Fault Detection for Unmanned Aerial Vehicles with Non-Redundant Sensors

Brandon Jeffrey Cannon
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/5308

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
Fault Detection for Unmanned Aerial Vehicles

with Non-Redundant Sensors

Brandon J. Cannon

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Timothy W. McLain, Chair
Mark B. Colton
Randal W. Beard

Department of Mechanical Engineering
Brigham Young University
November 2014

Copyright © 2014 Brandon J. Cannon
All Rights Reserved
ABSTRACT

Fault Detection for Unmanned Aerial Vehicles with Non-Redundant Sensors

Brandon J. Cannon
Department of Mechanical Engineering, BYU
Master of Science

To operate, autonomous systems of necessity employ a variety of sensors to perceive their environment. Many small unmanned aerial vehicles (UAV) are unable to carry redundant sensors due to size, weight, and power (SWaP) constraints. Faults in these sensors can cause undesired behavior, including system instability. Thus, detection of faults in these non-redundant sensors is of paramount importance.

The problem of detecting sensor faults in non-redundant sensors on board autonomous aircraft is non-trivial. Factors that make development of a solution difficult include both an inability to perfectly characterize systems and sensors as well as the SWaP constraints inherent with small UAV. An additional challenge is the ability of a fault-detection method to strike a balance between false-alarm rate and detection rate.

This thesis explores two model-based methods of fault-detection for non-redundant sensors, a Kalman filter based method and a particle filter based method. The Kalman filter based method employs tests of mean and covariance on the normalized innovation sequence to detect faults, while the particle filter based method uses a function of the average particle weights.

The Kalman filter based approach was implemented in real time on board an autonomous rotorcraft using an extended Kalman Filter (EKF). Faults tested included varied levels of bias, drift, and increased noise. Metrics included false-alarm rate, detection rate, and delay to detection. The particle filter based approach was implemented on a simulated system. This was then compared with an implementation of the EKF based approach for the same system. The same fault types and metrics were also used for these tests.

The EKF based method of fault-detection performed well onboard the autonomous rotorcraft and should be generalizable to other systems for which an EKF or Kalman filter can be implemented. The theory indicates that the particle filter based algorithm should have performed better, though the simulations showed poor detection characteristics in comparison to the Kalman filter based method. Future work should be performed to explore improvements to the particle filter based method.

Keywords: fault-detection, Kalman filter, particle filter, non-redundant sensor, estimator, UAV, unmanned aircraft
ACKNOWLEDGMENTS

Many experiences in life bring great growth and strength. The writing of this thesis, with its attendant work, has been such an experience for me. I am grateful to have had it. And, because it would not have been possible without the help and support of those around me, I would like to take this opportunity to acknowledge and thank those who helped me along the way.

First, I would like to acknowledge the assistance of Air Force Research Laboratory (AFRL). They provided an interesting problem to pursue, along with the requisite funding through AFRL Contract AF103-008, a phase II SBIR. I would also like to acknowledge the assistance of the many great people at Scientific Systems Company, Inc. who provided invaluable guidance, especially Joseph Jackson and Jovan Bošković, who collaborated closely with me on this project.

I would also like to thank my fellow MAGICC Lab members for their assistance with a variety of tasks, both large and small. In particular, I owe a great deal of thanks to Robert Leishman, whose work on developing an estimator for the Hexacopter has been critical to the completion of my own work on fault detection.

I would next like to acknowledge and thank my graduate committee members, Dr. Mark Colton of the BYU Mechanical Engineering department and Dr. Randal Beard of the BYU Electrical Engineering department, both of whose teaching enlightened me and inspired me to follow the course of study that led me to this work.

Next, I would like to express my appreciation to my graduate advisor, Dr. Timothy McLain of the BYU Mechanical Engineering department for the support and guidance he has provided me as I have worked toward the completion of this work. It has been a wonderful privilege to work alongside him and to learn from him. He is not only an exemplary engineer, but a great man also. In our work together, I have come to know his character and desire to develop the kindness, patience, and other qualities he displays. I immensely appreciate
the invaluable advice and encouragement he has offered, both related to the project at hand and to life in general.

I would like also to thank my parents, who have instilled in me a love of knowledge and a desire to always do my best. They have always believed in me and encouraged me in my endeavors. I appreciate the sacrifices they have made for me and the examples they have set.

I would like to next express my profoundest gratitude to my wonderful wife, Cecilee. Her belief in me has spurred me on to greater achievements than I would have ever thought possible, and her support and encouragement have been invaluable. She makes every day brighter with her smile and her kind words, and she has been a wonderful wife and mother, showing great strength and love as she has raised our three children essentially on her own while I worked on this thesis. I owe her more than I could ever hope to repay, and yet she would never expect anything in return. Nothing I might write here could possibly convey all she has done for me nor how much I appreciate and love her. I only hope that my actions and words always let her know.

I also thank my three children, Zoe, Ainsley, and Quade, who have helped motivate me. Their hugs and kisses and laughs and giggles have been much appreciated. I love them very much.

Finally, I must thank my Father in Heaven. He has watched over and protected our family and blessed us beyond measure. Many a prayer were offered and answered in pursuit of this thesis, and without His support, none of this would have been possible.

Thank you also to all those who have helped me in any way during this process, be it directly or indirectly through your friendship. I am deeply grateful.
# Table of Contents

## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vii</td>
<td>vii</td>
</tr>
</tbody>
</table>

## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
<td>viii</td>
</tr>
</tbody>
</table>

## Chapter 1 Introduction

1.1 Motivation ........................................... 1
   1.1.1 Problem Description ............................... 1
   1.1.2 Significance ....................................... 2

1.2 Technical Challenges .................................. 3

1.3 Technical Approach .................................... 4

1.4 Contributions ....................................... 5

## Chapter 2 Experimental Setup

2.1 Scenario ............................................. 6

2.2 Hardware Description .................................. 6
   2.2.1 Vehicle ........................................... 7
   2.2.2 IMU ............................................... 8
   2.2.3 Laser Rangefinder ................................. 9
   2.2.4 RGB-D Camera .................................... 10
   2.2.5 Onboard Computer ................................ 10
   2.2.6 Groundstation .................................. 11
   2.2.7 Truth Acquisition System ......................... 11
   2.2.8 Communication ................................... 13

2.3 ROS .................................................. 13
   2.3.1 Overview ........................................ 14
   2.3.2 Implementation .................................. 15

2.4 Testing Procedure .................................... 16

## Chapter 3 EKF-Based Fault Detection

3.1 Motivation ........................................... 19

3.2 Background .......................................... 20

3.3 The Extended Kalman Filter (EKF) ..................... 20

3.4 EKF Fault Detection Method ............................ 23
   3.4.1 Test of Mean .................................... 25
   3.4.2 Test of Covariance ............................... 26
   3.4.3 Threshold Test .................................. 26

3.5 Estimator ............................................ 27
   3.5.1 State Dynamics .................................. 30
   3.5.2 Error Dynamics .................................. 32
   3.5.3 Measurement Updates ............................... 32

