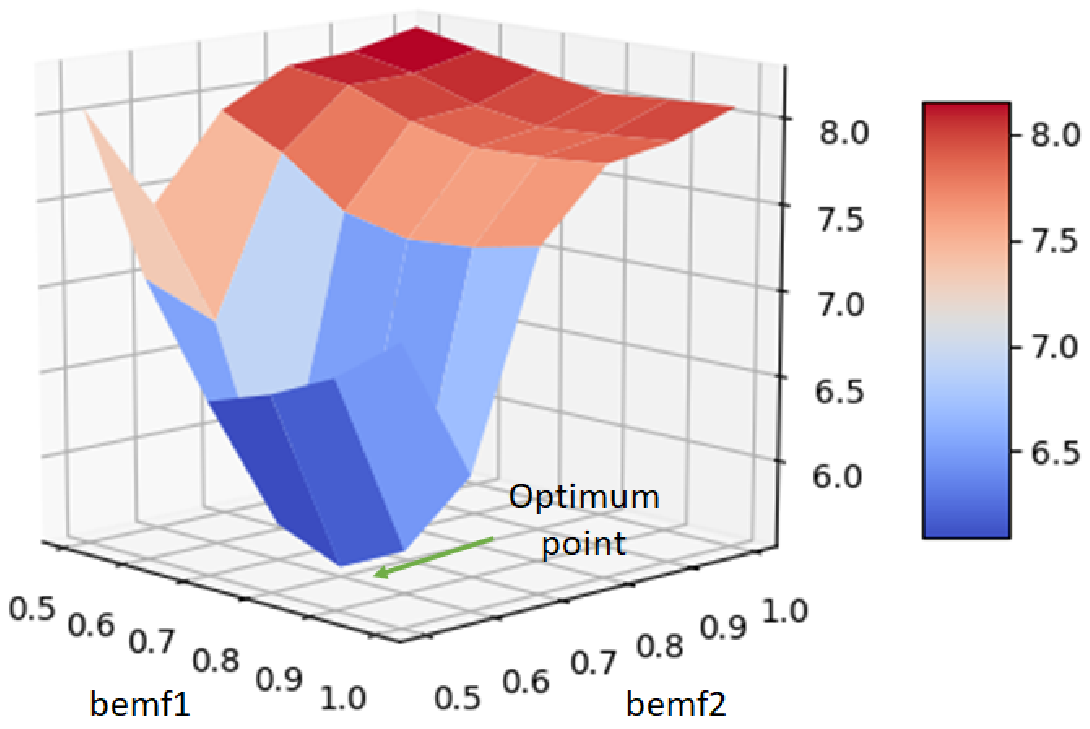


Figure 5.12: Risk of actuator parametric deviation

for different control parameter to minimize the following risk:

$$k^* = \arg\min_k \left(\|chk_{golden}(t_i) - chk(ctrl(k), t_i)\|_2\right) \quad \text{subject to, } 0 < t_i \le t_k \tag{5.39}$$

Where, k represents control parameter, $chk_{golden}(t_i)$ represents reference check readings which an ideal system should exhibit, and $chk(ctrl(k), t_i)$ represents the check readings when the actuator is controlled by a controller having k control parameters. As shown in the Figure 5.12, an appropriately designed cost function of the risk takes a convex shape, and the MAB algorithm will learn the optimum value.

The reward (defined as inverse of risk) of the reinforcement learning problem is defined as follows:

$$R = \frac{1}{(\|chkr_{golden}(t_i) - chk(ctrl(k), t_i)\|_2) + 0.01} \tag{5.40}$$

An example reward map is shown in the following figure for different arm knobs (which is control parameter in this case). As can be seen from the figure 5.13, the controller parameters are correctly tuned when a set of actuators' parameters were changed. The reward value is showing a high value for certain combination of controller parameters which represents the best pair of control actions in this situation. A Gaussian Process based regressor is trained on the fly to compensate for the error signal and it is deployed in parallel to the EKF check. After the correction approach is applied, the check error goes down to nominal case, which is shown in Figure 5.14.

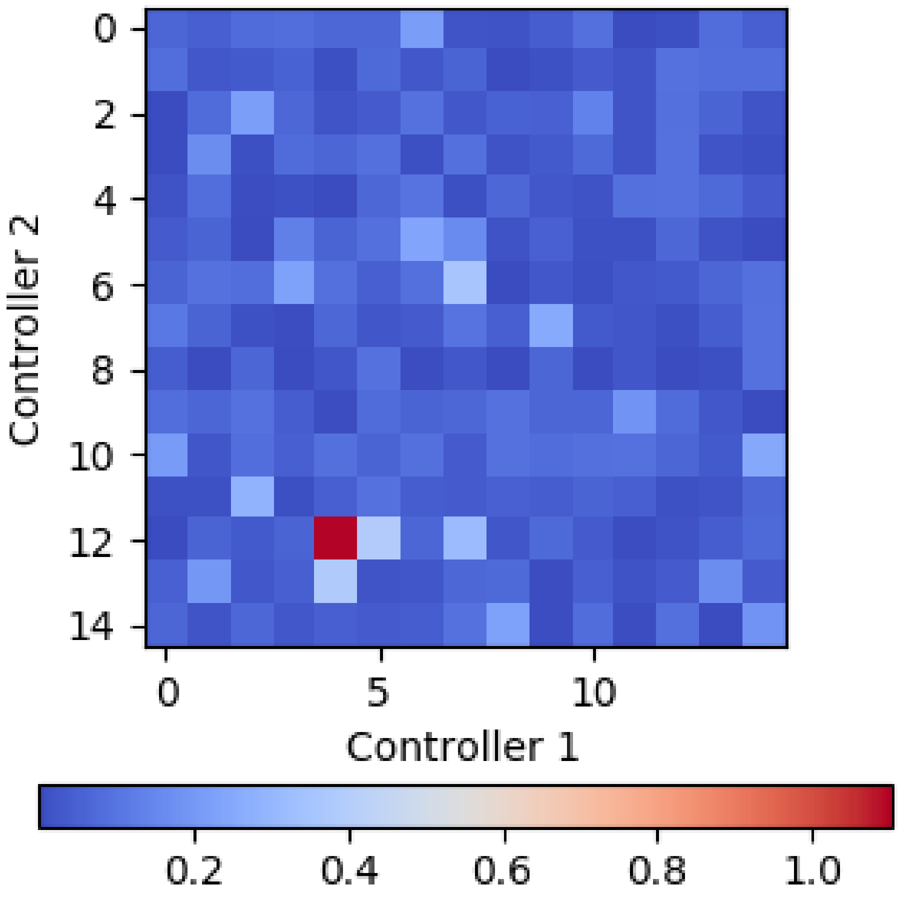


Figure 5.13: Reward (=1/Risk) value of different control parameter for Multi-Armed Bandit based control parameter reconfiguration

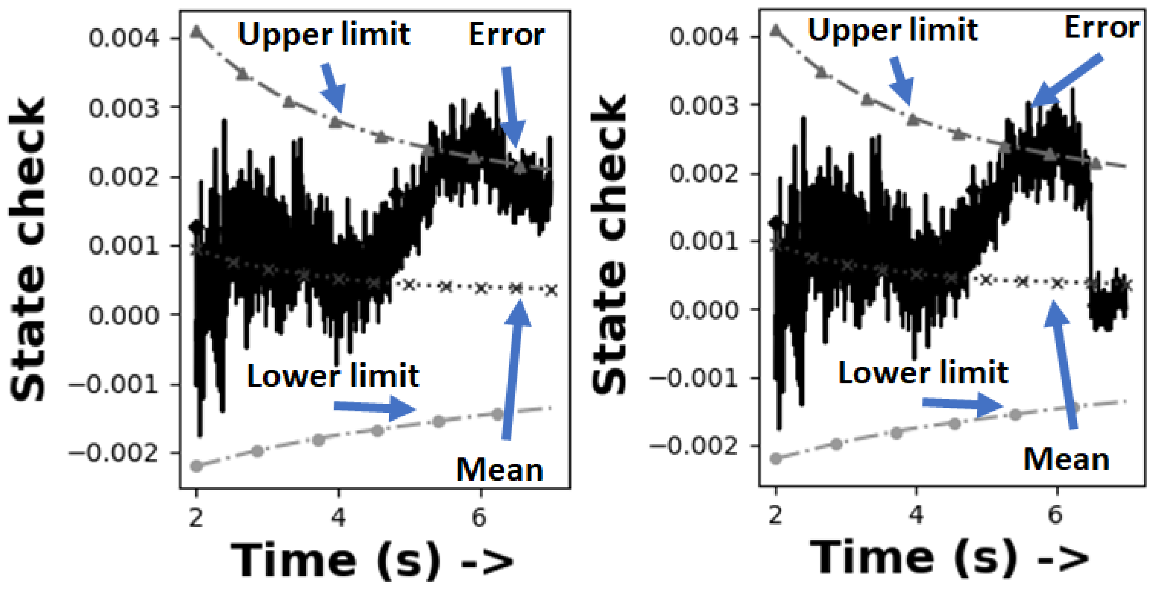


Figure 5.14: Check error plot for (left) without, and (right) with correction for actuator failure

Control Program Fault

For the correction of the transient error in Control program, we propose control action restoration. As the error in control program execution is detected and diagnosed, the last saved control action is applied to restore the safe behavior of the system in fail-safe manner. It is assumed that, the fail-safe action is being executed from a safe part of the controller which is immune from internal or external failures. By this way, the control program fault is corrected.

5.5 Test Cases: Overview and EKF Checks

5.5.1 Quadcopter

A quadcopter is modeled as a nonlinear state variable system with 12 state variables: $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ (see Figure 5.15), where, x, y and z represent the position of the quadcopter in 3 dimensional inertial reference frame and ϕ, θ and ψ represent the roll, pitch and yaw angle in the body frame. The dynamics of the quadcopter is explained in Section 4.3 and hence it is skipped here.

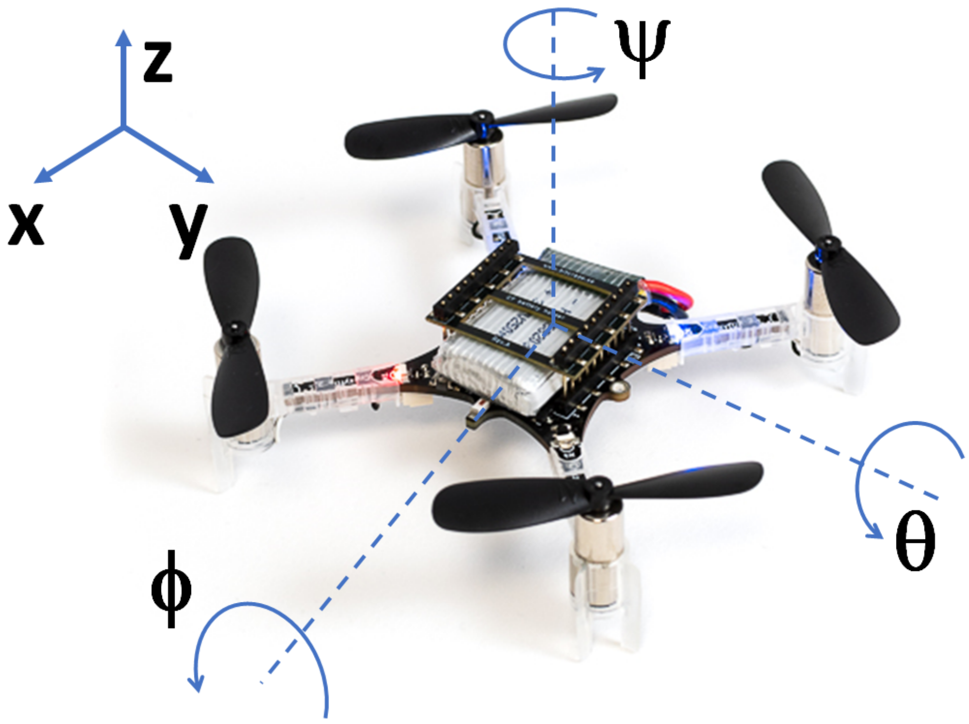


Figure 5.15: An example quadcopter system (adopted from Crazyflie [142])

To implement the EKF on quadcopter system, the Jacobian F_k is computed. The full size of the Jacobian F_k is 12×12 with the following non-zero components:

$$\begin{aligned}
&F_{k(0:2,3:5)} = F_{k(6:8,9:11)} = \mathcal{I}(3), F_{k(3,6)} = T(-c_\psi s_\theta s_\phi + s_\psi c_\phi)/m,\\
&F_{k(3,7)} = T(c_\psi c_\theta c_\phi)/m,\ F_{k(3,8)} = T(-s_\psi s_\theta c_\phi + c_\psi s_\phi)/m,\\
&F_{k(4,6)} = T(-s_\psi s_\theta s_\phi - c_\psi c_\phi)/m,\ F_{k(4,7)} = T(s_\psi c_\theta c_\phi)/m,\\
&F_{k(4,8)} = T(c_\psi s_\theta c_\phi + s_\psi s_\phi)/m,\ F_{k(5,6)} = T(-c_\theta s_\phi)/m,\\
&F_{k(5,7)} = T(-s_\theta c_\phi)/m,\ F_{k(9,10)} = -(I_z - I_y)\dot{\psi}/I_x,\\
&F_{k(9,11)} = -(I_z - I_y)\dot{\theta}/I_x,\ F_{k(10,9)} = -(I_x - I_z)\dot{\psi}/I_y,\\
&F_{k(10,11)} = -(I_x - I_z)\dot{\phi}/I_y,\ F_{k(11,9)} = -(I_y - I_x)\dot{\theta}/I_z,\\
&F_{k(11,10)} = -(I_y - I_x)\dot{\phi}/I_z,\ F_{k(3:5,3:5)} = -k_D\mathcal{I}(3)/m
\end{aligned} \tag{5.41}$$

where, $\mathcal{I}(.)$ = identity matrix, T = Thrust = $T_B[2]$, $[I_x, I_y, I_z] = diag(I)$ (T_B and I are defined earlier), and c_* and s_* represents $\cos(*)$ and $\sin(*)$, respectively. For an appropriately selected coding vector $CV_k \in \mathcal{R}^{12}$, the expression for the state update check becomes $e_{x,u}(k) = \Sigma_{i=1}^{12}(C_k - CV_k x_k^{post})$, algorithmic state check becomes $e_{x,a}(k) = C_k - CV_k.x_k^{post}$, variance update check becomes $e_{v,u}(k) = var(C_k^{prior}) - var(C_k^{post})$, and statistical variance check becomes $e_{v,a}(k) = var(C_k) - \sum_{i=0}^{n}\sum_{j=0}^{n}\alpha_{ik}\alpha_{jk}cov(s_{ik}^{post}, s_{jk}^{post})$. The coding vector CV_k is updated as suggested by Lemma 1, and the statistical bounds for these two checks were determined.

5.5.2 Steer by Wire System

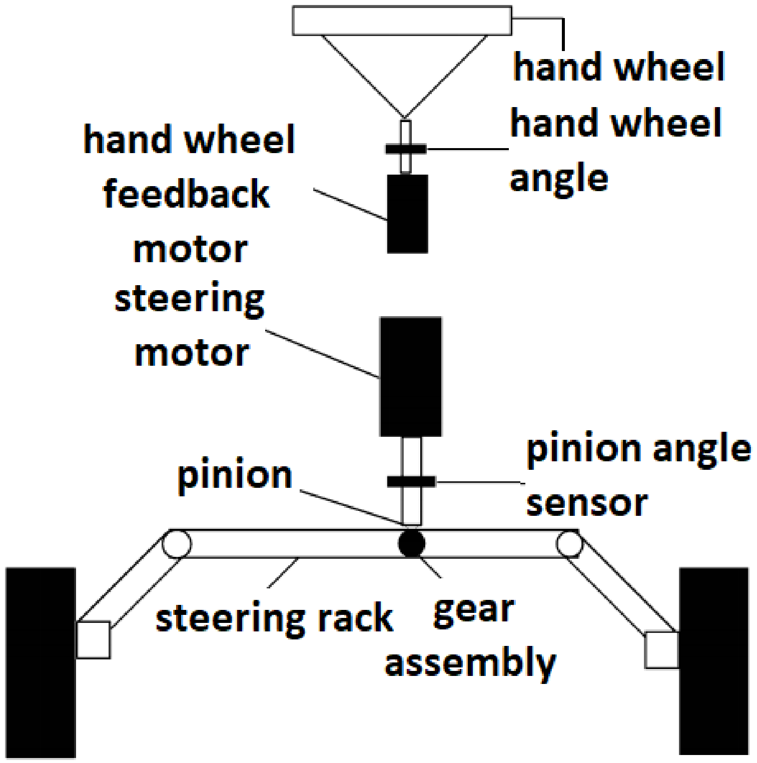


Figure 5.16: Steer-by-Wire system with rotary motors (adopted from [177])

An automotive subsystem, Steer-by-Wire (SbW) is considered as the second test case. This system is an integral part of automotive system and has been focus of research interest [178–183]. As shown in Figure 5.16, a SbW system can be decomposed into two parts, namely hand wheel and front wheel subsystems. The hand wheel subsystem is comprised of the hand wheel, the hand wheel angle sensor, and the feedback motor. The front wheel subsystem is comprised of the steering motor, the pinion angle sensor, the rack and pinion gearbox, and the steered front wheels. These two subsystems are connected through the 'Electronic Control Unit' (ECU). The hand wheel motor generates the feedback torque as a representation of 'road feel', and the front wheel steering motor generates the actual torque to steer the front wheels. Figure 5.16 shows the block diagram of a SbW system.

The dynamics of the hand wheel and front wheel subsystems are given in the following [177]:

$$\begin{aligned} J_h\ddot{\theta}_h + B_h\dot{\theta}_h + c_h\theta_h + \tau_r &= \tau_h \\ J_{eq}\ddot{\delta}_f + B_{eq}\dot{\delta}_f + F_s sign(\dot{\delta}_f) + \tau_e &= \tau_{eq} \end{aligned} \tag{5.42}$$

The vehicle's slip angle β and yaw rate γ dynamics are:

$$\begin{bmatrix} \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} -\frac{C_f+C_r}{MV_{CG}} & -1+\frac{C_r l_r - C_f l_f}{MV_{CG}^2} \\ \frac{C_r l_r - C_f l_f}{I_Z} & -\frac{C_r l_r^2 + C_f l_f^2}{I_Z V_{CG}} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} + \begin{bmatrix} \frac{C_f}{MV_{CG}} \\ \frac{C_f l_f}{I_Z} \end{bmatrix} \delta_f \tag{5.43}$$

where, $J_h(J_{eq})$ = front (equivalent) wheel moment of inertia, $B_h(B_{eq})$ = front (equivalent) wheel viscous friction, c_h = torsional stiffness of the hand wheel shaft, θ_h = hand wheel rotational angle, $\tau_h(\tau_r)$ = hand wheel input (feedback) torque, N_θ = ratio between the hand wheel rotational angle and the front-wheel steering angle, F_s = Coulomb friction constant, $C_f(C_r)$ = front (rear) wheel cornering stiffness, $l_c(l_p)$ = mechanical (pneumatic) trail, $l_f(l_r)$ = front (rear) wheel center to center of gravity (CG) distance, M = vehicle mass, I_Z = vehicle inertia around CG, V_{CG} = vehicle velocity. The expression of J_{eq} and B_{eq} are given below:

$$\begin{aligned} J_{eq} &= J_{fw} + \left(\frac{rN_2}{N_1}\right)^2 J_{sm} \\ B_{eq} &= B_{fw} + \left(\frac{rN_2}{N_1}\right)^2 B_{sm} \end{aligned} \tag{5.44}$$

where, $J_{fw}(J_{sm})$ = moment of inertia of the front wheel (steering motor), $B_{fw}(B_{sm})$ = viscous friction of front wheels (steering motor), r = scale factor to account for the conversion from linear motion of the rack to the rotation at the steering arm and vice versa, N_1 and N_2 are the tooth numbers of the rack and pinion gearbox.

The hand wheel subsystem is controlled by a proportional and derivative (PD) controller, which produces the response θ_h. This is fed to ECU, which generates the reference angle $\theta_{hr} = \theta_h / N_\theta$ for the front-wheel subsystem. The input to this subsystem is τ_{eq} which is computed from reference front-wheel angle θ_{hr} by a sliding mode controller (SMC). To implement the EKF on the frontwheel subsystem of the SbW, the whole system was constructed with systems states $x_k = [\delta_f, \dot{\delta}_f, \beta, \gamma]^T$ at time step k. the Jacobian F_k was computed which has following non-zero elements:

$$
\begin{aligned}
&F_{k(0,1)} = 1,\ F_{k(1,0)} = -C_f(l_c + l_p)/J_{eq},\\
&F_{k(1,1)} = -(2F_s\Delta(0) + B_{eq})/J_{eq},\ F_{k(1,2)} = c_f(lc + lp)/J_{eq},\\
&F_{k(1,3)} = c_f(l_c + l_p)l_f/(J_{eq}v_{CG}),\ F_{k(2,0)} = c_f/(Mv_{CG}),\\
&F_{k(2,2)} = -(c_f + c_r)/(Mv_{CG}),\\
&F_{k(2,3)} = -1 + (c_r l_r - c_f l_f)/(Mv_{CG}^2),\\
&F_{k(3,0)} = c_f l_f / I_z,\ F_{k(3,2)} = (c_r l_r - c_f l_f)/I_z,\\
&F_{k(3,3)} = -(c_r l_r^2 + c_f l_f^2)/(I_z v_{CG})
\end{aligned}
\tag{5.45}
$$

where, $\Delta(.)$ is a 'Kronecker delta' function which is 1 if the argument is zero and 0 otherwise. The coding vectors $CV_k \in \mathcal{R}^4$ were updated as suggested by Lemma 1 and the state check, variance check and statistical bounds for these two quantities were computed using previously described methods.

5.6 Experimental Results

The quadcopter used in this work has following parameters (defined in [141]): $L = 0.3\,m$, $r = 0.1\,m$, $m = 1.2\,kg$, $b = 0.0245$, $d = 10\,in$, and $pitch = 4.5\,in$. The steer-by-wire system used in this work (adopted from [177]) has the following parameters: $J_{fw} = 2.6\,kg.m^2$, $B_{fw} = 12\,Nms/rad$, $J_{eq} = 9.498\,kg.m^2$, $B_{eq} = 24.312\,Nms/rad$, $c_h = 0.2\,Nm/rad$, $N_\theta = 12$, $F_s = 2.68\,Nm$, $C_f = C_r = 45000\,N/rad$, $l_f = 1.2\,m$, $l_c = 0.016\,m$, $l_p = 0.023\,m$, $l_r = 1.05\,m$, $M = 2000\,kg$, $I_Z = 1300\,Kg.m^2$, and $V_{CG} = 10\,m/sec$. The initial coding vector CV_0 (initial CV at $t = 0$) for quadcopter was chosen to be $[1/4, 1/4, 1/4, 1, 1, 1/8, 4, 4, 4, 2, 2, 2]$, and that for SbW system was chosen to be $[1, 1, 1, 1]$. Subsequent coding vectors and check detection thresholds are computed as per the discussion in Section 5.3. The proposed error detection framework scales across a variety of error and failure mechanisms. We discuss the statistical analysis based coding parameter and threshold selection strategy in the following. This is followed by results for transient errors in sensor values, Kalman filter computations and control program execution and offset errors in actuators. Comparison with other state of the art methodologies is discussed and hardware validation experiments are presented.

5.6.1 Statistical analysis based best bound selection:

Nonlinear systems having the form shown in Figure 5.1 are subject to process and measurement noise. Having a check whose threshold is low results in vulnerability to noise. For this reason, the proposed Kalman filter check uses all four checks, namely *'state update check'*, *'algorithmic state check'*, *'variance update check'* and *'statistical variance check'*. In general, the quantity ρ is selected to minimize false positives while maximizing error coverage. Low threshold values result in higher '*false positive*' counts (detection of errors when there are none). On the other hand, high values of threshold (high values of ρ) result in loss of error coverage and hurts performance as well. As a consequence, a proper value of ρ needs to be determined in Equation 5.37 that balances this tradeoff. We use a statistical approach to systematically address this issue. We vary the value of ρ and determine the '*Precision*' and '*Recall*' of the model. '*Precision*' is defined as $\frac{TP}{TP+FP}$ and '*Recall*' is defined as $\frac{TP}{TP+FN}$, where TP, FP, and FN are defined as TP = *True Positive* = the number of errors that cause failures in performance that are detected, FP = *False Positive* = the number of events (error injections) that do not cause failures in performance but are flagged as errors and FN = *False Negative* = The number of events that cause failures in performance but are flagged as error-free. Performance failure for a quadcopter is defined to occur when the L_∞ norm of the difference in the x, y and z co-ordinates of the quadcopter deviates from its intended trajectory by 5 inches, where the diameter of the spherical volume of the quadcopter is 10 inches. Similarly, for the Steer-by-Wire system, performance failure occurs when the lateral acceleration of the vehicle exceeds its maximum permissible bound, (0.4 g at any speed, set by United States Federal Highway Administration [183]).

To determine the best value of ρ for the quadcopter check, we performed 2000 experiments where faults were injected in 1000 experiments (both the fault models and the fault value were selected in random) and the remaining experiments were fault-free. From these experiments, the value of '*Precision*' and '*Recall*' were determined. The '*F1 score,* F_1' was determined from these two quantities which is defined as $F_1 = \frac{2}{Recall^{-1}+Precision^{-1}}$. The maximum value of F_1 corresponds to the best balance between '*Precision*' and '*Recall*' and this represents the selected value of ρ. For the quadcopter, the values of *F1 score*, *Precision*, and *Recall* are plotted in Figure 5.17. As seen from this plots, the value $\rho = 4.1$ corresponds to the best threshold bound.

Similar to the previous experiment, we performed F_1 *score* based analysis for the Steer-by-Wire system, and the values of F_1 *score*, *Precision*, and *Recall* are plotted in Figure 5.18. As can be seen from these plots, $\rho = 3$ corresponds to the best threshold bound.

In the following, we illustrate the detection of failures in sensors, actuators, covariance computation and control program execution with examples. This is followed by a discus-

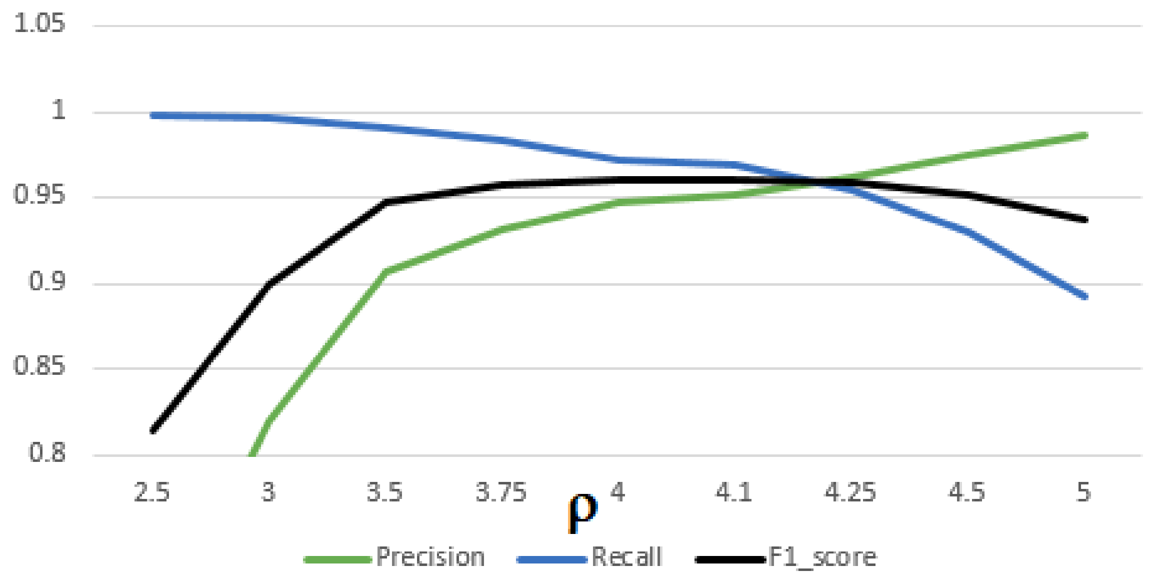


Figure 5.17: F1 score profile for quadcopter system

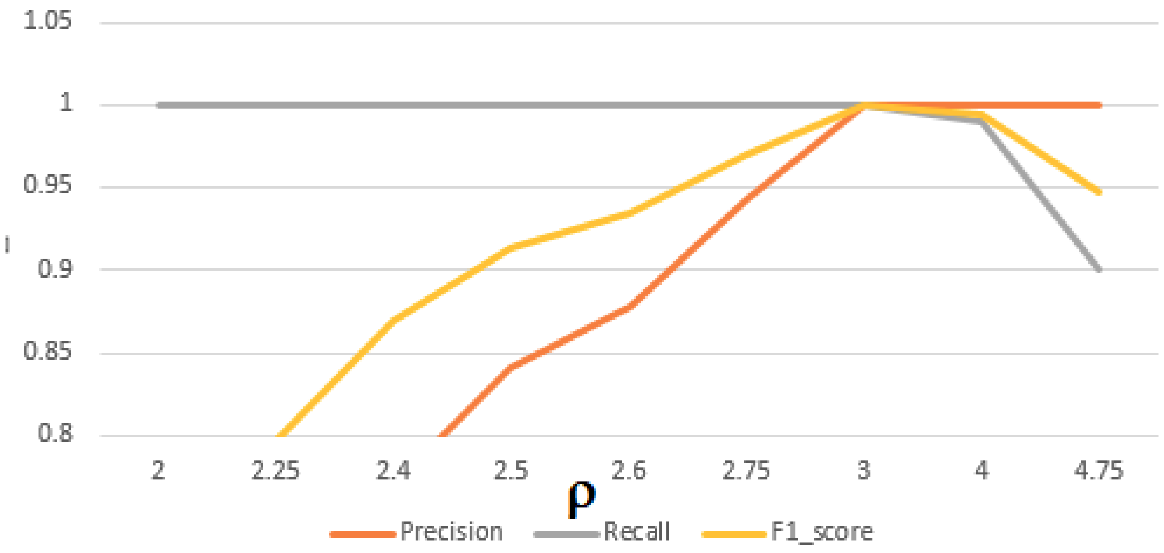


Figure 5.18: F1 score profile for Steer-by-Wire system

sion of the benefits of the proposed methodology vis-a-vis the state of the art.

5.6.2 Sensor Failure

Multiple bit-flips were introduced into the sensor readings (inertial measurement unit (IMU) reading) of the quadcopter from time $t = 3$ sec to $t = 4.2$ sec of a planned flight path. The state check and variance check obtained for this failure are shown in Figure 5.19, and it is observed from the plots that the injected error is almost instantaneously detected at $t = 3$

sec. Similarly, random bit-flips were injected into the sensor readings of the SbW system at $t = 0.7$ sec, and both the state and variance checks are shown in the Figure 5.20. As can be seen from these plots, the error is detected at $t = 0.701$ sec with only 1 ms detection latency. As discussed earlier, both the state check and the variance check are adaptively encoded with time-varying detection thresholds.

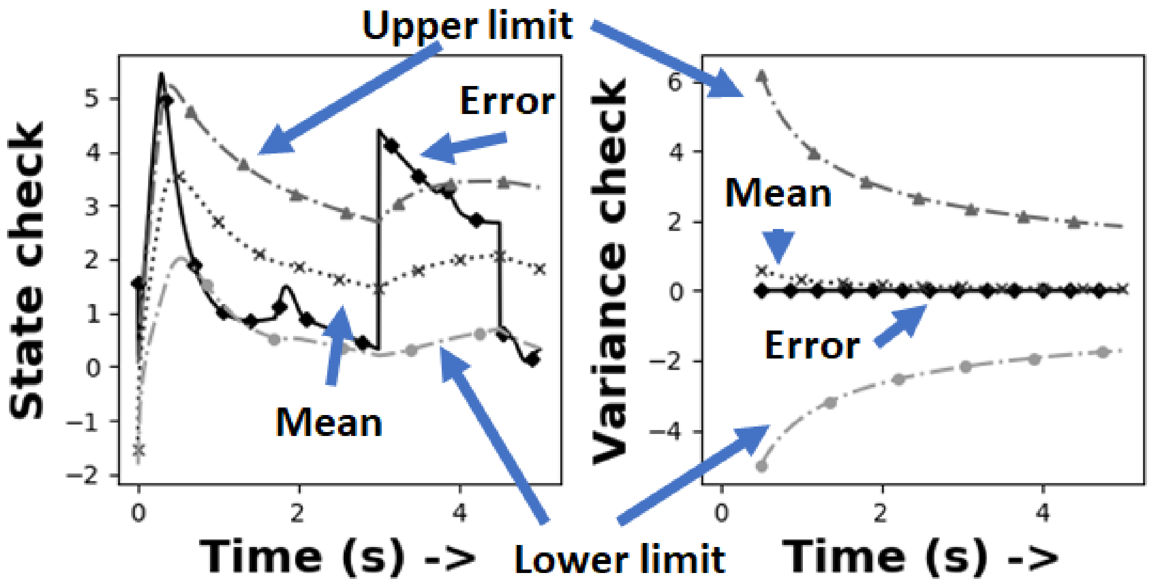


Figure 5.19: Checks for quadcopter system in presence of sensor error

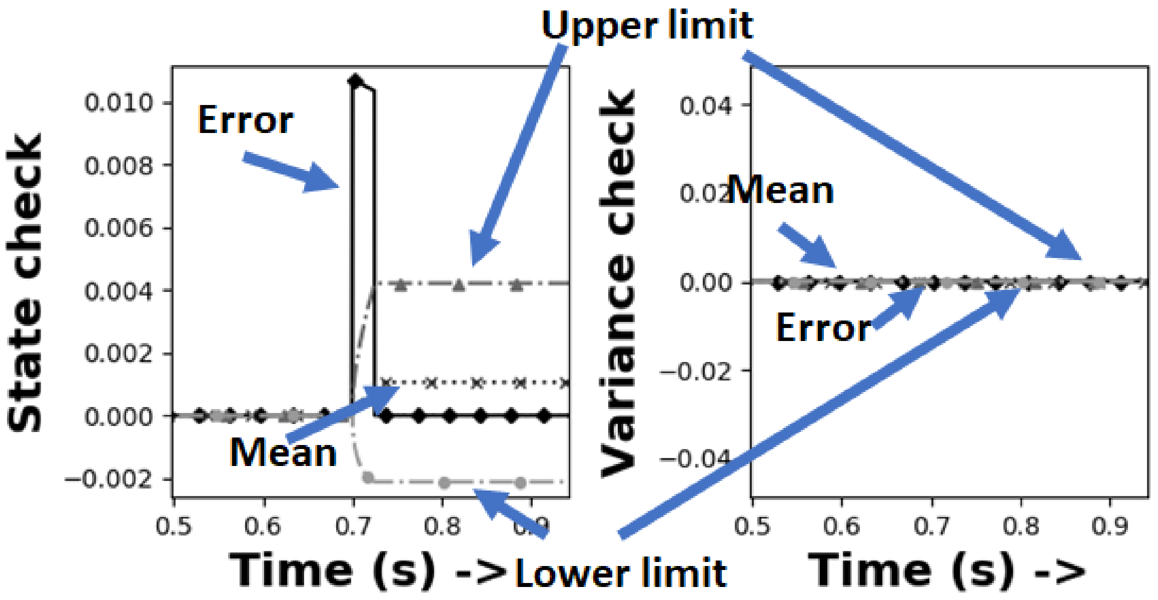


Figure 5.20: Checks for SBW system in presence of sensor error

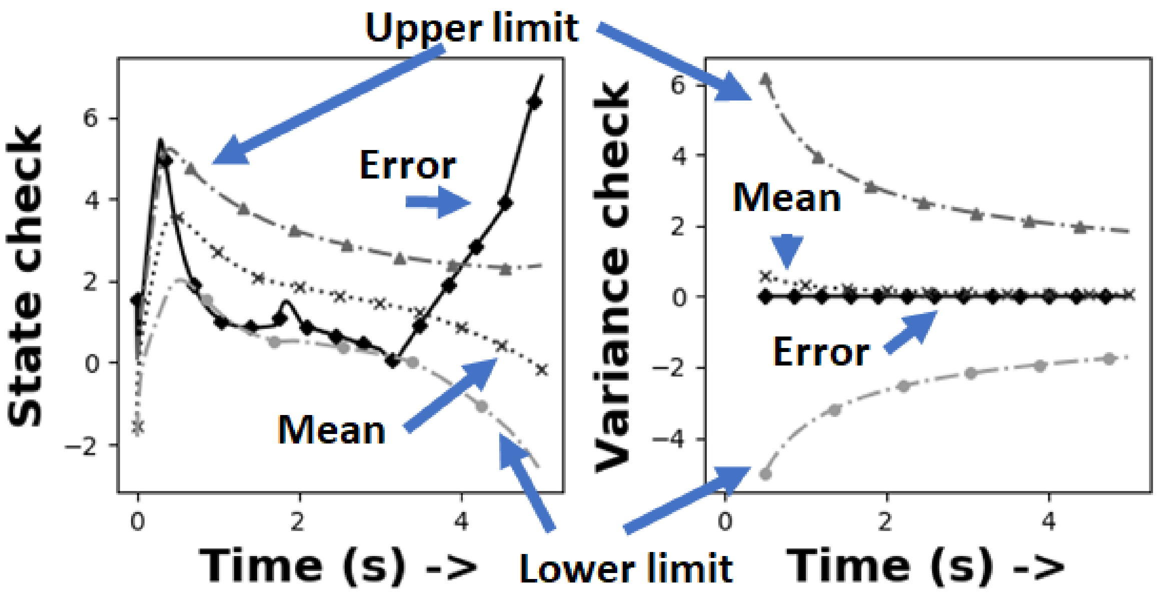


Figure 5.21: Checks for quadcopter system in presence of actuator error

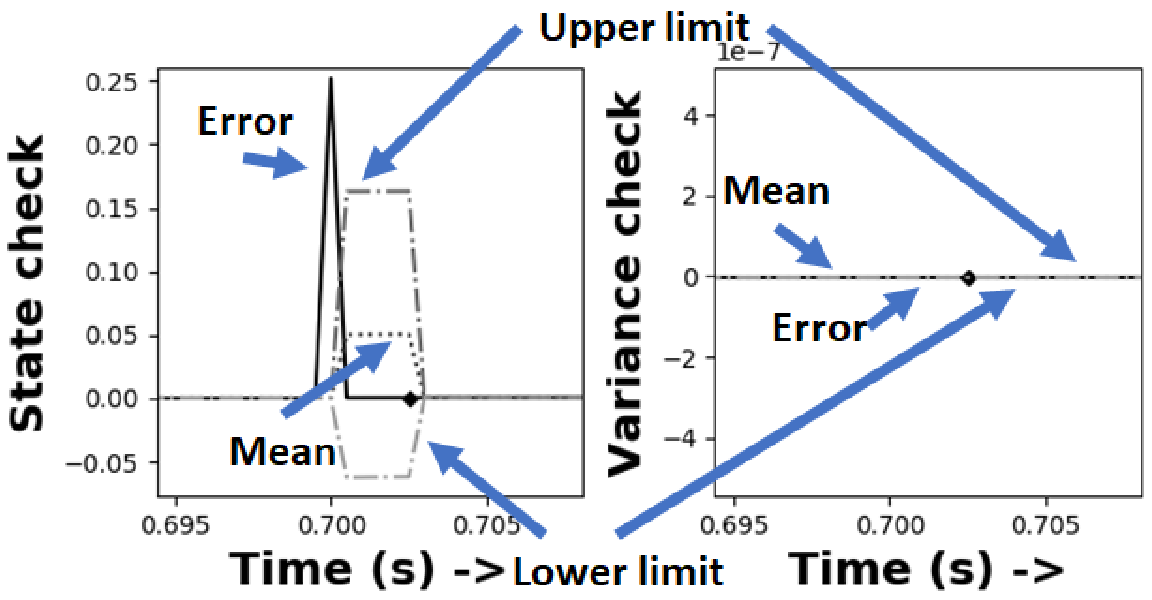


Figure 5.22: Checks for SBW system in presence of actuator error

5.6.3 Actuator Error

Actuator failures for the quadcopter systems are modeled as offsets in actuator values (motor speed). Figure 5.21 shows plots for state and variance checks when the system experiences an offset in actuator output (failure) at $t = 3.15$ sec. The state check shows a clear breach of bounds at $t = 4$ sec which indicates the detection of failure within 0.85 sec. For the SbW system, an offset was introduced into the steering motor (brushless DC motor) output at $t = 0.7$ sec. The corresponding state and variance checks are shown in Figure 5.22 which shows that the error was detected with a latency of only 1 ms. Most other injected failures were detected with low latency (within 10 ms).