3.6 Experimental Results .................................. 34
   3.6.1 Experimental Setup ................................ 35
LIST OF TABLES

3.1 EKF Fault Detection Results: Bias .................................................. 39
3.2 EKF Fault Detection Results: Drift .................................................. 39
3.3 EKF Fault Detection Results: Increased Noise ................................. 39
LIST OF FIGURES

2.1 Mikrokopter Hexacopter XL Test Vehicle ........................................ 7
2.2 Close-up View of Sensor Payload .................................................. 8
2.3 Hokuyo URG-04LX Scanning Laser Rangefinder ................................ 9
2.4 ASUS Xtion Pro Live RGB-D Camera .............................................. 10
2.5 Motion Capture Facility and Camera .............................................. 12
2.6 Truth System Capture Volume Depiction ....................................... 12
2.7 Hardware Communication Network Diagram ................................... 14
2.8 System Diagram ............................................................................. 16
2.9 System Diagram for First Hardware Tests ......................................... 17

3.1 Relative Navigation Using Nodes and Edges ..................................... 29
3.2 Playback Method for Estimation and Fault-detection ......................... 36
3.3 EKF Results Plot: 0.25 m Bias ...................................................... 40
3.4 EKF Results Plot: 0.02 m/s Drift .................................................. 42
3.5 EKF Results Plot: 0.035 m Standard Deviation Additive Noise ....... 43
3.6 Hardware Test Photos ...................................................................... 44

4.1 Parametric and Particle Representations Example ......................... 46
4.2 Problem with Weighted Mean Estimates ......................................... 51
4.3 Particle Filter Example: Initialization ........................................... 54
4.4 Particle Filter Example: Measurement Update 1 ............................. 55
4.5 Particle Filter Example: Resample 1 ............................................. 55
4.6 Particle Filter Example: Prediction ............................................... 56
4.7 Particle Filter Example: Measurement Update 2 ............................. 57
4.8 Particle Filter Example: Resample 2 ............................................. 58
4.9 Particle Filter Results: False-alarm ............................................... 65
4.10 Particle Filter Results: Bias ....................................................... 66
4.11 Particle Filter Results: Drift ....................................................... 67
4.12 Particle Filter Results: Increased Noise ........................................ 68
CHAPTER 1. INTRODUCTION

Unmanned aerial vehicles (UAV) of varying levels of autonomy are growing in popularity. Some require a great deal of operator involvement, while others are highly autonomous. Autonomous aircraft are the target of this research. These aircraft use mission objectives and other high level commands to compute control inputs and other actions. To do so successfully, UAV must be aware of their state at any given time. The state cannot in general be measured directly, nor can it be predicted by a dynamic model due to unmodeled effects. To determine its state then, an autonomous aircraft must employ an estimator, which fuses knowledge of the system model with measurements from a variety of sensors to yield an estimate. This estimate is then used in necessary decision-making processes. The quality of a state estimate then, is dependent upon the quality of the system model and the measurements it receives.

1.1 Motivation

1.1.1 Problem Description

As mentioned, to operate, autonomous systems of necessity employ a variety of sensors to perceive their environment. It is often presumed that sensors correctly measure their intended quantities, within some assumed probability distribution (usually Gaussian). This noise is part of normal operation and is handled by the system’s estimator. However, deviation of sensors from this idealized behavior, or sensor faults can cause an estimator to diverge and result in undesirable system behavior and instability. In the context of UAV, the consequences of such behavior could include injury to persons or damage to property. To avoid these consequences, a robust sensor fault-detection scheme must be implemented.
One common approach to fault-detection is the use and comparison of physically redundant sensors. This method is simple and works very effectively, but requires a minimum of two identical sensors to detect faults and a minimum of three to isolate the faulty sensor [1]. This limits the applicability of this method, as implementing physical redundancy is not always possible. Small UAV, for example, are unable to carry redundant sensors due to size, weight, and power (SWaP) constraints. This precludes the use of physical redundancy for fault-detection for these aircraft. This thesis describes a robust method of detecting sensor faults in the absence of physical redundancy that has been verified through hardware testing.

1.1.2 Significance

The global UAV market is expanding at a very rapid rate. According to [2], before the war on terror began in 2001, the Pentagon had fewer than 50 unmanned aircraft, whereas now it has approximately 7,500. A recent report detailing the projected science and technology efforts of the US Air Force [3] states that “an increasing shift to remotely piloted systems is inevitable over the next decade.” Another similar report, [4], was released by the Office of the US Secretary of Defense that outlined desired improvements in unmanned aircraft system (UAS) capabilities. According to [5], there are a number of attractive civil applications of UAV as well. These include remote sensing/monitoring, disaster response, search and rescue, transport, comm relay, delivery, and cinematography, among others. According to their Aerospace Forecast for 2013 [6], the Federal Aviation Administration (FAA) is pursuing regulations that will allow more widespread UAS access to the National Airspace System (NAS). They predict that within five years of these regulations taking effect, approximately 7,500 commercial small unmanned aircraft systems (sUAS) will be viable. Thus, we can expect a great increase in the number of UAS in service both in the military and in the civil sector in the near future.

One of the major points in both the Air Force and Department of Defense reports ( [3], [4]) was that there is a need for greater autonomy in unmanned systems. There is a concern that many existing systems rely too heavily on a human in the loop. Additionally, an increase in the number of UAS in service will no doubt raise many concerns, chief among them being safety. In a news article the FAA released in late 2012 [7], they stated, ”The
FAA’s sole mission ... as it focuses on the integration of unmanned aircraft systems is safety.” The ability of an autonomous system to detect faults in its sensors is very important for safe operation, especially with the push for greater autonomy. And, because many systems are subject to strict SWaP constraints, the need for a method to detect faults in non-redundant sensors is needed.

1.2 Technical Challenges

The problem of detecting sensor faults in non-redundant sensors is not trivial. There are a number of factors that make a solution difficult. One is an inability to perfectly characterize systems, including sensors. Unmodeled dynamics, unknown environmental factors, and non-trivial noise characteristics all contribute to this difficulty. The proliferation of inexpensive commercial off the shelf (COTS) sensors compounds this problem, as they generally possess less desirable noise characteristics. Indeed, a robust model may be out of reach for many systems.

Additionally, the small UAVs to which we would like to apply this method are generally subject to strict SWaP constraints, limiting the amount of computation that can be performed. Thus, even in the case that a robust model is found, it may not be feasible to use the robust model for fault-detection in real time. For example, this method may be necessary on a small UAV performing simultaneous localization and mapping (SLAM) or some other computationally expensive operation, and the added overhead of a complex algorithm would be prohibitive. Thus, a method is needed which adds very little computational overhead if it is to be integrated into such a system.

An additional requirement for a robust fault-detection system is the ability to detect varying fault types. It must be able to detect slowly-developing faults such as drift in addition to abrupt faults such as bias. It is also desirable to detect a wide range of fault magnitudes. This is made difficult by the need to strike a balance between correctly detecting faults and generating too many false-alarms. The algorithm must also detect faults quickly, as the use of faulty sensor measurements for update can quickly lead to system instability.
1.3 Technical Approach

One approach to fault-detection where physical redundancy is not possible is to simulate physical redundancy using analytical redundancy [8]. This is achieved by using knowledge of the system along with available weakly-correlated sensors to determine the health of a sensor of interest. The use of analytical redundancy to aid in fault-detection has received a great deal of attention in the literature. A series of survey papers [8–10] groups the approaches into three categories: quantitative model-based methods, qualitative model-based methods, and process history based methods. Each of these types of methods is briefly described below, with different examples mentioned.

Quantitative model-based methods use mathematical expressions to model a system. Differences between the expected and actual system behavior are then used as fault indicators. Examples of strategies that fall into this category are parity equation approaches [11], Kalman filter based approaches [12,13], and parameter estimation techniques [14], which all require accurate models of the system.

Qualitative model-based methods are developed based on some fundamental understanding of the process. Order of magnitude comparisons and similar qualitative comparisons between expected and actual system output are used to determine whether the system is functioning properly. Examples of strategies that fall into this category include digraphs [15] and qualitative physics methods [16], which require only qualitative models.

In contrast to both of the model-based methods, process history-based methods do not require a priori knowledge about the system; they instead require a large amount of historical data. This data is then transformed into a priori knowledge through a feature extraction process. Examples of this type of strategy include statistical feature extraction [17] and principle component analysis/partial least squares [18]. The examples given here are just a few of the numerous fault-detection methods in existence.

In pursuit of a robust fault-detection method, this research considers two model-based methods of fault-detection. The two methods considered are an extended Kalman filter (EKF) based method, and a particle filter (PF) based method. These methods were selected because they possess many desirable qualities, such as ease of use and their ubiquity
as estimation methods among autonomous systems. For systems that already employ one of these filters, this results in very little overhead for fault-detection.

1.4 Contributions

A number of contributions were made as part of this research, both to the field of fault-detection and to the BYU MAGICC lab. They are:

• **Demonstrated Real-Time Onboard Fault Detection Capability** - This research shows that the proposed EKF based fault-detection method is capable of detecting bias, drift, and increased sensor noise real-time onboard an autonomous rotorcraft. Such results have not been found in the literature.

• **Implementation of Particle Filter Based Fault-Detection Method in Simulation** - This work shows the implementation of a particle filter based fault-detection method and its comparison to the Kalman filter based method through simulation.

• **Presentation and Publication of Work at 2013 AIAA GNC Conference** - The work detailing the Kalman filter based method of fault-detection was presented and subsequently published in the conference proceedings [19].

The remainder of this thesis is structured as follows. Chapter 2 provides details of a test scenario that has been implemented to demonstrate the efficacy of the proposed EKF based fault-detection algorithm. Next, Chapter 3 describes the EKF based method, while the PF based method is discussed in Chapter 4. Finally, conclusions and suggestions for future work are presented in Chapter 5.
CHAPTER 2. EXPERIMENTAL SETUP

This chapter details the hardware testing scenario and the experimental setup employed to perform the desired tests. Section 2.1 describes the test scenario. Section 2.2 details the various hardware components used for the experiments. The software components of the experimental setup are described next in Section 2.3. Finally, Section 2.4 outlines the testing procedure.

2.1 Scenario

The scenario selected for testing the EKF algorithm focuses on the detection of faults in a laser rangefinder acting as a height-above-ground (HAG) sensor onboard an autonomous rotorcraft. The rotorcraft is flown in a controlled indoor environment. A flat ground is assumed for simplicity, however, this condition could be relaxed given availability of a digital elevation map. Additional sensors onboard the rotorcraft include an inertial measurement unit (IMU), which includes accelerometers and rate gyros for each of the three principal axes. It also incorporates a forward-looking RGB-D camera, which provides updates by means of a visual odometry (VO) algorithm. Faults tested include varying levels of bias, drift, and increased noise.

2.2 Hardware Description

The hardware implementation of the algorithm is very important. It shows that the theoretical algorithm can be successfully adapted to achieve fault-detection on a real system. It also demonstrates that the algorithm is capable of real-time onboard implementation. This section describes each of the major components of the hardware in more detail.
2.2.1 Vehicle

The vehicle used for these experiments is the Mikrokopter Hexacopter XL. The XL designation indicates its increased size and payload capacity. We outfit the vehicle with six MK-3638 brushless DC motors and 13 inch propellers. This gives the vehicle a payload capacity sufficient to actuate our test vehicle, which weighs 4.05 kg. The vehicle also measures approximately 1.1 m in diameter. The large size and payload capacity allow us to incorporate all the sensors and the onboard computer. These are mounted on a platform suspended underneath the main body of the vehicle, as seen in Figure 2.1. A closer view of the sensors is shown in Figure 2.2. The rotorcraft is controlled via the Mikrokopter Flight-Ctrl V2.1 ME autopilot. Commands are sent to the autopilot from the onboard computer via a serial connection at approximately 20 Hz. The autopilot also allows manual override of the controls via remote control. This is especially useful in cases where the vehicle becomes unstable and remote intervention is required.

Figure 2.1: Mikrokopter Hexacopter XL used for testing. The sensor payload is mounted on the suspended platform below the main body of the vehicle. The vehicle measures approximately 1 m in diameter and weighs approximately 4 kg.
2.2.2 IMU

Many autonomous vehicles employ an IMU to help determine changes in orientation and velocity. This is an important input to the state estimator. The IMU used for these experiments is the MicroStrain® 3DM-GX3®-15. This device incorporates MEMS accelerometers and gyroscopes for three orthogonal axes. Initial accelerometer bias for this model is listed as ±0.002g with an in-run bias stability of ±0.04mg [20]. Initial gyroscope bias is listed as ±0.25°/s with an in-run bias stability of 18°/hr. The IMU is mounted on a vibration-isolation platform to avoid high-frequency noise from the rotors of the vehicle. For these tests, it streams accelerometer and gyroscope measurements via an RS-232 serial connection to the onboard computer at a rate of 100 Hz.
2.2.3 Laser Rangefinder

The laser rangefinder is the sensor of interest in our experiments. The specific laser used is the Hokuyo URG-04 LX Scanning Laser Rangefinder (see Figure 2.3). This laser has a minimum range of 60 mm and a maximum range of approximately 4 m. It also has a field of view of 240 degrees with a 0.36 degree resolution. Because this sensor is to be used as a height-above-ground sensor, only the center scan line is used. This also reinforces the single-threaded nature we want to impart in our test scenario. The laser rangefinder also possesses good measurement properties. In our scenario, we operate in an environment that reduces the possibility of sensor faults occurring in the laser. We also operate well within the range of the sensor. These facts, combined with the high accuracy of the laser, allow us to have a high level of confidence that the only faults present in the laser datastream are those we inject. This control over the insertion of errors is important in the analysis of our algorithm’s performance. The laser rangefinder is capable of providing measurements at a rate of 10 Hz, which it does in these tests. Communication with the device occurs over a USB connection to the onboard computer.