5.6.4 Covariance Computation Failure

We also considered computation errors in the covariance matrix block of the Kalman filter. Computation of this matrix is particularly important as the *coding vectors* are scaled according to the diagonal elements of this matrix and errors in computation directly impact the error checking process. Figures 5.23 and 5.24 show plots for the state and variance

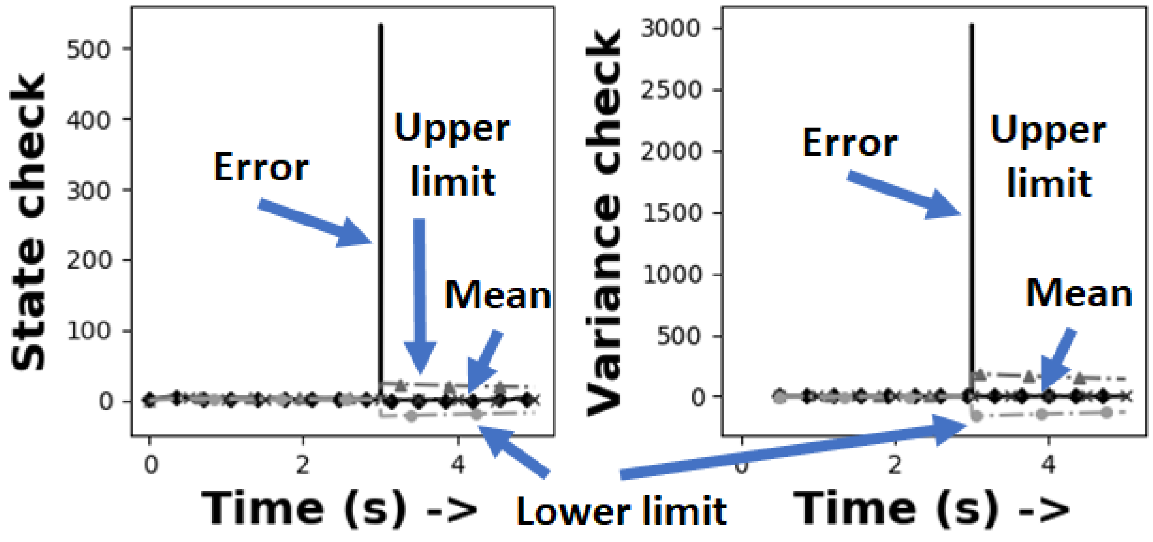


Figure 5.23: State Check for quadcopter system in presence of covariance computation error

checks for the quadcopter and the SbW system respectively, when the covariance matrix stored in memory is corrupted due to soft errors. The errors for the quadcopter system and the SbW system were injected at $t = 3$ sec and $t = 0.7$ sec, respectively. As seen from these plots, both the state and variance checks were able to flag the computation error in real-time at $t = 3.001$ sec and $t = 0.701$ respectively, resulting in corresponding error detection latencies of 1 ms in both cases.

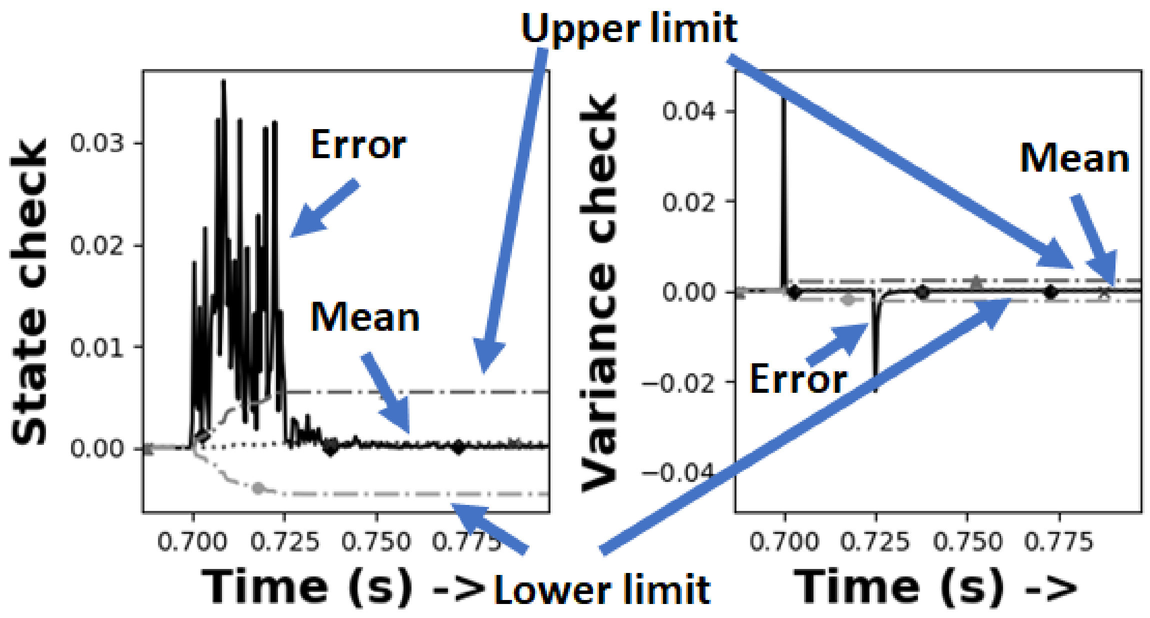


Figure 5.24: State Check for SBW system in presence of covariance computation error

5.6.5 Control Program Error

Soft errors involve fault injection in the controller (digital processor core) block of Figure 5.1. These errors are modeled as spurious bit flips in the processor core on which the control algorithm is executed. In a simulation environment, the plant, controller and checking algorithms are coded together in a combined manner and executed on a common processor platform. However, in reality, only the controller and the fault detection methodologies are executed on an embedded processor since the rest of the simulation software mimics the actual physical behavior of the system (plant). For this reason, the location of the control action computations was analyzed carefully in assembly level code and faults were injected into the registers involved in control action computations. The range of least significant bits of the registers which are affected by error injection is referred to as the '*injection range*'. 1000 injection experiments for single and multiple bit error each were performed and for each set of experiments, the number of errors detected and number of experiments that resulted in silent data corruption (SDC) are shown in Tables 5.1 and 5.2 respectively. As seen in this case, the lower value of injection range for both multiple and single bit-flips resulted in low value of errors detected (52.2% and 0% respectively) and potential crashes,

Table 5.1: Error detection experiments on quadcopter system

	Multiple bit-flip		Single bit-flip	
Injection range	Potential crashes with 100% detection	# of SDC	Potential crashes with 100% detection	# of SDC
32	846	154	542	458
16	576	424	68	932
8	522	478	0	1000

Table 5.2: Error detection experiments on SbW system

	Multiple bit-flip		Single bit-flip	
Injection range	Errors detected	# of SDC	Errors detected	# of SDC
32	864	136	627	373
16	766	234	421	579
8	100	900	21	979

as these bit-flips did not have strong manifestation into system failure. Additionally, single bit-flips resulted in fewer errors in all experiments. For silent data corruption, the errors do not change the performance of the system and hence the proposed check did not detect those events. However, as the injection range increases, the system failure also increases which are detected by the proposed approach. As seen from the Table 5.1, the error detection accuracy increases to 84.6% (for multiple bit error) and 54.2% (for single bit error) when injection range is 32. For the SbW case, similar to the quadcopter case, fewer multiple and single bit-flip errors were detected at low injection range of 8 (10% and 2.1% respectively), which increases to 86.4% and 62.7%, respectively when injection range is 32 as reported in Table 5.2.

5.6.6 Comparison with state of the art detection methods

In the following, we compare the proposed error detection methodology using Kalman filter checks with variable coding vectors and variable thresholds against three most relevant error detection techniques: (a) use of Kalman filter checks with constant coding vector and constant threshold [175], (b) use of a machine learning based state encoding technique proposed in [83], and (c) use of an encoded residual generator [184]. In all of (a), (b), and (c), linear sums of the system states were used for state encoding and the state encoding was constant over time. Also, in all of (a), (b), and (c), fixed time-independent error detection thresholds were used. The error threshold was set to worst case magnitude of the coding error determined for a fault free system under anticipated operating conditions. We considered faults in sensors (accelerometer and gyroscope, selected in random), actuators, control action computation, and EKF computation. We performed 1000 experiments for each of these failure modes and simulated each of the above error detection techniques under the same set of operating conditions. For each of these combinations, we determined detection accuracy (defined as $\frac{Number\,of\,true\,events\,detected}{Number\,of\,total\,events}$) and average detection latency. The results are given in Table 5.3.

As observed from Table 5.3, our proposed approach had superior or comparable performance to the two other state of the art error detection methods in the majority of metrics considered. In each case, the proposed technique was overwhelmingly superior to the machine learning based checking approach of [83]. In only one case, the Steer-by-Wire actuator detection latency was 44.2 ms vs. 27.2 ms for the Kalman check with constant coding vector. This is due to the extra computations needed by the proposed approach (can be fine tuned by code optimization) but allows significant improvements in failure coverage (98.2% vs. 7.5%).

We also performed a comparative study or detection accuracy and latency among all four checks - State update check, Algorithmic state check, Variance update check and Statistical variance check for both quadcopter and steer-by-wire system. The results are shown in Tables 5.5 and 5.6, respectively.

This clearly demonstrates that with proper design of coding vector and threshold bounds, the check can be implemented with very low computation and memory overhead and can perform in almost real-time.

Table 5.3: Comparative study for quadcopter system

Comparison metric	This research [185]	Kalman check accross time [175]	Kalman check with constant CV and Threshold [175]	Machine learning based approach [83]	Encoded residual generator [184]
Comparison of detection accuracy					
Sensor fault	99.7%	99%	97.2%	99.1%	86%
Actuator fault	83.7%	53.0%	12.5%	83.3%	57%
Control program fault	95%	50 %	97.2%	88%	85%
EKF computation fault	100%	100 %	37.0%	NA	89%
Comparison of average detection latency					
Sensor fault	$<$ 1 ms	4ms	3.8 ms	1.9 sec	1.08 sec
Actuator fault	0.28 sec	1.61 sec	1.9 sec	1.4 sec	2.78 sec
Control program fault	0.21 sec	2.78 sec	0.28 sec	0.2 sec	1.65 sec
EKF computation fault	10 ms	16 ms	2.32 s	NA	1.89 sec
Other comparisons					
Training time for machine learned model*	NA	NA	NA	15 min	NA
Computation Overhead	21.17 %	20.21 %	21.05 %	646.88 %	0.1%
Memory Overhead	15.81 %	15.81%	15.81 %	4842.79 %	1.4%

*Training was done in Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 16GB RAM, without GPU

5.6.7 Hardware validation

We implemented the *Kalman Filter Check* on a '*Crazyflie 2.1*' [142] quadcopter system (see Figure 5.25). The hardware configuration of the system is shown in Table 5.7. The

Table 5.4: Comparative study for Steer-by-Wire system

Comparison metric	This research [185]	Kalman check accross time [175]	Kalman check with constant CV and Threshold [175]	Machine learning based approach [83]	Encoded residual sensor [184]
Comparison of detection accuracy					
Sensor fault	100%	100%	76.2%	99.6%	60 %
Actuator fault	98.2 %	98%	7.5%	9.2%	98 %
Control program fault	100%	94%	2%	2.2%	21 %
EKF computation fault	100%	100 %	11.2%	NA	93%
Comparison of average detection latency					
Sensor fault	4 ms	135 ms	40.9 ms	501 ms	61 ms
Actuator fault	44.2 ms	146 ms	27.2 ms	252 ms	176 ms
Control program fault	65 ms	194 ms	146.5 ms	605 ms	654 ms
EKF computation fault	4 ms	6 ms	36 ms	NA	167 ms
Other comparisons					
Training time for machine learned model*	NA	NA	NA	9 min 10 sec	NA
Computation Overhead	61.76 %	60.56 %	60.56 %	589.44%	0.67%
Memory Overhead	43.75 %	43.75 %	43.75 %	4902.08%	2.1%

*Same computer mentioned in Table 5.3 was used here

Crazyflie quadcopter communicates with a PC via Bluetooth interface and a propriety Crazy RealTime Communication Protocol (CRTP). The server and client platforms for the setup were implemented using python and C respectively. The server was used only to send commands to the client and all the necessary computations related to control action gen-

Table 5.5: Comparative study among different checks for quadcopter system

Comparison metric	State update check	Algorithmic state check	Variance update check	Statistical variance check
Comparison of detection accuracy				
Sensor fault	99.7%	99%	0%	0%
Actuator fault	83.7 %	53%	0%	0%
Control program fault	95%	50%	0%	0%
EKF computation fault	100%	98 %	100%	100%
Comparison of average detection latency				
Sensor fault	1 ms	4 ms	NA	NA
Actuator fault	0.28 sec	1.61 sec	NA	NA
Control program fault	0.21 sec	2.78 sec	NA	NA
EKF computation fault	33 ms	165 ms	10 ms	16 ms

Table 5.6: Comparative study among different checks for Steer-by-Wire system

Comparison metric	State update check	Algorithmic state check	Variance update check	Statistical variance check
Comparison of detection accuracy				
Sensor fault	100%	100%	0%	0%
Actuator fault	98.2 %	98%	0%	0%
Control program fault	100%	94%	0%	0%
EKF computation fault	100%	100 %	100%	100%
Comparison of average detection latency				
Sensor fault	4 ms	135 ms	NA	NA
Actuator fault	44.2 ms	146 ms	NA	NA
Control program fault	65 ms	194 ms	NA	NA
EKF computation fault	8 ms	311 ms	4 ms	6 ms

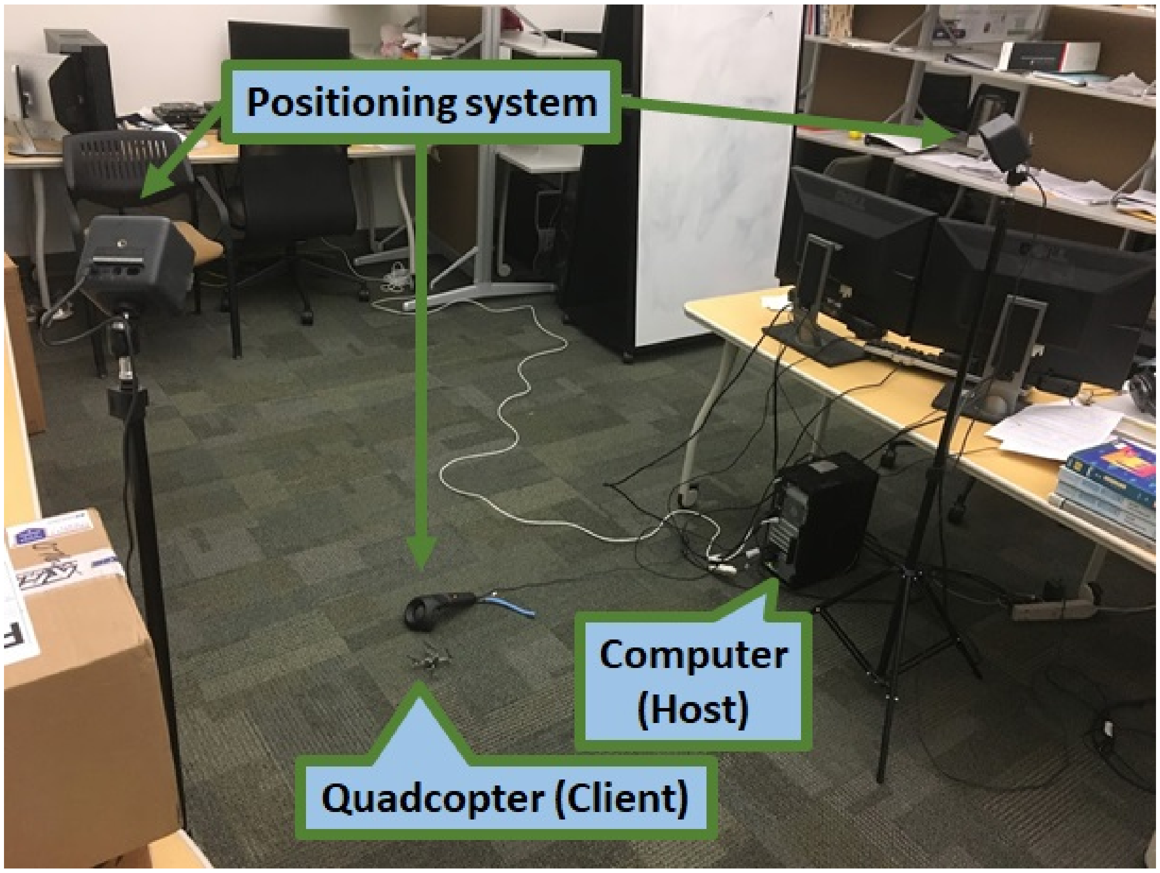


Figure 5.25: Hardware setup

Table 5.7: Hardware configuration of Crazyflie 2.1

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

eration, necessary system level consistency check for the client, Kalman filter based state estimation, our proposed checks etc. were implemented in the CrazyFlie Microcontroller unit (MCU). The client 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. Failures were injected into the accelerometer, gyroscope, actuator circuit, and control program of the quadcopter and the obtained results are discussed below:

Gyroscope fault

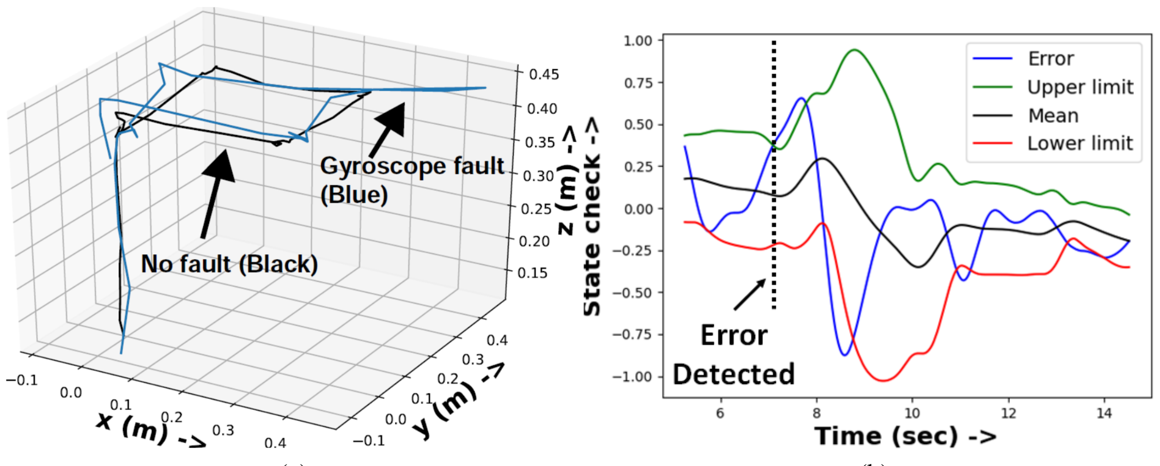
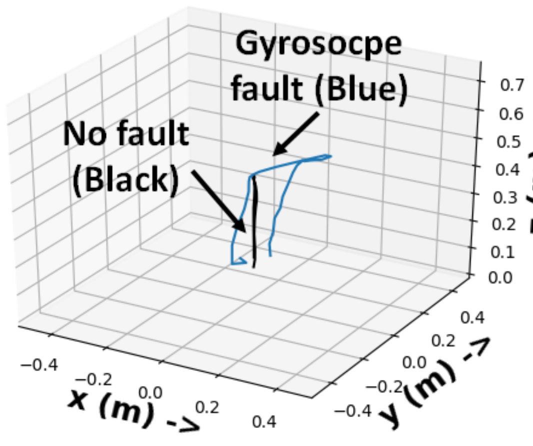


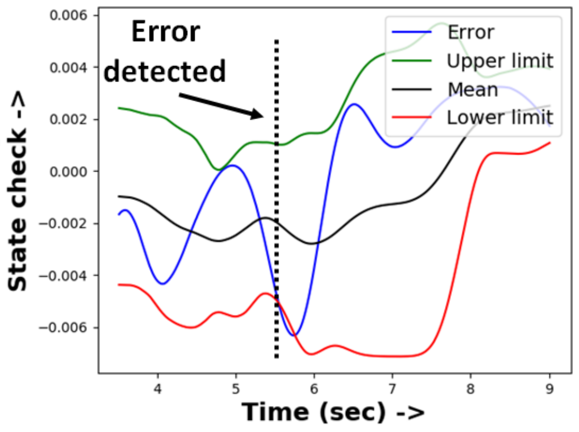
Figure 5.26: (a) Trajectory of the quadcopter under gyroscope fault and (b) corresponding error plot

Offsets in gyroscope readings (the most common form of gyroscope faults) were used to emulate gyroscope failures. The planned trajectory of the quadcopter under nominal conditions and with gyroscope fault are shown in Figure 5.26a. The corresponding state check, $e_x(k)$ is shown in the Figure 5.26b. From the error plot, it is seen that the fault was injected at time $t = 7.11$ sec and the check produced a non-zero output above the threshold at time $t = 7.12$ sec and the error was detected.

The trajectories and state checks are shown for gyroscope failure in Figures 5.27 and 5.28 where the system is going through relatively idle (hovering) condition to more sharp turn. As seen from these plots, the proposed check was able to detect the failure real-time for hovering condition at time $t = 5.50$ sec and for sharp turn at time $t = 6.02$ sec

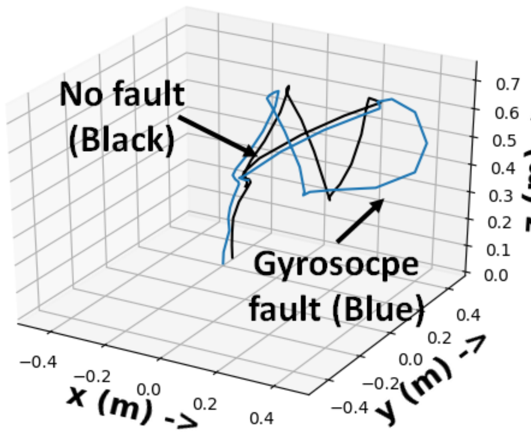


(a)

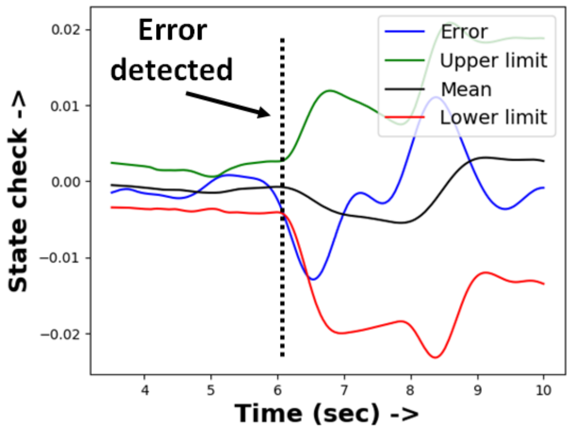


(b)

Figure 5.27: (a) Trajectory of the quadcopter under gyroscope fault and (b) corresponding error plot at hovering condition



(a)



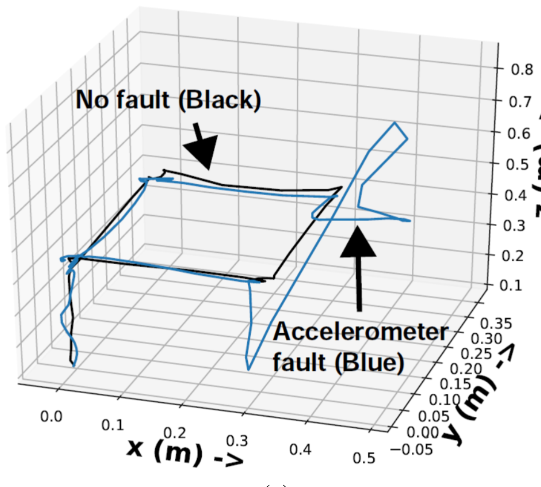
(b)

Figure 5.28: (a) Trajectory of the quadcopter under gyroscope fault and (b) corresponding error plot at sharp turn

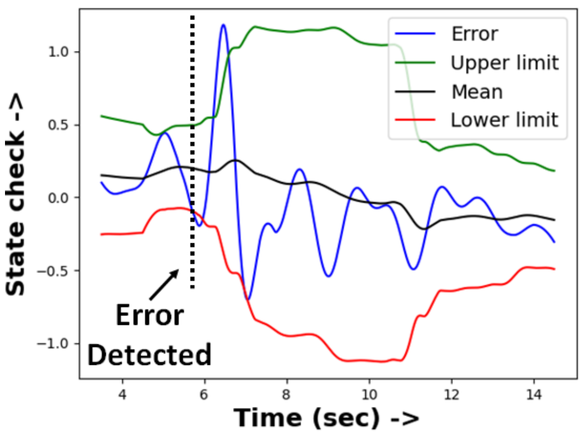
Accelerometer fault

Accelerometer faults were emulated and an offset (the most common form of accelerometer faults) was introduced in the accelerometer reading at $t = 5.4$ sec. It was successfully detected at $t = 5.71$ sec as seen at Figure 5.29b with an error detection latency of 0.31 sec.

Similar to gyroscope case, we introduced failure in the accelerometer during hovering condition and sharp turn. The trajectories and checks are shown in Figures 5.30 and 5.31, respectively. The fault in accelerometer introduced non-zero error value which was

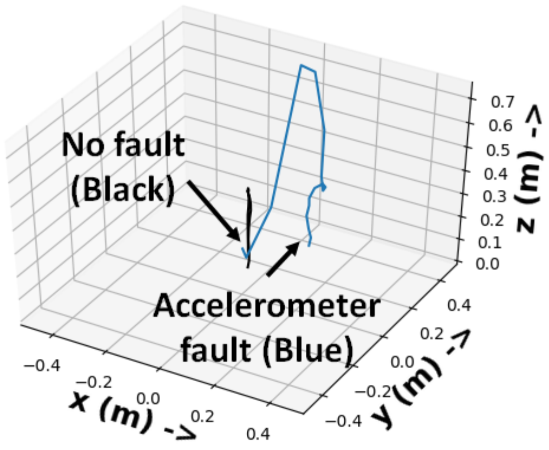


(a)

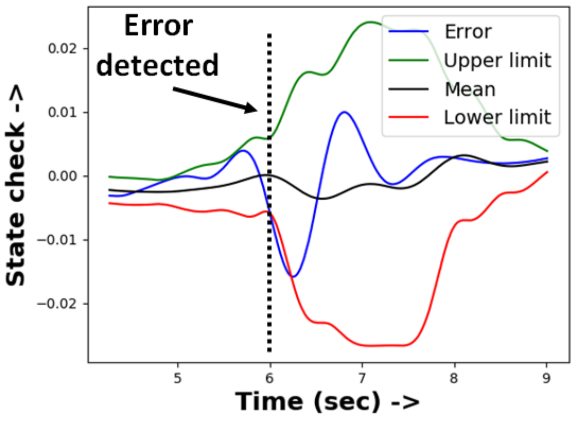


(b)

Figure 5.29: (a) Trajectory of the quadcopter under Accelerometer fault and (b) corresponding error plot



(a)

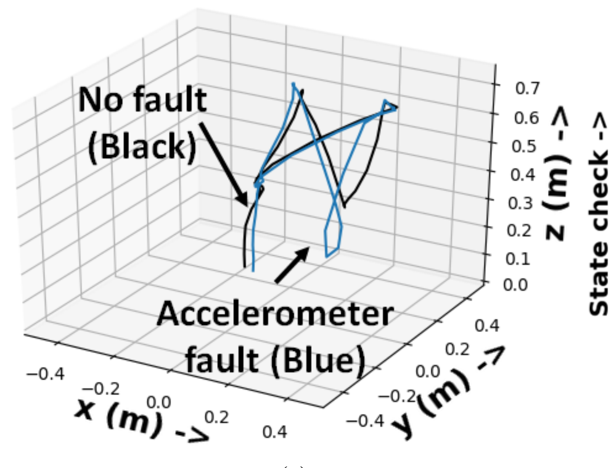


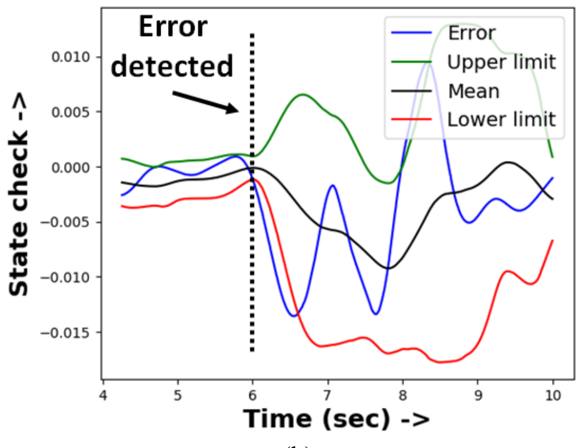
(b)

Figure 5.30: (a) Trajectory of the quadcopter under Accelerometer fault and (b) corresponding error plot at hovering condition
detected instantaneously at time $t = 5.99$ sec and $t = x5.99$ sec, respectively.

Actuator fault

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 DC motors. In this experiment, we introduced temporary speed changes in one of the DC motors which results in change of thrust. The fault was injected at $t = 8$ sec

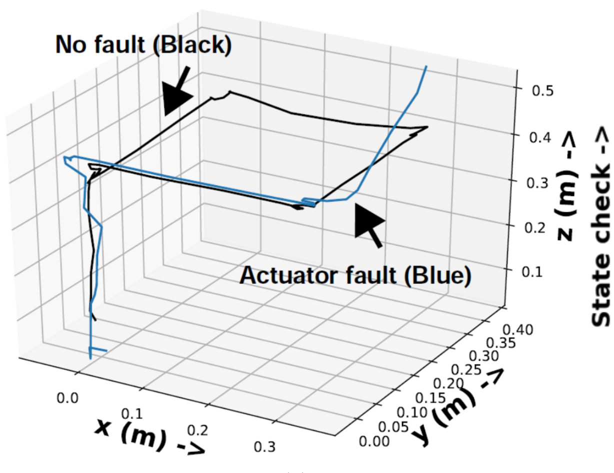


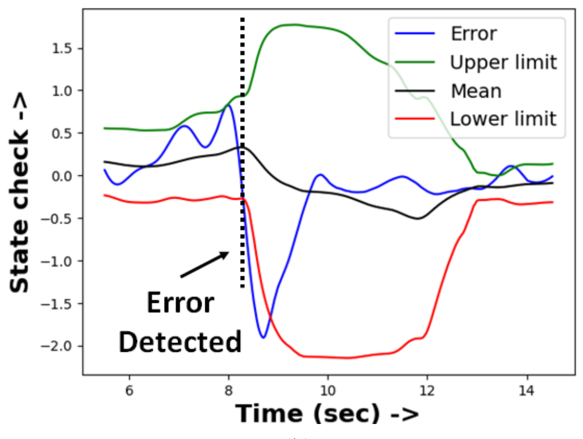


(a) (b)

Figure 5.31: (a) Trajectory of the quadcopter under Accelerometer fault and (b) corresponding error plot at sharp turn

and was detected by the state check at $t = 8.25$ sec as seen in Figure 5.32b.

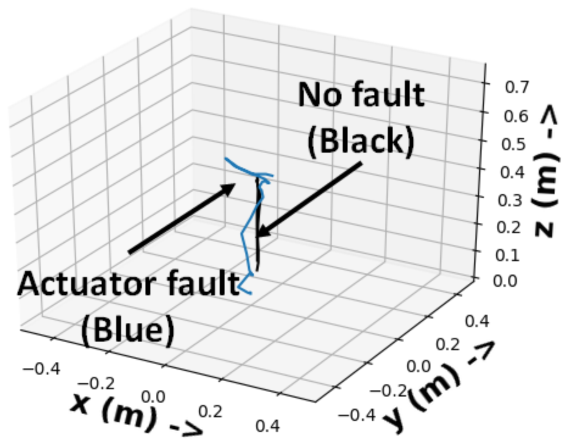




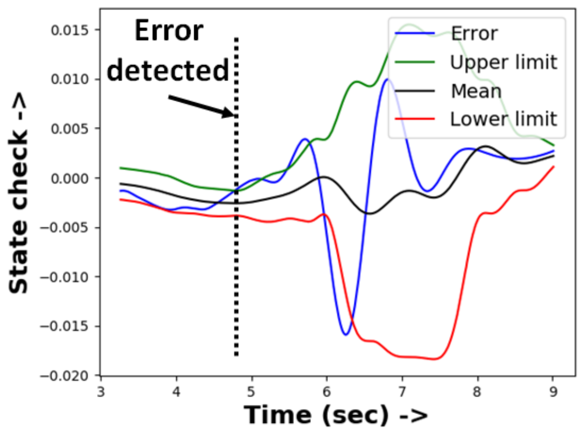
(a) (b)

Figure 5.32: (a) Trajectory of the quadcopter under Actuator fault and (b) corresponding error plot

The error detection experiment was repeated for actuator failure during hovering condition and sharp maneuver in Figures 5.33 and 5.34, respectively. The system shows different trajectories in presence of failure which is detected by the proposed check successfully at time $t = 4.75$ sec and $t = 6.41$ sec, respectively.

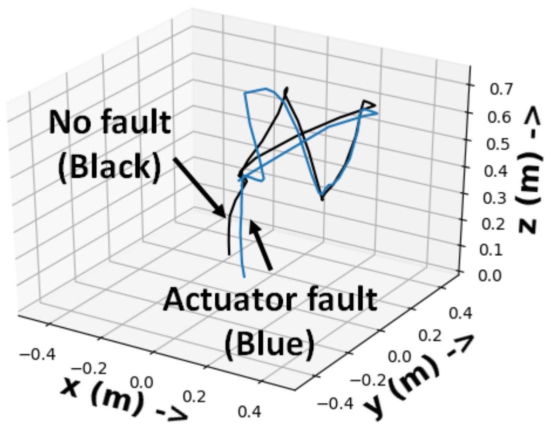


(a)

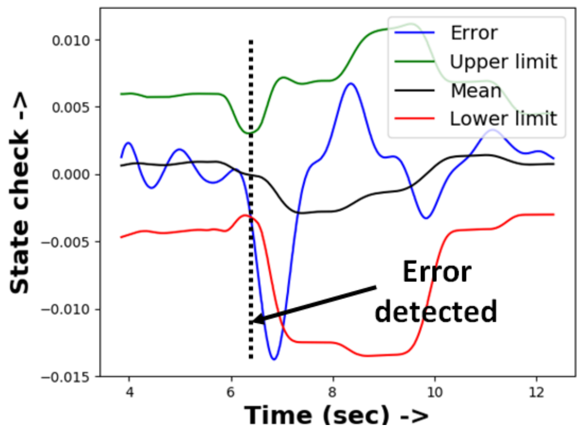


(b)

Figure 5.33: (a) Trajectory of the quadcopter under Actuator fault and (b) corresponding error plot at hovering condition



(a)

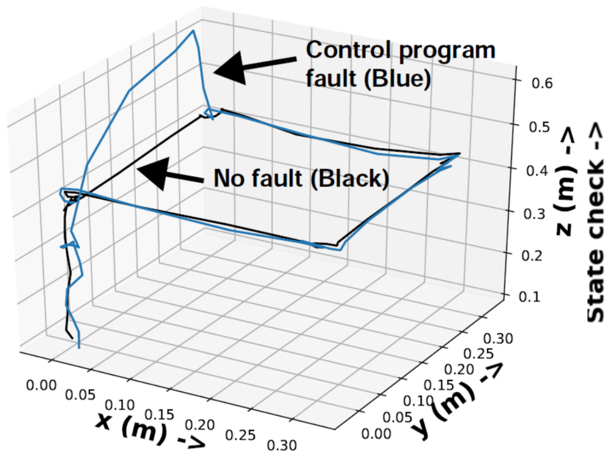


(b)

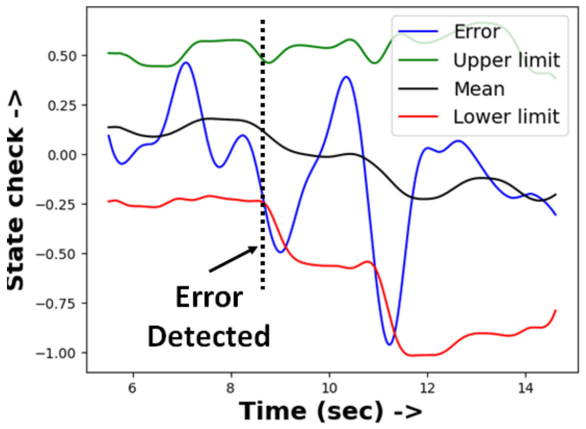
Figure 5.34: (a) Trajectory of the quadcopter under Actuator fault and (b) corresponding error plot at sharp turn

Control program fault

We introduced failure in control program execution in the form of spurious bit-flips in the processor core. Introduction of spurious bit-flips results in incorrect quadcopter control action. In this experiment, a fault was introduced at time $t = 8.23$ sec and caused unwanted deviation of the quadcopter from its desired trajectory. As seen from the Figure 5.35b, the proposed check was able to capture this at $t = 8.69$ sec and the fault was successfully detected. The same experiment was repeated for different trajectories as sown in Figures

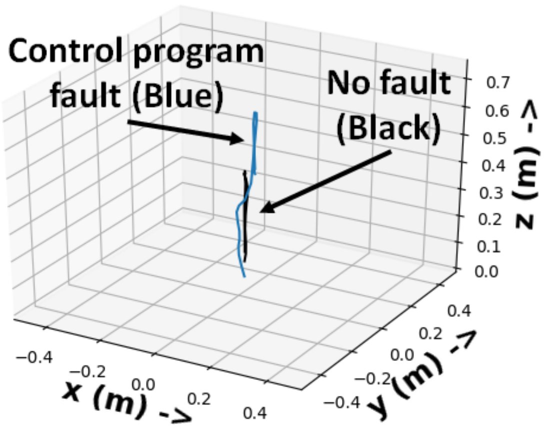


(a)

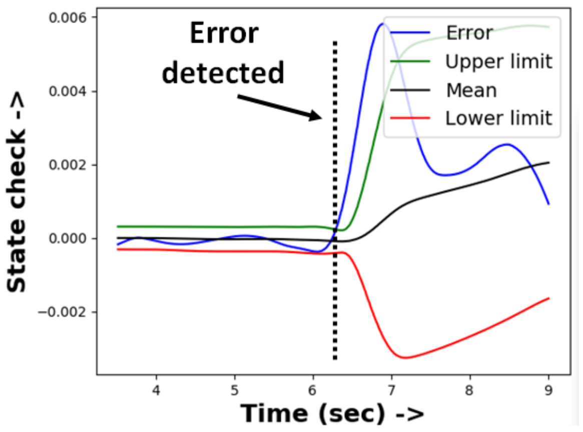


(b)

Figure 5.35: (a) Trajectory of the quadcopter under Control program fault and (b) corresponding error plot



(a)



(b)

Figure 5.36: (a) Trajectory of the quadcopter under Control program fault and (b) corresponding error plot at hovering condition

5.36 and 5.37 for hovering condition and sharp maneuvers, respectively. The proposed check was able to detect the failure at $t = 6.31$ sec and $t = 5.90$ sec, respectively.

5.6.8 Simulation result: error correction

Error diagnosis

The methodology of the error diagnosis is explained in Subsection 5.4.1. A support vector machine based model was trained with $\gamma = 0.1$ and $C = 1$ was used which results into a classification score of 100%. The observation window length of the check was determined

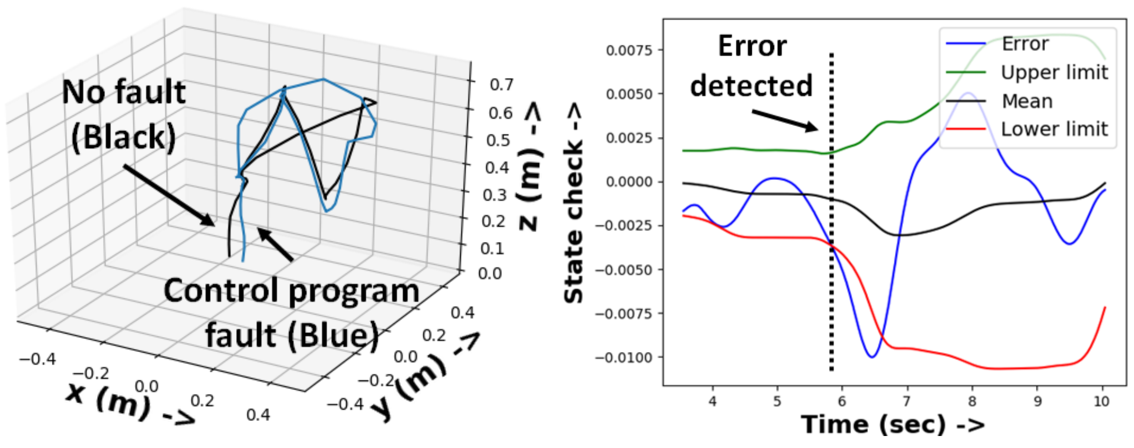


Figure 5.37: (a) Trajectory of the quadcopter under Control program fault and (b) corresponding error plot at sharp turn

from a trade-off study and was found to be 100 samples (equivalent to 0.100 sec) as shown in Figure 5.38.

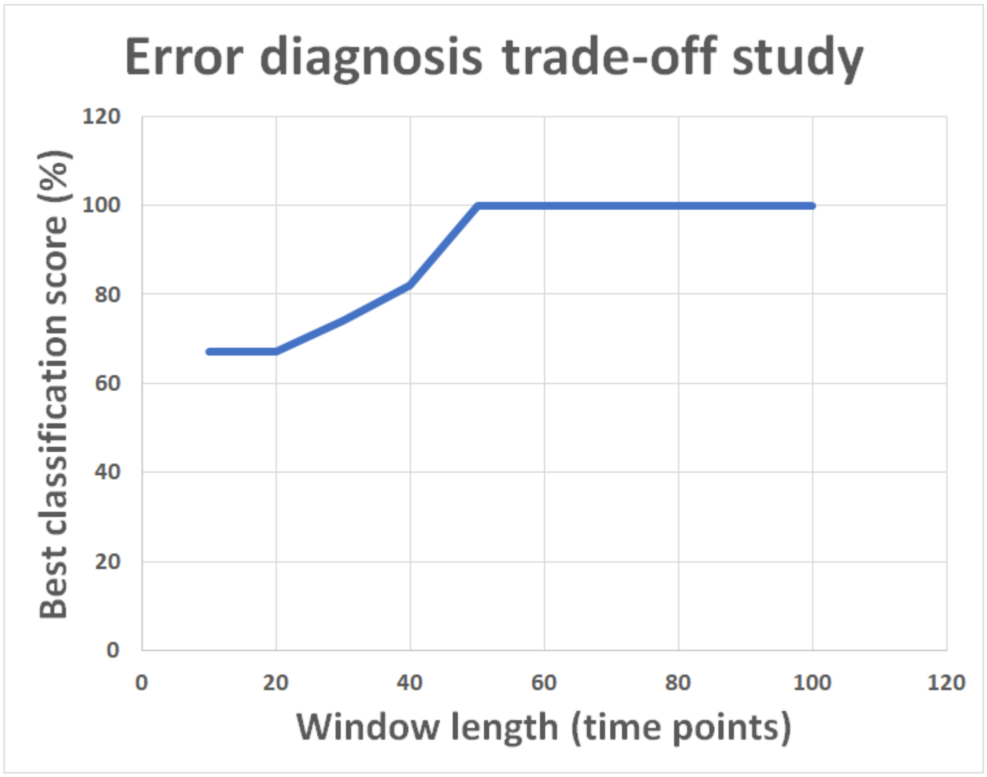


Figure 5.38: Error diagnosis trade-off study

Sensor fault

Gyroscope offset [186] was considered as the first failure model for this experiment. An offset was introduced in the sensor which corrupts the sensor behavior resulting into incorrect behavior of the whole system. Figure 5.39 shows the trajectories of the system at fault-free, faulty without correction and faulty with correction cases. The fault was introduced at $t = 4.5$ sec which introduces deviation in the flight path. However, it was detected and diagnosed in real-time and the parameter estimation was performed by solving the optimization problem explained before. Once the estimated parameter value is known, the sensor reading is corrected and the proposed correction approach was able to improve the system behavior in presence of fault.

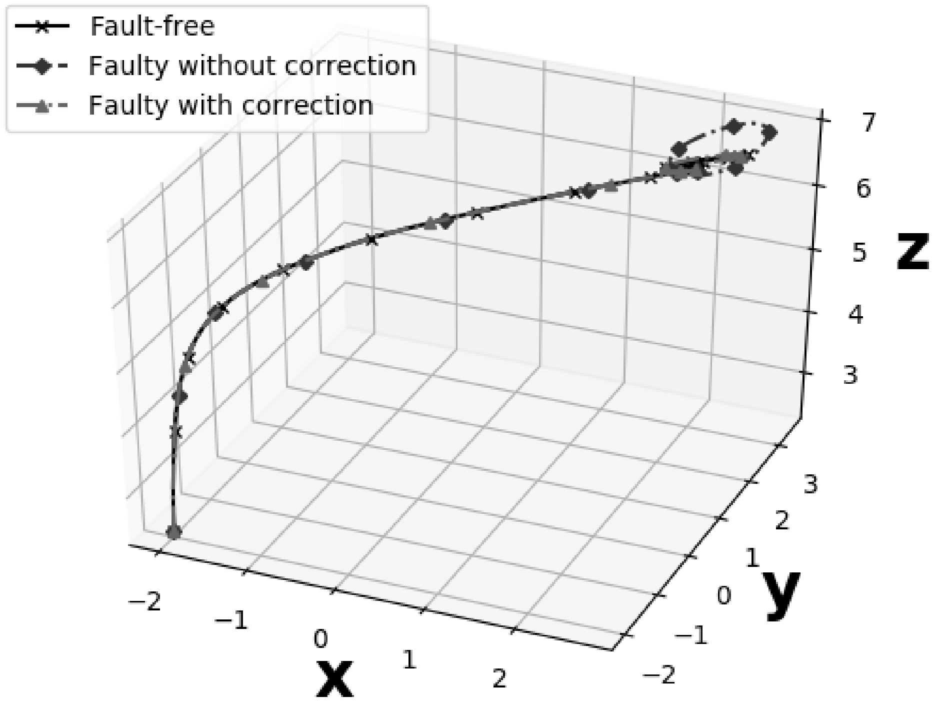


Figure 5.39: Error correction of gyroscope parametric failure

Accelerometer fault was investigated as the next failure model. Failure was introduced as the form of offset which made the quadcopter go totally different (possibly dangerous) direction. Figure 5.40 shows the trajectories of the quadcopter in fault-free, faulty without correction and faulty with correction cases. Once the error was detected and diagnosed, the already saved check data was used to estimate the changed parameter value and correction was performed. As can be observed from these plots, the faulty system goes through different trajectories; however, the proposed approach was able to restore the system behavior.

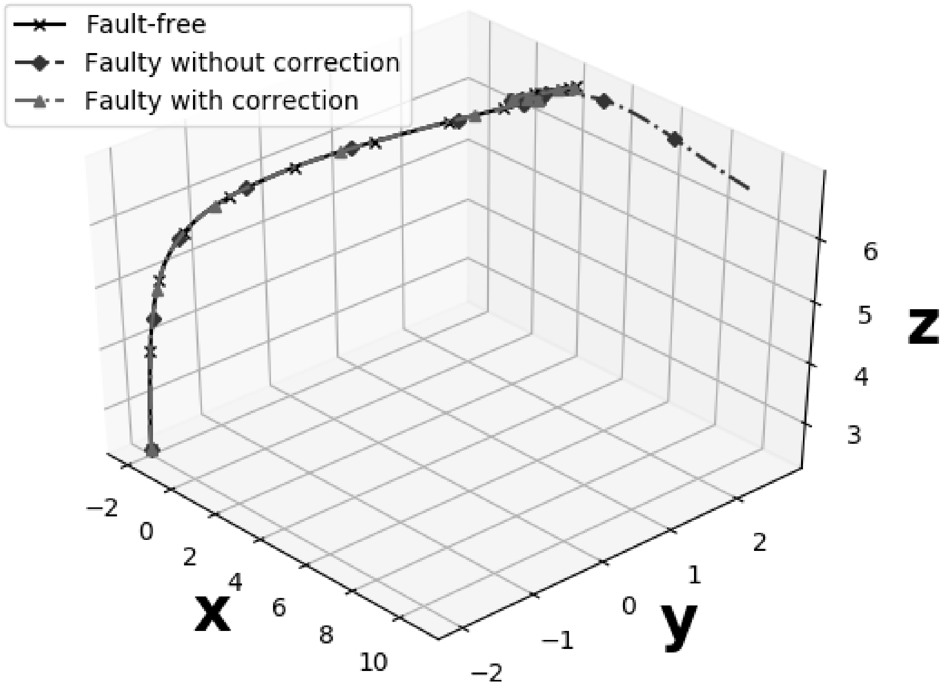
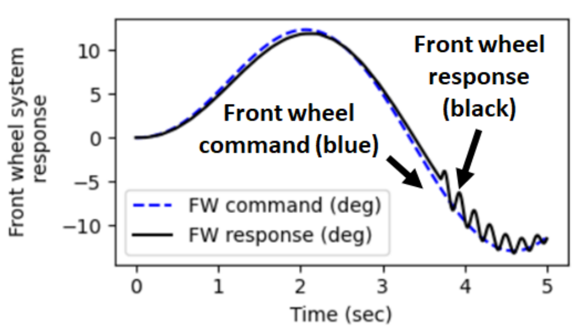
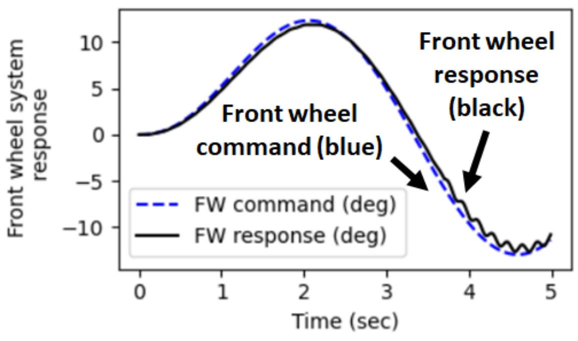


Figure 5.40: Error correction of accelerometer parametric failure

For the SbW test-case, system has a single Hall effect sensor to measure the absolute angle of steering in the wheels. The SbW sensor measures angle using two Hall sensors placed perpendicular to one another on the steering shaft [187]. The Hall effect uses the rotation of a ferromagnetic wheel on the shaft to induce voltages in the sensors' probes. We have injected faults into the Hall sensors of the SbW, considering scaling and offset errors in the current or voltage values measured by the sensors propagating to their final computed angle or angular velocity values. Figure 5.41a shows the trajectories of the SbW system in presence of sensor failure. As shown in the plot, the system behavior changes (lateral acceleration is observed) in presence of failure which can dangerous in high speed maneuvers. However, once the error was detected and diagnosed, the parameter estimation was performed and the corrective action is applied. The corrected trajectory is shown in Figure 5.41b, and as can be observed from this plot, the system behavior was compensated.



(a)



(b)

Figure 5.41: (Left) Trajectory of the steer-by-wire system under sensor fault and (right) corrected trajectory

Actuator error

We consider the parametric deviation of actuator as the third example. For parametric failures of actuators, the back emf constant was gradually reduced to 70% as the failure model. 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.

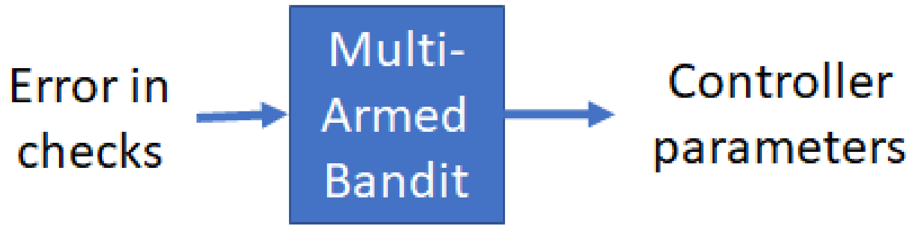


Figure 5.42: Block diagram of MAB framework

We correct this failure by reconfiguring the controller parameter by ‘Multi-Armed Bandit’ (MAB) based learner. A block diagram of the approach is shown in Figure 5.42, and in Figures 5.43 and 5.44, we show the reward per episode plot of MAB experiments. Here, we have implemented a epsilon-greedy MAB framework [188] with learning rate = 0.25,

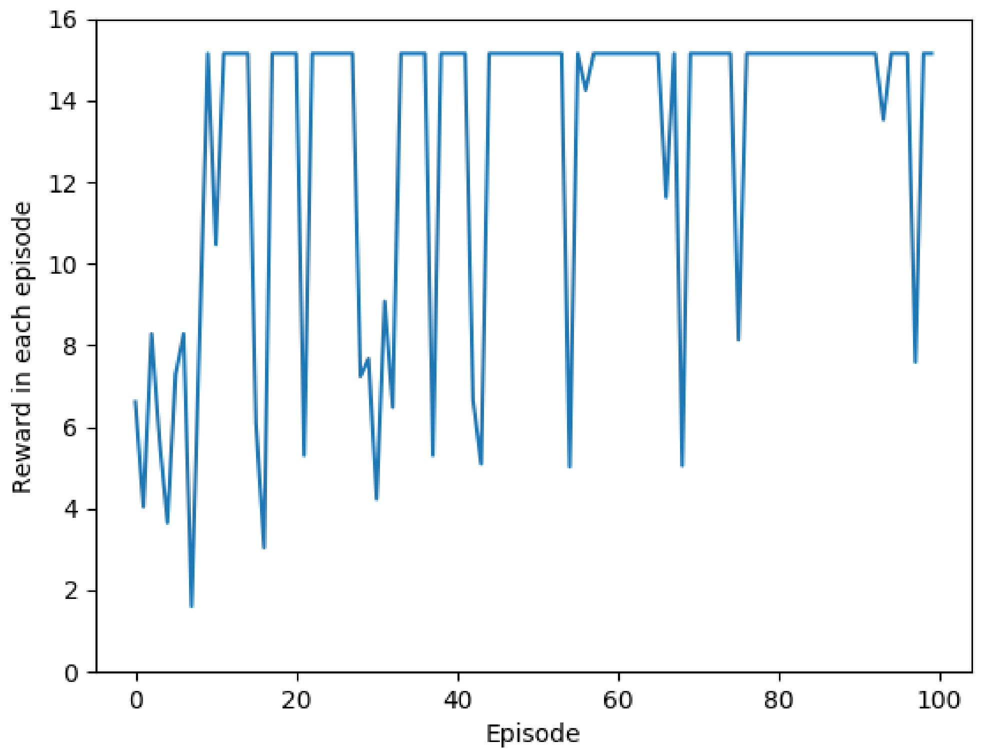


Figure 5.43: Reward per episode plot of MAB learner for ‘cold start configuration’

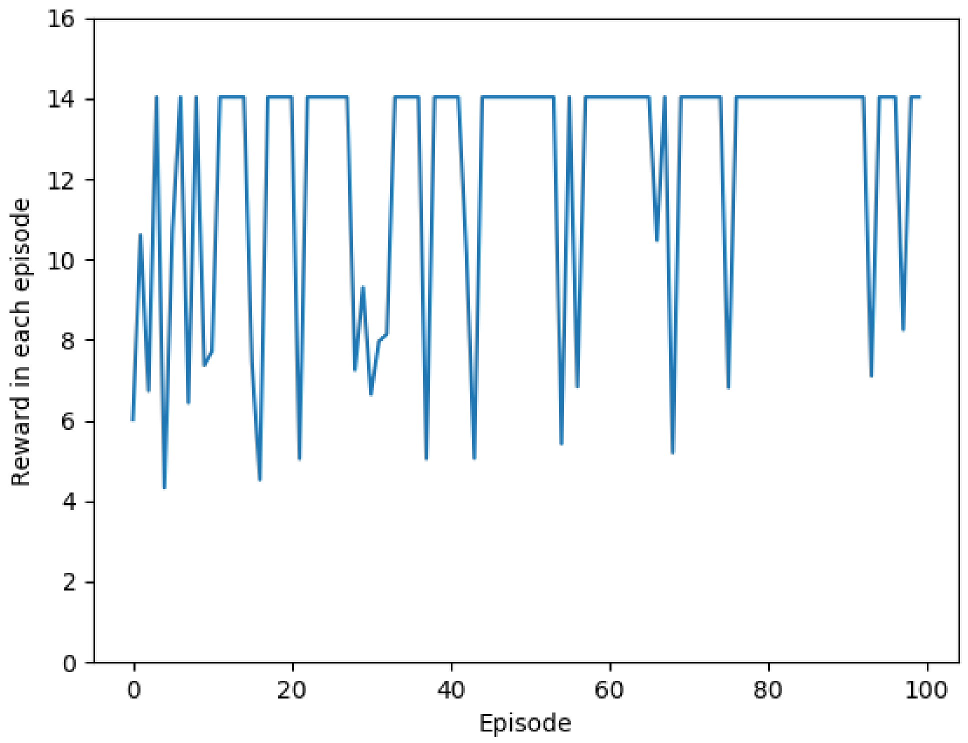


Figure 5.44: Reward per episode plot of MAB learner for ‘hot start configuration’

discount rate = 0.99, control action granularity = 0.1 $\times$ nominal controller parameter. The framework starts by selecting purely random control parameter at '*Cold start configuration*' and tries to match the sensor readings with ideal behavior. However, it learns the best control parameter very quickly when we see the framework is selecting the best controller parameter and achieving best reward/episode. After the learner is deployed, the learning still continues and it re-adapts to new control parameter once the actuator parameter changes. This later process happens in '*Hot start configuration*' to save learning time. Due the epsilon-greedy nature of the framework, in both cases, the learner is still selecting some random action some time and in those trials, the rewards are reduced. We show these experiments for two parametric failures in Figures 5.43 for '*Cold start configuration*' and 5.44 for '*Hot start configuration*', and as can be seen from these Figures and the required iterations, for both cases, the MAB framework was able to find the best control parameters rapidly with minimal number of iterations. The trajectories of fault-free, faulty without correction and faulty with correction cases are shown in Figure 5.45.