Figure 2.3: Hokuyo URG-04LX Scanning Laser Rangefinder. This sensor is used as the sensor of interest in the fault-detection algorithm and tests.
2.2.4 RGB-D Camera

An RGB-D camera is used to perform visual odometry (VO) onboard the vehicle. As mentioned in [21], RGB-D cameras are growing in popularity, in part due to the fact that they use infrared projection to provide 3D data, which is mostly independent of illumination. Vision algorithms using this type of device, then, are more robust than those employing traditional stereo-vision cameras. The RGB-D camera used in our experiments is the ASUS Xtion Pro Live camera, shown in Figure 2.4. It comprises an RGB camera, an infrared projector, and an infrared receiver. According to the manufacturer ([22]), the camera consumes less than 2.5 W, has a range of between 0.8 and 3.5 meters, has a 58° horizontal field of view, a 45° vertical field of view, and outputs a 640x480 pixel depth image at up to 30 frames per second. For the purposes of this research it is sufficient to downsample this to 15 Hz. The camera communicates over USB with the onboard computer.

2.2.5 Onboard Computer

The onboard computer is an important part of the autonomous system. It is where most of the computation is performed, exceptions being the selection of goal locations and the acquiring of truth data for control and evaluation of algorithm performance. The onboard computer used for these tests is built upon a Global American EPI-QM67 EPIC form-factor
motherboard. This board measures 16.5 cm by 11.5 cm and weighs only 0.46 kg. The processor used on this board is an Intel® Core i7-2710QE, which contains four cores clocked at 2.1 GHz. A solid-state drive is used onboard the vehicle to both increase read/write speed and to prevent damage to the disk due to vibration and crashes of the platform. A 120 GB Intel® SSD 320-series 2.5 inch hard drive is used for this application. The operating system installed on the onboard computer is Ubuntu 12.04 LTS. This is a long-term support version of Ubuntu, making it an attractive choice for installation. Ubuntu is also the only officially supported operating system for ROS (the Robot Operating System) [23], which we chose to use for this project. More information about ROS and its role in the tests is given in Section 2.3.

2.2.6 Groundstation

A groundstation is used in these tests as a way for the operator to interact with the rotorcraft during flight. It shows the state of the vehicle and other system outputs, including the fault-detection decision. Through this interface, the operator can also select goal locations for the rotorcraft. This goal location is then transmitted to the aircraft. Additionally, truth data is passed from the groundstation to the aircraft. This truth data is not used by the onboard estimator or the fault-detection algorithm, but is used for controlling the aircraft, to avoid instability due to the injected faults. The groundstation employed for these tests is a desktop PC with Ubuntu 12.04 LTS installed.

2.2.7 Truth Acquisition System

To evaluate the performance of the fault-detection algorithm, it is necessary to have access to the true state of the vehicle at any given time. The Brigham Young University MAGICC Lab has an indoor flight facility with a system that provides this information. The system comprises eight Hawk digital IR cameras (see Figure 2.5a) and motion capture software from Motion Analysis Corporation. The fields of view of these cameras creates a capture volume measuring approximately 2.8m x 4.3m on the ground and 1.8m high. A representation of the capture volume is shown in Figure 2.6. The system is capable of
resolving the pose of the vehicle to within sub-millimeter and sub-degree accuracy at a rate of up to 200 Hz. Because this system only provides position and orientation information, rate information is found by numerically differentiating the output. In addition to providing truth information for comparison, the truth data can be used to control the aircraft. This is useful in our tests as the faulty sensor information injected into the estimator has the potential to cause aircraft instability and damage. Thus, truth data is used for aircraft control, with the estimator running separately. The truth data streams from the cameras.
over Ethernet to a second groundstation and is then transferred via Ethernet to the main groundstation. From here, this information is relayed to the vehicle. This brings us to the topic of system communication.

2.2.8 Communication

The communication system for these experiments is described in this section. The first component we will discuss is the vehicle’s autopilot. The autopilot must receive commands both via remote control and from the onboard computer. The remote control used in these experiments is the JR/DSM X9503, which uses a Spektrum DSMX 2.4 GHz RF link. The autopilot receives commands from the remote control through a Spektrum Satellite receiver, which is installed onboard the vehicle. This allows manual flight of the aircraft, including emergency takeover in the event of aircraft instability. The autopilot receives commands from the onboard computer through a serial connection. Another important communication link is between the onboard computer and the groundstation. An 802.11n wireless network router is used for this purpose. The groundstation sends goal locations and truth data to the onboard computer over the wireless network. The onboard computer also send information to the groundstation such as the fault-detection status of the laser. Additionally, the groundstation receives true pose information from another computer that is connected to the IR cameras via Ethernet. The diagram in Figure 2.7 shows this communication network. All onboard communication (e.g., between different sensors, the estimator, and the autopilot) is handled through ROS. The next section details this part of the system.

2.3 ROS

As noted previously, to perform hardware tests, the Robot Operating System (ROS) framework is employed. This section gives details about ROS and how it is used in our hardware tests.
2.3.1 Overview

ROS was developed to facilitate the writing of software for robotic systems. According to [24], one of the design goals of ROS was to create a peer-to-peer topology. This structure allows a system to consist of many processes, possibly distributed across different hosts, that communicate with each other. These processes are called nodes. ROS facilitates this communication by allowing operators to specify message types that nodes can emit and/or receive. These messages are like C structures which have strict types, and as such, are able to be created or received by any node that knows the message type. Nodes send and receive messages by publishing and listening to topics. Topics are set up such that multiple nodes can publish and subscribe to them. This architecture consisting of nodes, messages, and topics makes ROS very flexible. The creators of ROS also allowed for the ability to log and play back messages via the rosbag record and rosbag play utilities. This is very useful in debugging and experimentation, as a dataset can be played back again and again. This allows a new system to be set up to listen to the replayed topics and process that data as if it were streaming from the robotic system. The data can then be analyzed and manipulated by new nodes. The interested reader can learn more about ROS by reading reference [24].
2.3.2 Implementation

In our experiments, both the groundstation and the onboard computer have ROS Fuerte installed. The architecture of the system is shown in Figure 2.8. This figure shows hardware components of our system as blue oblate hexagons and ROS nodes as yellow rectangles with solid borders, with the nodes inside of gray rectangles with dashed borders indicating on which computer the node is run. As shown, the groundstation houses only the node which receives motion capture information and relays that to nodes running on the onboard computer and the node that manages the user interface (UI). The user interface allows the operator to pick goal locations for the vehicle. It also outputs pose information and the current fault decision to the groundstation monitor.