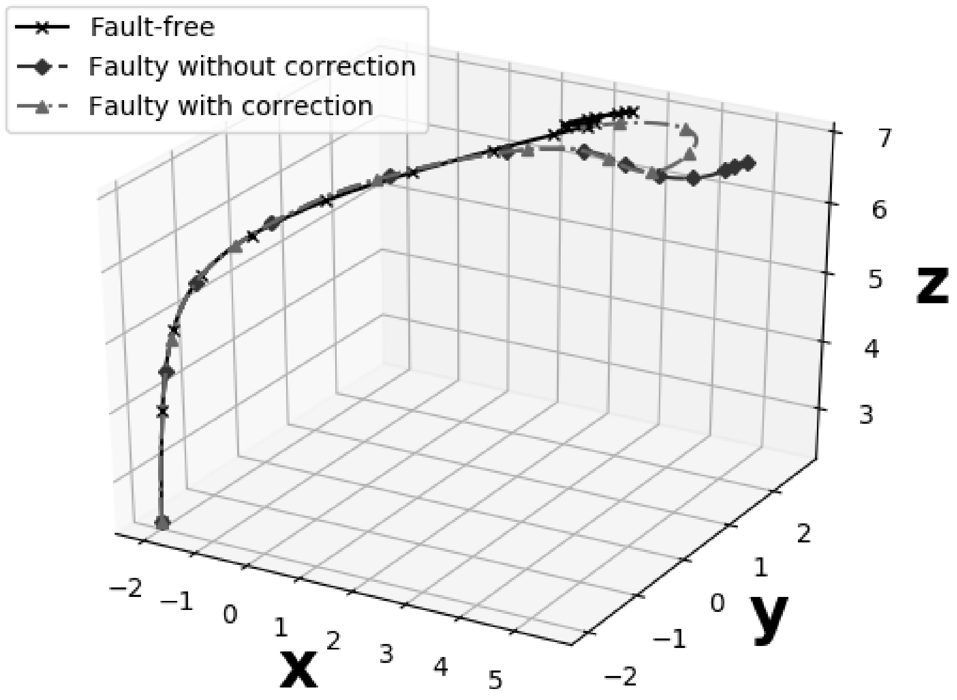


Figure 5.45: Error correction of actuator parametric failure

For the SbW case, the steer by wire subsystem relies on a three-phase induction motor to drive the steering axle to a required steering angle, which is in turn fed by an automotive alternator [189]. Fault was introduced in the co-efficient of wheel viscous friction, which happens in the real system due to repetitive use. Figure 5.47a shows the system behavior increased viscous friction. The reward per episode plot is shown in Figure 5.46 and the system behavior faulty without correction and with correction are shown in Figure 5.47. As seen from the plot, there is considerable behavior change from the system command due to this fault. However, MAB based controller reconfiguration was performed as the corrective measure and as can be observed from Figure 5.47b, the system behavior was restored when controller reconfiguration was performed.

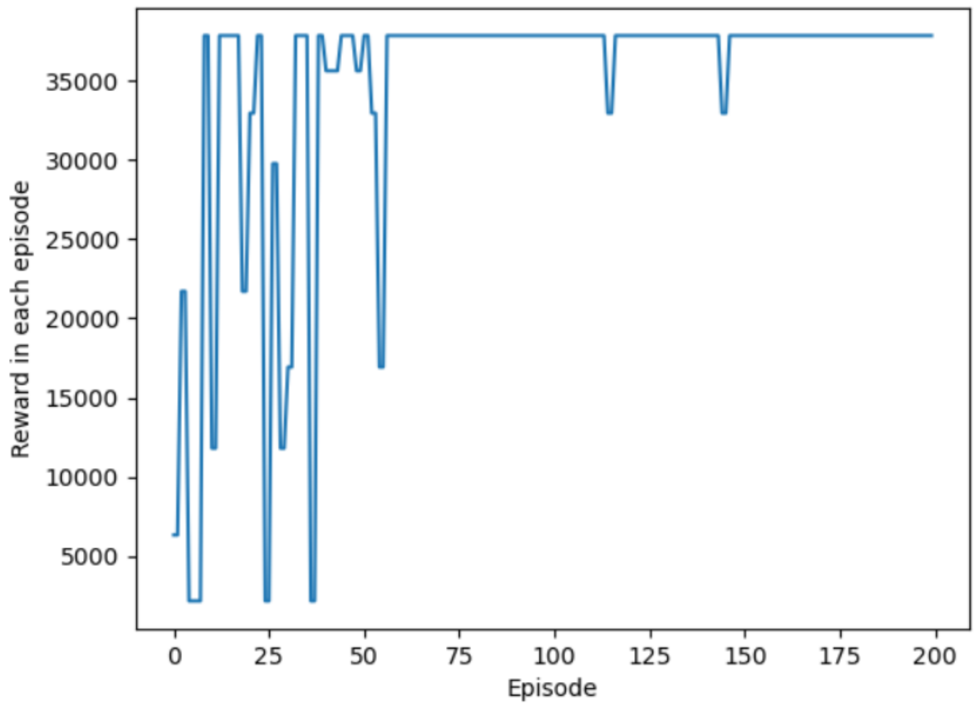
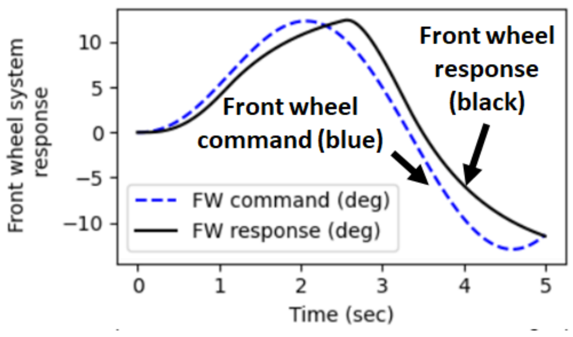
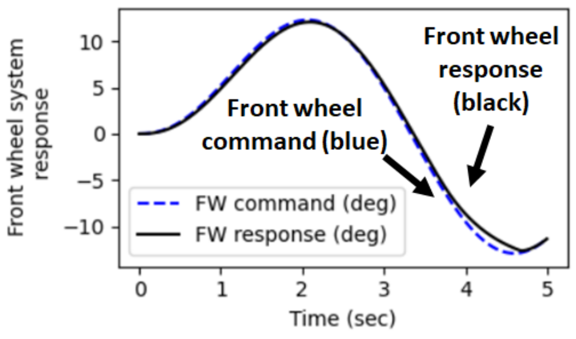


Figure 5.46: Reward per episode plot of MAB learner for SbW system

Table 5.8 shows the average number of episode required to obtain the correct controller parameter. 20 experiments were performed where the MAB learner was run for 100 experiments. As can be observed from this table, the proposed approach was rap[idly able to obtain the correct controller parameter.



(a)



(b)

Figure 5.47: (Left) Trajectory of the steer-by-wire system under actuator parametric fault and (right) corrected trajectory

Table 5.8: Average number of trial required to learn controller parameter

Quadcopter system	SbW system
14	40

Control program error

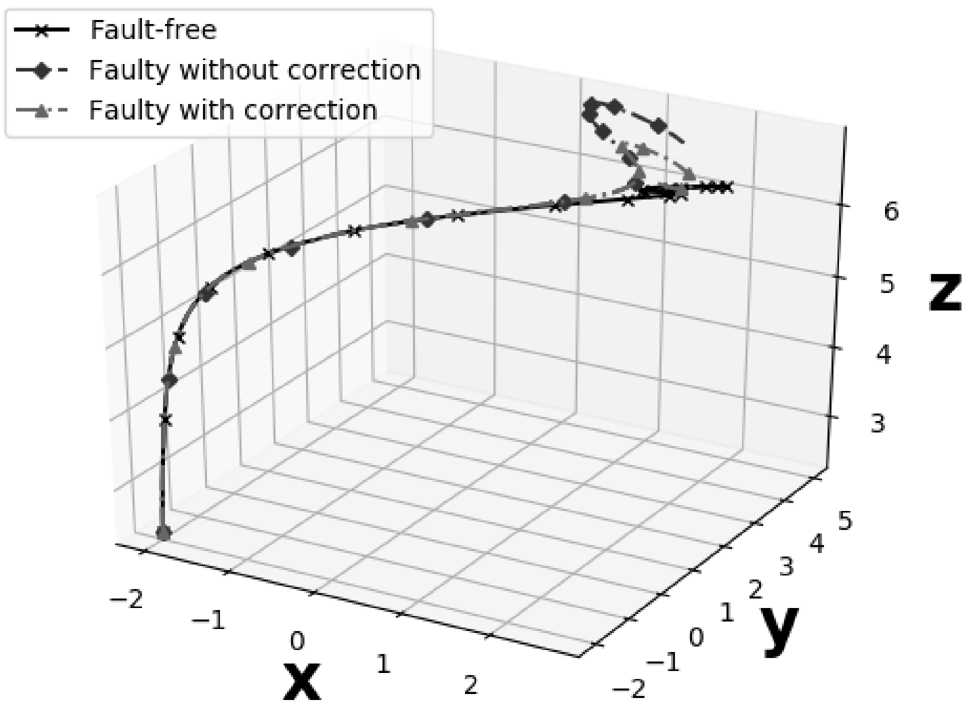


Figure 5.48: Error correction of control program failure

The control program error was introduced as a form of transient and spurious bit-flips in the data of the controller. The failure was introduced at time $t = 2$ sec when the system goes through unknown trajectory. However, the control program value was restored to its last known value and by this way system trajectory was improved. Three trajectories, namely - fault-free, faulty without correction and faulty with correction are shown in Figure 5.48 which the efficacy of the proposed approach.

Similar to quadcopter case, transient fault in the form of spurious bit-flips were introduced for SbW case. Figure 5.49a shows the system behavior in presence of failure in the data of the control program. When the error was detected and diagnosed, the faulty action was skipped and last control action was applied to the system as the corrective measure. As can be observed from Figure 5.49b, the proposed correction approach was able to restore system behavior.

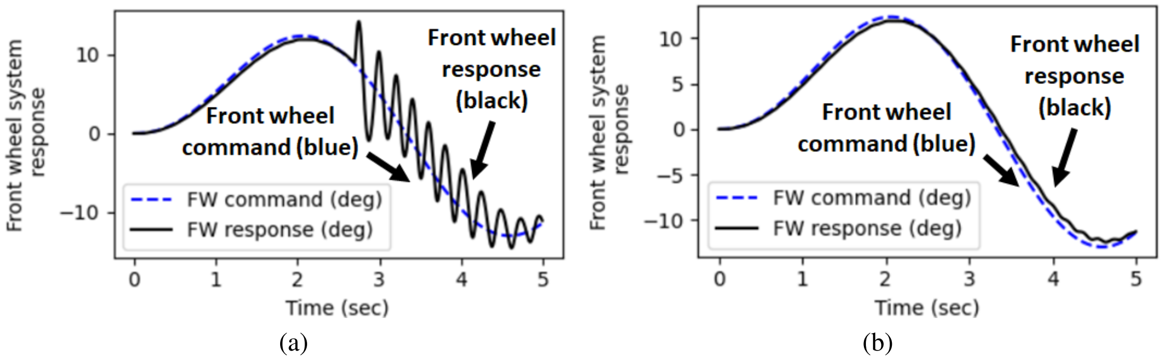


Figure 5.49: (Left) Trajectory of the steer-by-wire system under control program transient fault and (right) corrected trajectory

5.6.9 Hardware validation: Error correction

We implemented the proposed correction approach in the '*Crazyflie 2.1*' [142] quadcopter system. A data-driven model was generated with the works found in [190–193]. As shown in Table 5.7, this system has very limited hardware resource and timing cycle available, and for this reason, extensive code optimization was performed to accommodate the correction approach. In the error correction experiments, faults were injected into the sensor (gyroscope), and actuator circuit of the quadcopter and the obtained results are discussed below:

Sensor fault

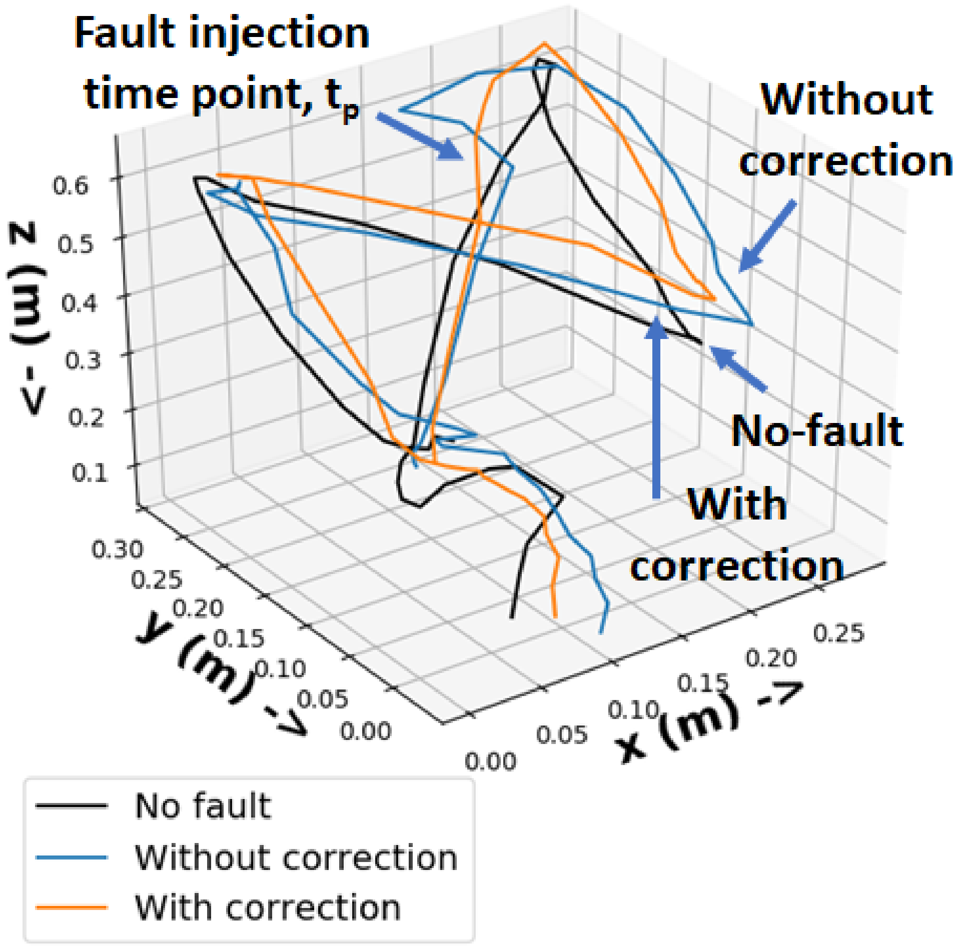


Figure 5.50: Error correction of sensor failure

We look at the failure at sensor as the first test case. The MEMS gyroscope experiences a systemic drift in time which is a common form of failure for this kind of sensor. This drift introduces incorrect reading at the gyroscope. The check error was recorded, followed by optimization and correction. Three trajectories of the quadcopter namely - at fault-free or nominal condition (black), with sensor fault when no correction is performed (blue), and finally with correction approach applied (orange) are showed in Figure 5.50. These plots show that after '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.

Actuator fault

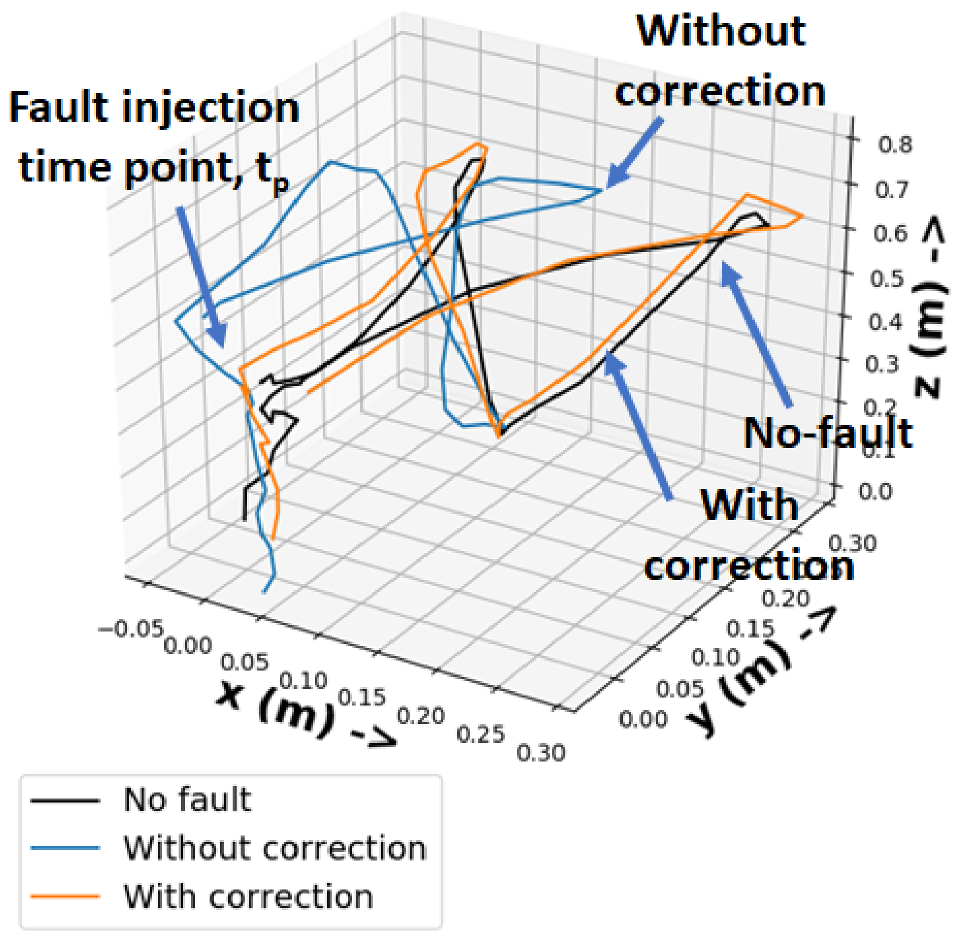


Figure 5.51: Error correction of actuator parametric failure

Epsilon-greedy MAB based control reconfiguration approach has been applied for actuator parametric correction. As stated in previous chapter, the quadcopter system has DC motors which produces necessary thrust from its input voltage, and the crazyflie system has an internal pulse-width modulator (PWM) circuit which effectively controls the generated torque of an arbitrary motor. When actuator goes through a parametric failure, it is needed to change the actuator excitation (through the PWM circuitry) to compensate for this. A previously learned MAB based learner was deployed which was trained for its nominal value. After the deployment, the torque constant was reduced to 80%, and the MAB learner reconfigured the correct control parameter, restoring the behavior. The trajectories during fault-free, faulty without correction, and faulty with correction are shown in Figure 5.51.