The onboard computer runs the rest of the nodes for this system. This includes a path planning node that takes in true pose information and operator-selected goal locations to generate a plan which is published for subscription by a low-level control node. This node uses knowledge of the true pose of the vehicle and the planned path it receives from the path planner to issue low-level commands to the autopilot. The autopilot in turn communicates information back to the estimator. There are also nodes running on the onboard computer that gather information from the sensors. For example, the IMU node gets data from the IMU and puts it into a message that it publishes for the estimator to subscribe to. The RGB-D camera feeds information into the visual odometry (VO) node, which computes visual odometry updates and publishes them on the VO topic. The estimator listens to this topic and processes new VO information when it is available. The laser relay node, as depicted here, is sometimes actually two nodes. The first is a node that receives laser measurements and creates the appropriate message. The second is a node that corrupts the laser measurement message with a specified fault type. This is useful in our tests as it allows us to take good sensor data collected during a flight and corrupt it in many different ways to analyze the effects of different faults. The estimator node processes the measurement topics of these sensors and predicts the state of the vehicle using a multiplicative extended Kalman filter (MEKF). More details about this filter will be given in Section 3.5. The detection of faults in the laser altimeter is included in the estimator and the fault-detection decision is
Figure 2.8: Diagram of the Hardware/Software Setup. Hardware components are represented by blue oblate hexagons and ROS nodes are represented by yellow rectangles situated within larger dashed gray rectangles representing the computer on which the nodes resided. Note that the groundstation only served to pass truth data to the planner and controller and to pass operator-defined goal locations to the planner.

emitted as a message from the estimator which is also visualized through the user interface node.

2.4 Testing Procedure

This section explains the procedures associated with the tests performed as part of this research. The fault modes that are tested include: no fault, bias, drift, and increased sensor noise. In each test case, the Hexacopter is flown autonomously using truth data acquired through the motion capture system so as to maintain control of the vehicle despite
the possibility of degraded estimates when injecting faults into the laser altimeter datastream. Three groups of tests are performed.

The first group of tests involves recording flight data using the `rosbag record` utility. All topics necessary for running the estimator, along with topics required for analysis are recorded. Then, a new ROS system is set up on a separate desktop computer for performing offline analysis of the data collected during these flights. A diagram of this system is shown in Figure 2.9. This system allows the operator to use the `rosbag play` tool to replay the data from the flight. The data, with exception for the laser data is relayed to the estimator as it was recorded. The laser data is first passed to a node that injects known faults into the datastream and then relays those corrupted laser measurements to the estimator. This

![Figure 2.9: Diagram of system setup for the first set of tests. The recorded data is played back via the `rosbag play` utility. The uncorrupted laser data flows through the laser relay block, which creates a faulty laser datastream with a specified fault type and magnitude. This information is then used by the estimator to estimate the rotorcraft state and to determine whether or not laser measurements are faulty.](image-url)
allows for determination of fault-detection capabilities of the algorithm. This data, together with the estimated states and fault-detection decision variables is then recorded to a new file for analysis using MATLAB.

The second group of tests involves detection of faults onboard the aircraft in real time. The faults are injected in the same way as in the first group of tests. The original laser measurements are passed through the laser relay node, which corrupts the datastream with known fault types and magnitudes before passing that information on to the estimator. All pertinent information required for analysis is recorded for analysis using MATLAB.

The third group of tests also involves detection of faults onboard the aircraft in real time. The faults, however, are not injected in software but are physically injected. The only fault tested in this way is a constant bias. The fault is injected by placing a raised platform on the ground underneath the vehicle’s flight path. Due to the nature of the algorithm, this unmodeled change in environment has the same effect as a fault in the sensor measurement. All pertinent information required for analysis is again recorded for analysis using MATLAB. Additionally, the algorithm’s fault decision is streamed to the groundstation in real time and displayed for the operator to see.

More details of these tests will be given in Chapter 3, Section 3.6, which details the specific tests performed and also discusses the results of those tests.
CHAPTER 3. EKF-BASED FAULT DETECTION

This chapter focuses on the EKF-based fault-detection method developed as part of this research. Section 3.1 explains the motivation for developing an EKF-based fault-detection method. Background information is given in Section 3.2. The EKF is then described in Section 3.3, including the notation used in this paper. Section 3.4 then details the fault-detection method developed as part of this research. The estimator for the experimental platform is then detailed in Section 3.5. Next, Section 3.6 presents and discusses results of the experiments detailed in Chapter 2. Finally, conclusions and suggestions for future work are presented in Section 3.7.

3.1 Motivation

An EKF-based algorithm was developed to address the problem of fault-detection. There are a number of reasons that an EKF was chosen as the basis for the developed fault-detection method. The first reason is the intuitive nature of the EKF. The procedure for an EKF can be understood as a simple predict-correct procedure, where we first predict a system’s state at a time, \( t \), and then correct that estimate using sensor measurements as we receive them. Another reason an EKF-based method is attractive is its ubiquity. According to [25], the EKF is likely the most-used method for nonlinear estimation problems. An EKF-based fault-detection method, then, can be integrated into many nonlinear systems with very little modification to the system’s current estimator. This results in shorter integration times and leads to decreased algorithmic overhead. This is especially desirable onboard small UAVs, which cannot accommodate computationally expensive methods of fault-detection.
3.2 Background

The EKF-based method employed here is based primarily on the method detailed in [12] and is implemented and tested for a scenario involving a height-above-ground sensor onboard an autonomous rotorcraft. Other work performed in the area of fault-detection for autonomous rotorcraft includes the work in [26] and [27]. As they note, rotorcraft are an ideal target for a robust fault-detection method because they are inherently unstable and have fast dynamics. Their work, like ours, employs model-based methods of fault-detection for autonomous rotorcraft. However, the method proposed in this research differs from their work in a few important ways. First, the work in [26] employs a Luenberger observer, which is deterministic, rather than a stochastic Kalman filter. This determinism will result in decreased robustness of the algorithm. They also assume that a fault is present the first time a innovation exceeds a threshold, which can lead to higher false-alarm rates. Additionally, the approach outlined in [27] uses a linear model for fault-detection as opposed to the more accurate nonlinear model we propose for our fault-detection approach. Also, while the researchers note that their experimental results were obtained using real flight data, no indication is given that fault-detection could be performed in real time. We will show that our approach is capable of real-time on-board implementation, a result we have not found in the literature.

3.3 The Extended Kalman Filter (EKF)

This section introduces the extended Kalman filter. The notation used here is standard and can be found, for example, in [28], from which this introduction is adapted.

The EKF is based on the Kalman filter [29], which is an optimal filter for known linear systems of the form

\[ x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \]  \hspace{1cm} (3.1)

\[ z_k = Hx_k + \nu_k. \]  \hspace{1cm} (3.2)
The matrices $A$ and $B$ map the state, $\bar{x}$, and input, $\bar{u}$, respectively, at time $k-1$ to the state at time $k$, and the matrix $H$ maps the current state to measurement space. The terms $\bar{w}_{k-1}$ and $\bar{\nu}_{k-1}$ represent the process and measurement noise, respectively, and are typically assumed to be zero mean multivariate Gaussian random variables, with probability distributions

$$p(\bar{w}_k) \sim \mathcal{N}(0, Q_k) \quad (3.3)$$
$$p(\bar{\nu}_k) \sim \mathcal{N}(0, R_k). \quad (3.4)$$

The extended Kalman filter (EKF) extends the Kalman filter to nonlinear systems of the general form

$$\bar{x}_k = f(\bar{x}_{k-1}, \bar{u}_{k-1}, \bar{w}_{k-1}) \quad (3.5)$$
$$\bar{z}_k = h(\bar{x}_k, \bar{\nu}_k). \quad (3.6)$$

This allows for more widespread implementation than the Kalman filter, as most systems of interest are not linear. The EKF achieves this by using multivariate Taylor Series expansions to linearize these functions about the current state and input.