5.7 Summary

An encoded Extended Kalman Filter based internal anomaly detection, and correction framework has been presented in this research. The proposed approach has been successfully implemented for detection and correction in two different test cases: (a) a quadcopter and (b) a steer-by-wire system. A detailed comparative study has been performed and it is found that the implementation overhead is very low compared to earlier machine learning based checking schemes for nonlinear systems. At the same time, higher coverage of errors and failures is achieved. Simulation data and hardware validation experiments obtained for different failure mechanisms in different subsystems corroborate the efficacy of the proposed technique. Future work will focus on extending this approach to other pervasive failure mechanisms and external security attacks.

CHAPTER 6
CONCLUSION AND FUTURE WORK

In this dissertation, a framework for real-time detection and correction of errors in linear and nonlinear systems is presented. The state-space encoding based checking methodology efficiently detects system errors due to component malfunctions in control systems as well as security attacks in cyber-physical systems.

Chapter 2 provided the theoretical basis of error detection and correction of switched capacitor circuits for multiple failure modes. The feedback of the constructed checksum error to the erroneous state was shown to accurately compensate the effects of faults on the system performance in real-time, and the error surface of different failure modes were shown.

Chapter 3 provided the foundational theory of system level state-space encoding for detection, diagnosis and correction of parametric failures for multiple linear autonomous systems. The theory of error detection, and diagnosis is developed, designed and implemented. It is also shown how a linear encoding can be used for error correction with the help of control reconfiguration. Simulation results on a linear system are presented to illustrate the detection and correction capabilities.

Chapter 4 demonstrated how the state encoding based error checking infrastructure can be extended to nonlinear state variable system. A properly learned machine learning model has been employed in hierarchical manner to diagnose the health of a nonlinear autonomous system decomposing the system down to individual subsystems. Error was detected and corrected in each of these subsystems.

Chapter 5 presented computationally inexpensive methods to detect and correct failures for nonlinear autonomous systems where the encoded checking scheme dynamically adapt to the operating condition of the system by adjusting its threshold. The proposed approach was been applied to quadcopter and ‘Steer-by-wire’ automotive system.

6.1 Future Work

The future endeavors from this research can be summarized as:

1. Pervasive deployment of autonomous vehicles will depend on their safety and trustworthiness. Risks to such vehicles from hazardous operation of all vehicles on the road need to be detected and diagnosed with low latency and high coverage across dynamic traffic conditions. A low-cost, dynamically adaptive schemed is needed to mitigate the anomalies for modern autonomous system on-the-fly. More research is required to address the trade-off among performance, robustness, and hardware and memory overhead of the control of modern autonomous vehicles.

2. The resilience of the cyber-physical system to external security attacks need to be extended to address malicious attacks on different systems. Developing a comprehensive security framework also need to consider malware based attacks on the control software present in the system and propose low cost algorithm based checking and error recovery. Recent work [175] has started focusing on real-time monitoring of control algorithms to check for any inconsistency induced by malware attacks or soft errors.

3. Safety is also an important factor in the control of autonomous systems. Numerous researches are addressing the need for performance and robustness for autonomous systems; however, very few research considers safety in this analysis. This is an important research direction.

4. Health of communication channel/medium which is responsible (vehicle to vehicle) V2V or (vehicle to infrastructure) V2X communication is an important component to ensure sound performance of fleets of autonomous vehicles/systems. A dynamically

changing infrastructure is need to spatio-temporally tune into appropriate direction in presence of system traffic is needed. This is an interesting research direction which recent work has started to look at [194].

REFERENCES

[1] K. Krewell. “The slowing of Moore’s law and its impact”. In: *https://goo.gl/XVNbF1* (July 30, 2015).

[2] L. Eeckhout. “Is Moore’s Law Slowing Down? What’s Next?” In: *IEEE Micro* 37.4 (2017), pp. 4–5.

[3] Economist Quarterly. “After Moore’s Law”. In: *https://goo.gl/RpZZaq* (March 12, 2016).

[4] A. Oreskovic. “Intel admits the engine behind the tech industry’s amazing 50-year run is slowing down”. In: *https://goo.gl/TTwm5E* (March 23, 2016).

[5] T. Simonite. “Intel puts the brakes on Moore’s law”. In: *https://goo.gl/rpjBbP* (June 23, 2014).

[6] R. McMillan. “Is the end of Moore’s law slowing the world’s supercomputing race?” In: *https://goo.gl/bgUsWj* (June 23, 2014).

[7] P. Bright. “Moore’s law really is dead this time”. In: *https://goo.gl/YJEJbS* (February 2, 2016).

[8] J. Stokes. “Transistors go 3D as Intel re-invents the microchip”. In: *https://goo.gl/JL8U18* (May 4, 2011).

[9] S. Borkar et al. “Parameter variations and impact on circuits and microarchitecture”. In: *Proceedings 2003. Design Automation Conference*. June 2003, pp. 338–342.

[10] W.S. Levine, T.L. Johnson, and M. Athans. “Optimal limited state variable feedback controllers for linear systems”. In: *Automatic Control, IEEE Transactions on* 16.6 (1971), pp. 785–793.

[11] C.J. Wenk and C. Knapp. “Parameter optimization in linear systems with arbitrarily constrained controller structure”. In: *Automatic Control, IEEE Transactions on* 25.3 (1980), pp. 496–500.

[12] T. Soderstrom. “On some algorithms for design of optimal constrained regulators”. In: *Automatic Control, IEEE Transactions on* 23.6 (1978), pp. 1100–1101.

[13] V. Gupta, B. Hassibi, and R. M. Murray. “On the synthesis of control laws for a network of autonomous agents”. In: *Proceedings of the 2004 American Control Conference*. Vol. 6. June 2004, 4927–4932 vol.6.

[14] V. Gupta, B. Hassibi, and R. M. Murray. “A sub-optimal algorithm to synthesize control laws for a network of dynamic agents”. In: *International Journal of Control* 78.16 (2005), pp. 1302–1313.

[15] G. A. de Castro. “Convex methods for the design of structured controllers”. PhD thesis. University of California, Los Angeles, 2001.

[16] C. Langbort, R.S. Chandra, and R. D’Andrea. “Distributed control design for systems interconnected over an arbitrary graph”. In: *Automatic Control, IEEE Transactions on* 49.9 (2004), pp. 1502–1519.

[17] C. Langbort, V. Gupta, and R. M. Murray. “Distributed control over failing channels”. In: *Networked embedded sensing and control*. Springer, 2006, pp. 325–342.

[18] Huibert Kwakernaak. *Linear Optimal Control Systems*. Ed. by Raphael Sivan. New York, NY, USA: John Wiley & Sons, Inc., 1972. ISBN: 0471511102.

[19] Mark Harris. *The 2,578 Problems With Self-Driving Cars*. Feb. 2017.

[20] *Sensors Linked to Boeing 737 Crashes Vulnerable to Failure*. Apr. 2019.

[21] *Boeing’s Crashes Expose Reliance on Sensors Vulnerable to Damage*. Apr. 2019.

[22] *Report: Uber’s Self-Driving Car Sensors Ignored Cyclist In Fatal Accident*. May 2018.

[23] John Von Neumann. “Probabilistic logics and the synthesis of reliable organisms from unreliable components”. In: *Automata studies* 34 (1956), pp. 43–98.

[24] N. J. A. Sloane and Aaron D. Wyner. “Reliable Circuits Using Less Reliable Relays”. In: *Claude E. Shannon:Collected Papers*. Wiley-IEEE Press, 1993, pp. 796–813.

[25] Kuang-Hua Huang and J. A. Abraham. “Algorithm-Based Fault Tolerance for Matrix Operations”. In: *IEEE Transactions on Computers* C-33.6 (June 1984), pp. 518–528.

[26] J.-Y. Jou and J.A. Abraham. “Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures”. In: *Proceedings of the IEEE* 74.5 (May 1986), pp. 732–741.

[27] J.-Y. Jou and J.A. Abraham. “Fault-tolerant FFT networks”. In: *Computers, IEEE Transactions on* 37.5 (May 1988), pp. 548–561.

[28] Rajamohana Hegde and Naresh R. Shanbhag. “Soft Digital Signal Processing”. In: *IEEE Trans. Very Large Scale Integr. Syst.* 9.6 (Dec. 2001), pp. 813–823.

[29] N. Shanbhag. “Reliable and energy-efficient digital signal processing”. In: *Design Automation Conference, 2002. Proceedings. 39th*. 2002, pp. 830–835.

[30] S. Byonghyo and N.R. Shanbhag. “Energy-efficient soft error-tolerant digital signal processing”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 14.4 (Apr. 2006), pp. 336–348.

[31] S. Byonghyo and N.R. Shanbhag. “Reduced precision redundancy for low-power digital filtering”. In: *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*. Vol. 1. Nov. 2001, 148–152 vol.1.

[32] Byonghyo Shim and N. R. Shanbhag. “Energy-efficient soft error-tolerant digital signal processing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.4 (Apr. 2006), pp. 336–348.

[33] B. Shim, N. R. Shanbhag, and S. Lee. “Energy-efficient soft error-tolerant digital signal processing”. In: *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*. Vol. 2. Nov. 2003, 1493–1497 Vol.2.

[34] Rajamohana Hegde and Naresh R. Shanbhag. “Energy-efficient signal processing via algorithmic noise-tolerance”. In: *in Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*. 1999, pp. 30–35.

[35] V.S.S. Nair and J.A. Abraham. “Real-number codes for fault-tolerant matrix operations on processor arrays”. In: *Computers, IEEE Transactions on* 39.4 (Apr. 1990), pp. 426–435.

[36] A. Chatterjee and M. A. d’Abreu. “The design of fault-tolerant linear digital state variable systems: theory and techniques”. In: *IEEE Transactions on Computers* 42.7 (July 1993), pp. 794–808.

[37] M. Ashouei and A. Chatterjee. “Checksum-Based Probabilistic Transient-Error Compensation for Linear Digital Systems”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 17.10 (2009), pp. 1447–1460.

[38] M.M. Nisar and A. Chatterjee. “Guided Probabilistic Checksums for Error Control in Low-Power Digital Filters”. In: *Computers, IEEE Transactions on* 60.9 (Sept. 2011), pp. 1313–1326.

[39] A. Chatterjee. “Concurrent error detection and fault-tolerance in linear analog circuits using continuous checksums”. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 1.2 (June 1993), pp. 138–150.

[40] A Chatterjee. “Checksum-based concurrent error detection in linear analog systems with second and higher order stages”. In: *VLSI Test Symposium*. 1992, pp. 286–291.

[41] Suvadeep Banerjee, Alvaro Gomez-Pau, and Abhijit Chatterjee. “Design of low cost fault tolerant analog circuits using real-time learned error compensation”. In: *European Test Symposium*. Paderborn, Germany: IEEE Computer Society, 2014, pp. 1–2.

[42] Alvaro Gomez-Pau, Suvadeep Banerjee, and Abhijit Chatterjee. “Real-time transient error and induced noise cancellation in linear analog filters using learning-assisted adaptive analog checksums”. In: *IEEE 20th International On-Line Testing Symposium (IOLTS)*. St Feliu de Guixols, Spain: IEEE Computer Society, 2014, pp. 25–30.

[43] Gary D. Hachtel and Fabio Somensi. *Logic Synthesis and Verification Algorithms*. Springer, 2012.

[44] Douglas Perry and Harry Foster. *Applied Formal Verification*. McGraw Hill, 2005.

[45] H.-G.D. Stratigopoulos and Y. Makris. “Concurrent detection of erroneous responses in linear analog circuits”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 25.5 (May 2006), pp. 878–891.

[46] H-G. D. Stratigopoulos and Y. Makris. “An Adaptive Checker for the Fully Differential Analog Code”. In: *IEEE Journal of Solid-State Circuits* 41.6 (2006), pp. 1421–1429.

[47] A. Chatterjee, B.C. Kim, and N. Nagi. “DC Built-In Self-Test for Linear Analog Circuits”. In: *Design and Test of Computers, IEEE* 13.2 (1996), pp. 26–33.

[48] A. Chatterjee and R.K. Roy. “Concurrent error detection in nonlinear digital circuits using time-freeze linearization”. In: *Computers, IEEE Transactions on* 46.11 (Nov. 1997), pp. 1208–1218.

[49] S. Banerjee, M. I. Momtaz, and A. Chatterjee. “Concurrent error detection in nonlinear digital filters using checksum linearization and residue prediction”. In: *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*. July 2015, pp. 53–58.

[50] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[51] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[52] Dong Wei and Kun Ji. “Resilient industrial control system (RICS): Concepts, formulation, metrics, and insights”. In: *Resilient Control Systems (ISRCS), 2010 3rd International Symposium on*. IEEE. 2010, pp. 15–22.

[53] Christoffer Sloth, Thomas Esbensen, and Jakob Stoustrup. “Robust and fault-tolerant linear parameter-varying control of wind turbines”. In: *Mechatronics* 21.4 (2011), pp. 645–659.

[54] Quanyan Zhu and Tamer Başar. “Robust and resilient control design for cyber-physical systems with an application to power systems”. In: *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE. 2011, pp. 4066–4071.

[55] Peter F Odgaard, Jakob Stoustrup, and Michel Kinnaert. “Fault-tolerant control of wind turbines: A benchmark model”. In: *Control Systems Technology, IEEE Transactions on* 21.4 (2013), pp. 1168–1182.

[56] Shen Yin, Guang Wang, and Hamid Reza Karimi. “Data-driven design of robust fault detection system for wind turbines”. In: *Mechatronics* 24.4 (2013), pp. 298–306.

[57] Hailiang Sun, Yanyang Zi, and Zhengjia He. “Wind turbine fault detection using multiwavelet denoising with the data-driven block threshold”. In: *Applied Acoustics* 77 (2014), pp. 122–129.

[58] Shen Yin, Hao Luo, and Steven X Ding. “Real-time implementation of fault-tolerant control systems with performance optimization”. In: *Industrial Electronics, IEEE Transactions on* 61.5 (2014), pp. 2402–2411.

[59] Marcos Orchard, George Vachtsevanos, and Kai Goebel. “A Combined Model-Based and Data-Driven Prognostic Approach for Aircraft System Life Management”. In: *Machine Learning and Knowledge Discovery for Engineering Systems Health Management* (2011), p. 363.

[60] Bin Zhang et al. “A probabilistic fault detection approach: Application to bearing fault detection”. In: *IEEE Transactions on Industrial Electronics* 58.5 (2010), pp. 2011–2018.

[61] Lennon R Cork, Rodney A Walker, and Shane Dunn. “Fault detection, identification and accommodation techniques for unmanned airborne vehicles”. In: Australian International Aerospace Congress (AIAC). 2005.

[62] Balaje T Thumati and Sarangapani Jagannathan. “A model-based fault-detection and prediction scheme for nonlinear multivariable discrete-time systems with asymp-

totic stability guarantees". In: *IEEE Transactions on Neural Networks* 21.3 (2010), pp. 404–423.

[63] Evangelia Lampiri. "Sensor anomaly detection and recovery in a nonlinear autonomous ground vehicle model". In: *2017 11th Asian Control Conference (ASCC)*. IEEE. 2017, pp. 430–435.

[64] Kamel Bouibed, Abdel Aitouche, and Mireille Bayart. "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. 2009, pp. 621–626.

[65] Puneet Goel et al. "Fault detection and identification in a mobile robot using multiple model estimation and neural network". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 3. IEEE. 2000, pp. 2302–2309.

[66] Lennon Cork and Rodney Walker. "Sensor fault detection for UAVs using a nonlinear dynamic model and the IMM-UKF algorithm". In: *2007 Information, Decision and Control*. IEEE. 2007, pp. 230–235.

[67] Douglas Brown et al. "Particle filter based anomaly detection for aircraft actuator systems". In: *2009 IEEE Aerospace conference*. IEEE. 2009, pp. 1–13.

[68] Marcos Eduardo Orchard. "A particle filtering-based framework for on-line fault diagnosis and failure prognosis". PhD thesis. Georgia Institute of Technology, 2007.

[69] Marcos E Orchard and George J Vachtsevanos. "A particle-filtering approach for on-line fault diagnosis and failure prognosis". In: *Transactions of the Institute of Measurement and Control* 31.3-4 (2009), pp. 221–246.

[70] M Orchard, F Tobar, and G Vachtsevanos. "Outer feedback correction loops in particle filtering-based prognostic algorithms: Statistical performance comparison". In: *Studies in Informatics and Control* 18.4 (2009), pp. 295–304.

[71] George J Vachtsevanos and George J Vachtsevanos. *Intelligent fault diagnosis and prognosis for engineering systems*. Vol. 456. Wiley Hoboken, 2006.

[72] M Orchard et al. "Risk-sensitive particle-filtering-based prognosis framework for estimation of remaining useful life in energy storage devices". In: *Studies in Informatics and Control* 19.3 (2010), pp. 209–218.

[73] Pinyao Guo et al. "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. 2018, pp. 574–585.

[74] Hong Zhang, W Steven Gray, and Oscar R Gonzalez. “Performance analysis of digital flight control systems with rollback error recovery subject to simulated neutron-induced upsets”. In: *IEEE Transactions on Control Systems Technology* 16.1 (2007), pp. 46–59.

[75] Marcello R Napolitano, Younghwan An, and Brad A Seanor. “A fault tolerant flight control system for sensor and actuator failures using neural networks”. In: *Aircraft Design* 3.2 (2000), pp. 103–128.

[76] A Alessandri, M Baglietto, and Thomas Parisini. “Robust model-based fault diagnosis using neural nonlinear estimators”. In: *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*. Vol. 1. IEEE. 1998, pp. 72–77.

[77] Marcello R Napolitano et al. “Sensor validation using hardware-based on-line learning neural networks”. In: *IEEE transactions on aerospace and electronic systems* 34.2 (1998), pp. 456–468.

[78] M Borairi and H Wang. “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)*. Vol. 1. IEEE. 1998, pp. 278–282.