The EKF, like the Kalman filter, is an iterative filter and follows a predict-then-update procedure where it first estimates the state by performing a time update and then adjusts that estimate based on measurement information. The general equations of the filter are given in Equations 3.7 - 3.17. First, the time update is performed, whose equations are given by

$$\hat{\bar{x}}_k^- = f(\hat{\bar{x}}_{k-1}, \bar{u}_{k-1}, 0) \quad (3.7)$$
$$P_k^- = A_{k-1}P_{k-1}A_{k-1}^T + B_{k-1}G_{k-1}B_{k-1}^T + W_{k-1}Q_{k-1}W_{k-1}^T, \quad (3.8)$$
where

\[ A_{k-1} = \frac{\partial f}{\partial \bar{x}} \bigg|_{(\hat{x}_{k-1}, \hat{u}_{k-1}, 0)} \]  
(3.9)

\[ B_{k-1} = \frac{\partial f}{\partial \bar{u}} \bigg|_{(\hat{x}_{k-1}, \hat{y}_{k-1}, 0)} \]  
(3.10)

\[ W_{k-1} = \frac{\partial f}{\partial \bar{w}} \bigg|_{(\hat{x}_{k-1}, \hat{y}_{k-1}, 0)} \]  
(3.11)

are the Jacobians of the dynamic update equation with respect to the prior state, the input, and the process noise, respectively, and \( G_{k-1} \) is the covariance of the input vector, \( \bar{u}_{k-1} \).

The variables \( \hat{x}^- \) and \( P^- \) in these equations represent the *a priori* (prior to measurement update) state estimate and covariance, respectively, and the variables \( \hat{x} \) and \( P \) represent the *a posteriori* (after measurement update) state estimate and covariance. As can be seen, the time-update equation for the mean of the estimate (Equation 3.7) uses the nonlinear state update equation to generate the a priori state estimate. The covariance of the a priori estimate is computed by mapping the covariances on the prior state, \( P_{k-1} \), the prior input, \( G_{k-1} \), and the Gaussian prediction uncertainty term, \( Q_{k-1} \) to the current time step via multiplication by the Jacobians of the nonlinear system equations, given in Equations 3.9-3.11. This is a result of using a first-order Taylor series expansion to linearize the system equations about the current estimate.

After the time update is performed, a measurement update is applied. The measurement update equations are given by

\[ r_k = z_k - h(\hat{x}_-, 0) \]  
(3.13)

\[ C_k = H_k P^- H_k^T + R_k \]  
(3.14)

\[ K_k = P^- H_k^T C_k^{-1} \]  
(3.15)

\[ \hat{x}_k = \hat{x}_- + K_k r_k \]  
(3.16)

\[ P_k = P^- - K_k H_k P^- \]  
(3.17)
where

\[ H_k = \frac{\partial h}{\partial \bar{x}} \bigg|_{\hat{x}_0} \]  

(3.18)

is the Jacobian of the measurement equation with respect to the a priori state estimate.

The first in the set of measurement equations (Equation 3.13) shows the computation of the innovation (or residual) of the filter, or the difference between the predicted and actual sensor measurements. This quantity, along with its covariance (Equation 3.14) are of particular interest for the development of the fault-detection algorithm proposed in this chapter. Equation 3.15 shows the computation of the Kalman gain. This term is essentially a weighting for how much to trust the time update versus the measurement coming from the sensor. A large Kalman gain will result in the sensor being trusted and, in the case that the measurement differs from the predicted measurement, the predicted state will be adjusted toward the measured value. A small Kalman gain, however, indicates a higher certainty of the predicted value compared to the sensor measurement. Thus, in the event that the predicted and actual measurements differ, the a posteriori state estimate will not be significantly different from the a priori state estimate. This can easily be seen by considering Equations 3.13 and 3.16. The innovation is multiplied in Equation 3.16 by the Kalman gain, \( K_k \), and their product is added to the a priori state estimate to yield the a posteriori state estimate. Thus, if the Kalman gain is small, the a posteriori estimate will not differ significantly from the a priori state estimate (measurements have little effect on the estimates). The Kalman gain is also used to compute the new covariance of the state estimate, as seen in Equation 3.17.

This process of prediction and measurement update is iterated as appropriate to yield estimates of the system state. The EKF is the basis for the fault-detection algorithm employed in this research, which is described in the following section.

### 3.4 EKF Fault Detection Method

As stated in the introduction, this method is based primarily on the method detailed in [12], which prescribes hypothesis tests for the innovation sequence of the state estimator.
as a means of detecting faults. The method begins with the computation of the innovation each time a measurement is received

\[ r_k = z_k - \hat{z}_k \]  

(3.19)

where \( z_k \) is the measurement from the sensor of interest at time \( k \) and \( \hat{z}_k \) is the predicted measurement generated by the model. The innovation is computed during the measurement update step of the Kalman filter (Equation 3.13). The covariance of the innovation term is then computed as

\[
C_k = E[r_k r_k^T] \\
= E[(z_k - \hat{z}_k)(z_k - \hat{z}_k)^T] \\
= E[(z_k - H_k \hat{x}_k^-)(z_k - H_k \hat{x}_k^-)^T] \\
= E[z_k z_k^T] - H_k E[\hat{x}_k^- \hat{x}_k^-^T] - H_k E[\hat{x}_k^- z_k^T] + H_k E[z_k \hat{x}_k^-^T] H_k^T \\
= R_k + H_k P_k^- H_k^T, 
\]

(3.20)

where \( H_k \) is the Jacobian of the measurement function with respect to the state of the system and \( R_k \) the covariance of the sensor(s) used in the update. The innovation is then normalized by computing

\[
\eta_k = C_k^{-\frac{1}{2}} r_k, 
\]

(3.21)

where \( k \) denotes the measurement index.

It is important to note that when the filter accurately predicts the state of the system and the sensor is functioning properly, the sequence of normalized innovation terms, \( \{\eta_0, \eta_1, \ldots, \eta_k\} \), is a zero-mean Gaussian white noise process with covariance \( I \). We will assume that our estimator is sufficiently accurate as to produce such an innovation sequence in the absence of sensor faults. Thus, faults in the system are recognized by deviations of the innovation sequence from its zero-mean, unit covariance, white noise properties. Significance of any deviations from these nominal properties can be assessed through hypothesis
testing. Following the work of [12], hypothesis tests on both the mean and the covariance of the innovation sequence are used to detect faults. These tests require that the innovation sequence be a white noise process. We do not explicitly perform a test of whiteness, as doing so results in a very large number of false-alarms. Instead, we assume whiteness and perform the tests of mean and covariance. These tests are performed locally using a small sliding window to promote fast detection and to reduce memory requirements. A measurement that causes either test to fail is flagged as a faulty measurement. These tests are preceded by a test that identifies measurements outside the functional range of the sensor. Details of the tests are given below.

3.4.1 Test of Mean

The test of mean is designed to determine whether the mean of the normalized innovation sequence is zero with some level of significance. This is verified by the following hypothesis test.

\[ H_0 : \mu = \mu_0 = 0 \]
\[ H_1 : \mu \neq \mu_0 = 0 \]  

(3.22)

To perform this test, first an estimate of the sequence mean, \( \hat{\mu} \), must be computed. The Maximum Likelihood (ML) estimate for the mean (the sample mean) given by

\[ \hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \eta_i, \]

(3.23)

is used, where \( N \) is the window size chosen for the test. The test statistic [30]

\[ Z^2 = N(\hat{\mu} - \mu_0)^T \Sigma^{-1}(\hat{\mu} - \mu_0) \]

(3.24)

is then constructed using an assumed known covariance, \( \Sigma \). For our tests, \( \Sigma = I \). This parallels the standard normal test for the univariate case. Under the null hypothesis, the test statistic \( Z^2 \) is distributed as a \( \chi^2 \) random variable with \( m \) degrees of freedom, where \( m \)
is the dimension of the innovation. Thus, the test of mean is given by

\[ Z^2 \overset{D_1}{\gtrless} D_0 Q_{\chi^2} (m, \alpha) \]  

(3.25)

where \( \alpha \) is the desired level of confidence, \( D_0 \) and \( D_1 \) represent failure to reject the null hypothesis and rejection of the null hypothesis, respectively, and \( Q_{\chi^2} (k, p) \) is the quantile function (inverse cumulative distribution function) for a \( \chi^2 \) distribution with \( k \) degrees of freedom and probability \( p \).

### 3.4.2 Test of Covariance

As in the test of mean, the test of covariance considers the no fault case as the null hypothesis, yielding

\[ H_0 : \Sigma = I \]

\[ H_1 : \Sigma \neq I \]  

(3.26)

The unbiased sample covariance,

\[ S = \frac{1}{N} \sum_{i=1}^{N} (\bar{\eta}_i - \hat{\mu})(\bar{\eta}_i - \hat{\mu})^\top, \]  

(3.27)

is used to perform this test. According to [31], \( \text{tr}(NS) \) has a \( \chi^2 \) distribution with \( m \nu \) degrees of freedom, where \( m \) is the dimension of \( S \) and \( \nu = N \). Thus, the test of covariance is denoted

\[ \text{tr}(NS) \overset{D_1}{\gtrless} D_0 Q_{\chi^2} (m \nu, \alpha). \]  

(3.28)

There are a number of other tests of covariance proposed in the literature [13, 30, 32, 33], however this method is very simple and has been shown to yield good results.

### 3.4.3 Threshold Test

The aforementioned tests flag individual measurements as faulty or not faulty. Our goal, however, is not simply to determine which measurements fall outside predicted bounds, but to determine when a sensor persists in a faulty state. Hence, a third test is added to
make the determination of sensor health. Declaring a sensor faulty when either test flags a measurement as faulty results in a higher than desired level of false-alarms. The number of false-alarms is reduced through the use of this threshold test as the sensor is declared faulty only when a large percentage of the most recent measurements are flagged by either the test of mean or the test of covariance. This prevents the vehicle from distrusting the sensor after only a few flagged measurements. A window of size $W$ is used for this test. Letting $n$ represent the number of faults within the current window, we pick some threshold, $T < W$, such that the test

$$n \frac{D_1}{D_0} \geq T$$

(3.29)

yields a decision about the health of the sensor. Here again, $D_0$ represents failure to declare a fault and $D_1$ represents the declaration of a fault. Window size for this test can be chosen independently of the window size for the tests of mean and covariance. Choosing a larger window will delay fault-detection and sometimes mask legitimate fault declarations, whereas too small a window will not appreciably reduce the false-alarm rate. The operator must use discretion to choose an appropriate window size and threshold for this test. These quantities will vary from system to system and should be selected in light of the considerations above.

The entire EKF based fault-detection scheme is summarized in Algorithm 1.

### 3.5 Estimator

As detailed in Chapter 2, the scenario developed to test the algorithm requires the detection of faults in a laser rangefinder acting as a height-above-ground (HAG) sensor onboard an autonomous rotorcraft. Due to the nature of the fault-detection algorithm, degraded state estimates will have the same effects as sensor faults, hence an accurate estimator is required. This section details the estimator used in the experiments performed to test the fault-detection algorithm. Much credit goes to Robert Leishman, who developed this estimator and who contributed most of this section describing it.\(^1\)

The rotorcraft uses a relative navigation approach to achieve accurate estimates of its state, as explained below. Sensors available onboard the rotorcraft include an inertial

\(^1\)For a more detailed explanation of this estimator and the underlying improved dynamic model, see [34].
Algorithm 1 EKF Fault-Detection Algorithm

1: \( k = 0 \) % Time index
2: \( ii = 0 \) % Counter for updates to sensor of interest
3: \( \eta_{\text{seq}} = [] \) % List of innovation (residual) terms
4: \( f_m = [] \) % List of fault declarations (meas. level)
5: \textbf{procedure} Time Update(\( \hat{x}_{k-1}, u_{k-1} \))
6: \( \ldots \) % Perform the time update
7: \textbf{procedure} Measurement Update(\( \hat{x}_k, z_k \))
8: \( \text{flag} = 0 \)
9: \textbf{procedure} testOutOfBounds(\( \bar{z} \))
10: \textbf{if} \( \bar{z} > \bar{z}_{\text{max}} \) or \( \bar{z} < \bar{z}_{\text{min}} \) \textbf{then}
11: \( \text{flag} += 1 \)
12: \( \bar{r}_k = z_k - h(\hat{x}_k) \) % Compute residual
13: \( C_k = H_k P_k H_k + R_k \) % Compute residual covariance
14: \( \eta_k = C_k^{-\frac{1}{2}} \bar{r}_k \) % Normalize residual
15: \( \eta_{\text{seq}}.\text{append}(\eta_k) \)
16: \textbf{if} \( \text{len}(\eta_{\text{seq}}) > M \) \textbf{then}
17: \( \eta_{\text{seq}}.\text{popFront()} \) % Remove old entries
18: \( N = \text{len}(\eta_{\text{seq}}) \)
19: \textbf{procedure} testOfMean(\( \eta_{\text{seq}}, \mu_0, \Sigma, N \))
20: \( \hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \eta_{\text{seq}_i} \)
21: \( Z_{\text{sqr}} = N(\hat{\mu} - \mu_0)\Sigma^{-1}(\hat{\mu} - \mu_0) \)
22: \textbf{if} \( Z_{\text{sqr}} > \text{threshMean} \) \textbf{then}
23: \( \text{flag} += 1 \)
24: \textbf{procedure} testOfCovariance(\( \hat{\mu}, N \))
25: \( S = \frac{1}{N} \sum_{i=1}^{N} (\eta_{\text{seq}_i} - \hat{\mu})(\eta_{\text{seq}_i} - \hat{\mu})^\top \)
26: \textbf{if} trace(\( NS \)) > \text{threshCov} \textbf{then}
27: \( \text{flag} += 1 \)
28: \textbf{procedure} ThresholdTest \textbf{if} \( \text{flag} > 1 \) \textbf{then}
29: \( f_m.\text{append}(1) \)
30: \textbf{else}
31: \( f_m.\text{append}(0) \)
32: \( \) lastMeasStatuses = \( f_m[ii - W : ii] \)
33: totalFaults = \( \sum \) lastMeasStatuses
34: \textbf{if} totalFaults > \( \text{thresh} \) \textbf{then}
35: sensorFault = True
36: \textbf{else}
37: sensorFault = False
38: \( \ldots \) % Continue measurement update
39: \( ii + = 1 \)
40: \( k + = 1 \)
measurement unit (IMU), which includes accelerometers and rate gyros for each of the three principle axes. Additionally, updates come from a visual odometry (VO) algorithm [36], which produces measurements using a forward-looking RGB-D camera. The vehicle also employs a laser rangefinder as an altimeter. This is the sensor of interest of the experiments described in this thesis.