[79] M. I. Momtaz, S. Banerjee, and A. Chatterjee. “On-line diagnosis and compensation for parametric failures in linear state variable circuits and systems using time-domain checksum observers”. In: *2017 IEEE 35th VLSI Test Symposium (VTS)*. Apr. 2017, pp. 1–6.

[80] M. I. Momtaz et al. “Cross-Layer Control Adaptation for Autonomous System Resilience”. In: *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*. July 2018, pp. 261–264.

[81] M. I. Momtaz and A. Chatterjee. “Hierarchical Check Based Detection and Diagnosisof Sensor-Actuator Malfunction in AutonomousSystems: A Quadcopter Study”. In: *25th IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*. July 2019, pp. 316–321.

[82] Suvadeep Banerjee et al. “Real-Time Error Detection in Nonlinear Control Systems Using Machine Learning Assisted State-Space Encoding”. In: *IEEE Transactions on Dependable and Secure Computing* (2019).

[83] S. Banerjee, A. Chatterjee, and J. A. Abraham. “Efficient cross-layer concurrent error detection in nonlinear control systems using mapped predictive check states”. In: *2016 IEEE International Test Conference (ITC)*. Nov. 2016, pp. 1–10.

[84] M. I. Momtaz and A. Chatterjee. “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)*. Dec. 2019, pp. 141–146.

[85] M. I. Momtaz and A. Chatterjee. “Diagnosis and Compensation of Control Program, Sensor and Actuator Failures in Nonlinear Systems Using Hierarchical State Space Checks”. In: *Journal of Electronic Testing* (2020), pp. 683–701.

[86] Manzar Abbas and George J Vachtsevanos. “A hierarchical framework for fault propagation analysis in complex systems”. In: *2009 IEEE AUTOTESTCON*. IEEE. 2009, pp. 353–358.

[87] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[88] Naira Hovakimyan and Chengyu Cao. *L1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. SIAM, 2010.

[89] Petros A Ioannou and Jing Sun. *Robust adaptive control*. Courier Corporation, 2012.

[90] Chengyu Cao et al. “Stabilization of cascaded systems via L1 adaptive controller with application to a UAV path following problem and flight test results”. In: *2007 American Control Conference*. IEEE. 2007, pp. 1787–1792.

[91] I Ge Jin et al. “Risk-aware motion planning for automated vehicle among human-driven cars”. In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 3987–3993.

[92] Jose E Naranjo et al. “Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver”. In: *IEEE Transactions on Intelligent Transportation Systems* 9.3 (2008), pp. 438–450.

[93] Jin Cui et al. “A review on safety failures, security attacks, and available countermeasures for autonomous vehicles”. In: *Ad Hoc Networks* 90 (2019), p. 101823.

[94] M. I. Momtaz, S. Banerjee, and A. Chatterjee. “Real-time DC motor error detection and control compensation using linear checksums”. In: *2016 IEEE 34th VLSI Test Symposium (VTS)*. Apr. 2016, pp. 1–6.

[95] S. Banerjee and A. Chatterjee. “Real-time self-learning for control law adaptation in nonlinear systems using encoded check states”. In: *2017 22nd IEEE European Test Symposium (ETS)*. May 2017, pp. 1–6.

[96] Bardienus P Duisterhof et al. “Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller”. In: *arXiv preprint arXiv:1909.11236* (2019).

[97] Felipe Codevilla et al. “End-to-end driving via conditional imitation learning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4693–4700.

[98] Yunpeng Pan et al. “Agile autonomous driving using end-to-end deep imitation learning”. In: *arXiv preprint arXiv:1709.07174* (2017).

[99] Ahmed Hussein et al. “Imitation learning: A survey of learning methods”. In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–35.

[100] Liting Sun et al. “A fast integrated planning and control framework for autonomous driving via imitation learning”. In: *Dynamic Systems and Control Conference*. Vol. 51913. American Society of Mechanical Engineers. 2018, V003T37A012.

[101] K Goebel and G Vachtsevanos. “Algorithms and their impact on integrated vehicle health management”. In: *Integrated Vehicle Health Management, Perspectives on an Emerging Field*. SAE International, 2011, pp. 67–76.

[102] George Vachtsevanos et al. “Resilient design and operation of cyber physical systems with emphasis on unmanned autonomous systems”. In: *Journal of Intelligent & Robotic Systems* 91.1 (2018), pp. 59–83.

[103] George Vachtsevanos, George Georgoulas, and George Nikolakopoulos. “Fault diagnosis, failure prognosis and fault tolerant control of aerospace/unmanned aerial systems”. In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. IEEE. 2016, pp. 366–371.

[104] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “On a formal model of safe and scalable self-driving cars”. In: *arXiv preprint arXiv:1708.06374* (2017).

[105] Guotao Xie et al. “Situational assessments based on uncertainty-risk awareness in complex traffic scenarios”. In: *Sustainability* 9.9 (2017), p. 1582.

[106] Jongsang Suh, Heungseok Chae, and Kyongsu Yi. “Stochastic model-predictive control for lane change decision of automated driving vehicles”. In: *IEEE Transactions on Vehicular Technology* 67.6 (2018), pp. 4771–4782.

[107] Stefano Di Cairano et al. “Stochastic MPC with learning for driver-predictive vehicle control and its application to HEV energy management”. In: *IEEE Transactions on Control Systems Technology* 22.3 (2013), pp. 1018–1031.

[108] Rahul Bhadani et al. "Real-time distance estimation and filtering of vehicle headways for smoothing of traffic waves". In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. 2019, pp. 280–290.

[109] S. Banerjee, A. Chatterjee, and J. A Abraham. "Checksum based error detection in linearized representations of non linear control systems". In: *2016 17th Latin-American Test Symposium (LATS)*. IEEE. 2016, pp. 182–182.

[110] M. I. Momtaz, S. Banerjee, and A. Chatterjee. "Probabilistic error detection and correction in switched capacitor circuits using checksum codes". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 271–276.

[111] Maryam Ashouei and Abhijit Chatterjee. "Checksum-based Probabilistic Transient-error Compensation for Linear Digital Systems". In: *IEEE Trans. Very Large Scale Integr. Syst.* 17.10 (Oct. 2009), pp. 1447–1460.

[112] Muhammad M. Nisar and Abhijit Chatterjee. "Guided Probabilistic Checksums for Error Control in Low-Power Digital Filters". In: *IEEE Trans. Comput.* 60.9 (Sept. 2011), pp. 1313–1326.

[113] Alvaro Gomez-Pau, Suvadeep Banerjee, and Abhijit Chatterjee. "Real-time transient error and induced noise cancellation in linear analog filters using learning-assisted adaptive analog checksums". In: *IEEE 20th International On-Line Testing Symposium (IOLTS)*. St Feliu de Guixols, Spain: IEEE Computer Society, 2014, pp. 25–30.

[114] D. J. Allstot. "Prof. Temes and Switched-Capacitor Circuits?Thirty-Five Years and Counting". In: *IEEE Solid-State Circuits Magazine* 5.2 (Spring 2013), pp. 10–10.

[115] *Biquad Active Filter*. Accessed Mar. 7, 2017.

[116] Phillip E. Allen and Douglas R. Holberg. *CMOS Analog Circuit Design*. Oxford University Press, 2011.

[117] O. Moseler and R. Isermann. "Application of model-based fault detection to a brushless DC motor". In: *IEEE Transactions on Industrial Electronics* 47.5 (2000), pp. 1015–1020.

[118] "Process fault detection based on modeling and estimation methods—A survey". In: *Automatica* 20.4 (1984), pp. 387–404.

[119] Xiang-Qun Liu et al. "Fault detection and diagnosis of permanent-magnet DC motor based on parameter estimation and neural network". In: *IEEE Transactions on Industrial Electronics* 47.5 (2000), pp. 1021–1030.

[120] “Fuzzy logic-based decision-making for fault diagnosis in a {DC} motor”. In: *Engineering Applications of Artificial Intelligence* 18.4 (2005), pp. 423–450.

[121] M. Hajiaghajani, H. A. Toliyat, and I. M. S. Panahi. “Advanced fault diagnosis of a DC motor”. In: *IEEE Transactions on Energy Conversion* 19.1 (2004), pp. 60–65.

[122] *Distributed control systems and introduction to distributed control*. 2009.

[123] V. Gupta, B. Hassibi, and R.M. Murray. “On the synthesis of control laws for a network of autonomous agents”. In: *American Control Conference, 2004. Proceedings of the 2004*. Vol. 6. 2004, 4927–4932 vol.6.

[124] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[125] Kenneth Levenberg. “A method for the solution of certain non-linear problems in least squares”. In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164–168.

[126] Donald W Marquardt. “An algorithm for least-squares estimation of nonlinear parameters”. In: *Journal of the society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441.

[127] International Organization for Standardization. “ISO 26262: Road Vehicles : Functional Safety”. In: (2011).

[128] Remus C Avram. “Fault diagnosis and fault-tolerant control of quadrotor uavs”. In: (2016).

[129] Juan-Pablo Afman et al. “Towards a New Paradigm of UAV Safety”. In: *arXiv preprint arXiv:1803.09026* (2018).

[130] Ding-Li Yu, T. K. Chang, and Ding-Wen Yu. “Fault tolerant control of multivariable processes using auto-tuning PID controller”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.1 (Feb. 2005), pp. 32–43.

[131] Michael Margaliot and Gideon Langholz. “Fuzzy Lyapunov-based approach to the design of fuzzy controllers”. In: *Fuzzy sets and systems* 106.1 (1999), pp. 49–59.

[132] Dale E Seborg et al. *Process dynamics and control*. John Wiley & Sons, 2010.

[133] Juan R Pimentel. *An architecture for a safety-critical steer-by-wire system*. Tech. rep. SAE Technical Paper, 2004.

[134] Agoston Restas et al. “Drone applications for supporting disaster management”. In: *World Journal of Engineering and Technology* 3.03 (2015), p. 316.

[135] Thomas Birtchnell and Chris Gibson. “Less talk more drone: Social research with UAVs”. In: *Journal of Geography in Higher Education* 39.1 (2015), pp. 182–189.

[136] Francisco Klauser and Silvana Pedrozo. “Power and space in the drone age: a literature review and politico-geographical research agenda”. In: *Geographica Helvetica* 70.4 (2015), pp. 285–293.

[137] Bilal Taha and Abdulhadi Shoufan. “Machine learning-based drone detection and classification: State-of-the-art in research”. In: *IEEE Access* 7 (2019), pp. 138669–138682.

[138] S Radiansyah, MD Kusrini, and LB Prasetyo. “Quadcopter applications for wildlife monitoring”. In: *IOP Conference Series: Earth and environmental science*. Vol. 54. 1. IOP Publishing. 2017, p. 012066.

[139] Disha Amrutlal Gandhi and Munmun Ghosal. “Novel low cost quadcopter for surveillance application”. In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE. 2018, pp. 412–414.

[140] S Ahirwar et al. “Application of drone in agriculture”. In: *International Journal of Current Microbiology and Applied Sciences* 8.01 (2019), pp. 2500–2505.

[141] F Sabatino. “Quadrotor control: modeling, nonlinear control design, and simulation”. MA thesis. KTH Royal Institute of Technology, 2015.

[142] Bitcraze. *Crazyflie 2.1*. 2020.

[143] Gabriel Staples. *Propeller Static Dynamic Thrust Calculation*. Jan. 1970.

[144] *Drone Motors: Choose the Best Motors for Your Quadcopter*.

[145] MURUGANANTHAM N and Palani Subbiah. “State space modeling and simulation of sensorless permanent magnet BLDC motor”. In: vol. 2. Oct. 2010.

[146] Randal W. Beard and Timothy W. McLain. *Small unmanned aircraft: theory and practice*. Princeton University Press, 2012.

[147] Muhammad Tanveer et al. “Stabilized controller design for attitude and altitude controlling of quad-rotor under disturbance and noisy conditions”. In: *American Journal of Applied Sciences* 10 (Aug. 2013), pp. 819–831.

[148] Hava T. Siegelmann and Eduardo D. Sontag. “Turing computability with neural nets”. In: *Applied Mathematics Letters* 4.6 (1991), pp. 77–80.

[149] *A Neural Network for Machine Translation, at Production Scale*. Sept. 2016.

[150] T. Mikolov et al. "Extensions of recurrent neural network language model". In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2011, pp. 5528–5531.

[151] Ilya Sutskever, James Martens, and Geoffrey Hinton. "Generating Text with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 1017–1024. ISBN: 978-1-4503-0619-5.

[152] T. Mikolov et al. "Recurrent Neural Network Based Language Model". In: *INTERSPEECH*. 2010, pp. 1045–1048.

[153] Deger Ayata, Murat Saraclar, and Arzucan Ozgur. "BUSEM at SemEval-2017 Task 4A Sentiment Analysis with Word Embedding and Long Short Term Memory RNN Approaches". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 777–783.

[154] Quanzeng You et al. "Image Captioning With Semantic Attention". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[155] Junhua Mao et al. "Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)". In: *arXiv e-prints*, arXiv:1412.6632 (Dec. 2014), arXiv:1412.6632. arXiv: 1412.6632 [cs.CV].

[156] Y. Miao, M. Gowayyed, and F. Metze. "EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. Dec. 2015, pp. 167–174.

[157] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80.

[158] Barry W. Johnson, ed. *Design &Amp; Analysis of Fault Tolerant Digital Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1988. ISBN: 0-201-07570-9.

[159] J. Kiefer. "Sequential Minimax Search for a Maximum". In: *Proceedings of the American Mathematical Society* 4.3 (1953), p. 502.

[160] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[161] Yunus Santur. “Sentiment analysis based on gated recurrent unit”. In: *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE. 2019, pp. 1–5.

[162] Guizhu Shen et al. “Deep learning with gated recurrent unit networks for financial sequence predictions”. In: *Procedia computer science* 131 (2018), pp. 895–903.

[163] Dang Lien Minh et al. “Deep learning approach for short-term stock trends prediction based on two-stream gated recurrent unit network”. In: *Ieee Access* 6 (2018), pp. 55392–55404.

[164] Jihyun Kim, Howon Kim, et al. “Classification performance using gated recurrent unit recurrent neural network on energy disaggregation”. In: *2016 international conference on machine learning and cybernetics (ICMLC)*. Vol. 1. IEEE. 2016, pp. 105–110.

[165] Wenchuan Yang, Wen Zuo, and Baojiang Cui. “Detecting malicious urls via a keyword-based convolutional gated-recurrent-unit neural network”. In: *IEEE Access* 7 (2019), pp. 29891–29900.

[166] Simon Andermatt, Simon Pezold, and Philippe Cattin. “Multi-dimensional gated recurrent units for the segmentation of biomedical 3D-data”. In: *Deep learning and data labeling for medical applications*. Springer, 2016, pp. 142–151.

[167] Yaodong Tang et al. “Question detection from acoustic features using recurrent neural network with gated recurrent unit”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 6125–6129.

[168] Kanwaldeep Kaur and Giselle Rampersad. “Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars”. In: *Journal of Engineering and Technology Management* 48 (2018), pp. 87–96.

[169] Azim Eskandarian. *Handbook of intelligent vehicles*. Springer, 2012.

[170] C. N. Amarnath, M. I. Momtaz, and A. Chatterjee. “Encoded Check Driven Concurrent Error Detection in Particle Filters for Nonlinear State Estimation”. In: *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2020, pp. 1–6.

[171] R.E. Kalman. *Contributions to the Theory of Optimal Control*. 1960.

[172] Simon Haykin. *Kalman filtering and neural networks*. Vol. 47. John Wiley & Sons, 2004.

[173] Max A Woodbury. *Inverting modified matrices*. Statistical Research Group, 1950.

[174] E. L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses*. Third. Springer Texts in Statistics. New York: Springer, 2005. ISBN: 0-387-98864-5.

[175] Sujay Pandey, Suvadeep Banerjee, and Abhijit Chatterjee. "Concurrent error detection and tolerance in kalman filters using encoded state and statistical covariance checks". In: *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2016, pp. 161–166.

[176] Michael N Katehakis and Arthur F Veinott Jr. "The multi-armed bandit problem: decomposition and computation". In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268.

[177] H. Wang et al. "Sliding Mode Control for Steer-by-Wire Systems With AC Motors in Road Vehicles". In: *IEEE Transactions on Industrial Electronics* 61.3 (Mar. 2014), pp. 1596–1611.

[178] Juan R Pimentel. *An architecture for a safety-critical steer-by-wire system*. Tech. rep. SAE Technical Paper, 2004.

[179] Salem Haggag et al. "Modeling, control, and validation of an electro-hydraulic steer-by-wire system for articulated vehicle applications". In: *IEEE/AsME Transactions on Mechatronics* 10.6 (2005), pp. 688–692.

[180] Omae Manabu et al. "The application of RTK-GPS and steer-by-wire technology to the automatic driving of vehicles and an evaluation of driver behavior". In: *IATSS research* 30.2 (2006), pp. 29–38.

[181] Sheikh Muhamad Hafiz Fahami et al. "Modeling and simulation of vehicle steer by wire system". In: *2012 IEEE Symposium on Humanities, Science and Engineering Research*. IEEE. 2012, pp. 765–770.

[182] Chao Huang et al. "Fault tolerant steer-by-wire systems: An overview". In: *Annual Reviews in Control* 47 (2019), pp. 98–111.

[183] William H Levison et al. *Development of a Driver Vehicle Module (DVM) for the Interactive Highway Safety Design Model (IHSDM)*. Tech. rep. United States. Federal Highway Administration. Office of Research, Development, and Technology, 2007.

[184] Erik Frisk. "Residual generation for fault diagnosis". PhD thesis. Linköpings universitet, 2001.

[185] M. I. Momtaz, C. N. Amarnath, and A. Chatterjee. "Concurrent Error Detection in Embedded Digital Control of Nonlinear Autonomous Systems Using Adaptive

State Space Checks”. In: *2020 IEEE International Test Conference (ITC)*. 2020, pp. 1–10.

[186] D Marano et al. “Modeling of a three-axes MEMS gyroscope with feedforward PI quadrature compensation”. In: *Advances on Mechanics, Design Engineering and Manufacturing*. Springer, 2017, pp. 71–80.

[187] Seong Tak Woo et al. “Angle sensor module for vehicle steering device based on multi-track impulse ring”. In: *Sensors* 19.3 (2019), p. 526.

[188] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[189] Chao Huang and Liang Li. “Architectural design and analysis of a steer-by-wire system in view of functional safety concept”. In: *Reliability Engineering & System Safety* 198 (2020), p. 106822.

[190] Carlos Luis and Jérôme Le Ny. “Design of a trajectory tracking controller for a nanoquadcopter”. In: *arXiv preprint arXiv:1608.05786* (2016).

[191] William Hanna. “Modelling and control of an unmanned aerial vehicle”. In: (2014).

[192] Giri Prashanth Subramanian. “Nonlinear control strategies for quadrotors and Cube-Sats”. In: (2015).

[193] Julian Förster. “System identification of the crazyflie 2.0 nano quadrocopter”. B.S. thesis. ETH Zurich, 2015.

[194] S. Komarraju and A. Chatterjee. “Fast EVM Tuning of MIMO Wireless Systems Using Collaborative Parallel Testing and Implicit Reward Driven Learning”. In: *2020 IEEE International Test Conference (ITC)*. 2020, pp. 1–10.

VITA

Md Imran Momtaz received the B.Sc. degree in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2009 and the M.Sc. degree in Electrical and Computer engineering from the Georgia Institute of Technology, Atlanta, USA in 2019. His current research interests include machine learning, error resilience in signal processing and nonlinear autonomous systems and intelligent and secure adaptive real-time systems.