In [34] and [37], the authors propose that a vehicle should navigate using a relative formulation of the vehicle state, rather than a global one. A combination of graph SLAM and an EKF is used to provide mapping and sensor fusion. The map is a pose graph, with images from the onboard camera as key components of the nodes. The EKF provides estimates at the high rate required for feedback control of the vehicle. The difference over other approaches is that the position and yaw states of the EKF are defined with respect to the current node in the map, rather than to a global origin. Relative state information affords many advantages, such as the ability to directly utilize relative exteroceptive measurements, elimination of required feedback to the filter from computationally-expensive SLAM algorithms, easy creation of map edges using the filter state and covariance, and flexible use of global information.

Figure 3.1: Relative navigation using nodes and edges. As the vehicle flies through the environment, nodes are created using the VO keyframes and the edges are defined between them using the relative states of the MEKF. The vehicle state is relative to node four in this illustration.

The map in Figure 3.1 illustrates the relative topological approach. The visual odometry algorithm initializes a keyframe at node 1 and an edge is added between the global frame
and the node frame once this information is known. The filter estimates the position and
yaw states of the vehicle with respect to the local coordinate frame at node 1 as the vehi-
cle travels. When the VO requires a new keyframe to maintain good performance, a new
keyframe and node are declared at pose 2. An edge is added to the map using the relative
states and covariance in the EKF. The navigation then continues with respect to node 2 by
marginalizing out the old relative states and augmenting the state vector with new ones.
This process continues as the vehicle moves through the environment, with new keyframes
and nodes being declared as necessary and the EKF changing the relative states each time
a new keyframe is declared.

The estimator we have employed is a multiplicative extended Kalman filter (MEKF).
The MEKF is an indirect EKF, which means that the error in the state $\Delta \hat{x}$ is maintained
in the filter rather than the best estimate $\hat{x}$.

3.5.1 State Dynamics

The states $\mathbf{x}$ of the rotorcraft are

$$\mathbf{x} = \begin{bmatrix} p_n^b & q_b^n & v_b^n & \beta^b & \alpha^b \end{bmatrix}^T.$$  \hfill (3.30)

The relative position vector $p_n^b$ is the displacement of the body with respect to the current
node in the map. The quaternion $q_b^n$ expresses the attitude of the body-fixed frame with
respect to the node frame; it is relative to the current node in the map for yaw only. $v_b^n$ is the
body-fixed frame velocity vector. The gyroscope biases are in the vector $\beta$. Accelerometer
biases in the body $x$ and $y$ directions are represented by $\alpha$.

The inputs to the model are the gyroscope measurements and the $z$-accelerometer

$$\mathbf{u} = \begin{bmatrix} p_{\text{gyro}} & q_{\text{gyro}} & r_{\text{gyro}} & z_{\text{accel}} \end{bmatrix}^T.$$  \hfill (3.31)
The nonlinear update equations for the states (3.30) are

\[
\dot{p}^n = R^\top(q_n^b)v^b, \quad (3.32)
\]

\[
\dot{q}_n^b = \frac{1}{2} \Omega \left( u_{(1:3)} - \beta - \eta_\omega \right) q_n^b, \quad (3.33)
\]

\[
\dot{v}^b = v^b \times \left( u_{(1:3)} - \beta - \eta_\omega \right) + R(q_n^b)g
\]

\[ - \frac{1}{m} M v^b + u_{(4)} \vec{d}_j, \quad (3.34) \]

\[
\dot{\beta} = \eta_\beta, \quad (3.35)
\]

\[
\dot{\alpha} = \eta_\alpha, \quad (3.36)
\]

\[
\dot{q}_c^b = \eta_{cq}, \quad (3.37)
\]

\[
\dot{p}^b = \eta_{cp}. \quad (3.38)
\]

A rotation matrix $R(q_a^b)$ from a quaternion $q_a^b$ rotates the vector $v$, expressed in frame $a$, into frame $b$. The operator

\[
\Omega(\omega) = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix}
\]

assumes that the order of the quaternion it multiplies is $[q_x \ q_y \ q_z \ q_w]^\top$. The noise $\eta_\omega$ is the zero-mean Gaussian noise in the measured gyroscopes from the inputs $u$. The constant matrix $M$ is

\[
M = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (3.39)
\]

and the constants $g$ and $\mu$ are the gravity and rotor drag coefficient, respectively.
An improved model of the hexacopter dynamics in (3.34), which accounts for the rotor drag with coefficient $\mu$, provides the ability to fully utilize the information contained in the accelerometer measurements [34]. As a consequence, estimation accuracy improves and the requirements for view matching or any other exteroceptive measurement updates are reduced. This improvement in estimation accuracy also enhances our ability to detect faults in sensors.

3.5.2 Error Dynamics

The error dynamics, with error state $\Delta \bar{x}$, are used to propagate the error covariance matrix $P$ and are derived from the nonlinear dynamics (3.32) through (3.38) (see [34] for details).

The error dynamics can be linearized and result in a linear model

$$\dot{\Delta \bar{x}} = A \Delta \bar{x} + B \bar{u},$$  \hspace{1cm} (3.40)

where $A$ is the Jacobian of the error dynamics with respect to the error state $\Delta \bar{x}$ and $B$ is the Jacobian of the error dynamics with respect to the input $\bar{u}$.

3.5.3 Measurement Updates

We update the filter using laser, accelerometer, view-matching position, and view-matching orientation measurements. Each measurement update follows the same procedure, detailed in [34].

The innovation and its covariance are computed as in (3.19) and (3.20). The Kalman gain $K$ is

$$K = P^{-1} H^T C^{-1}.$$  \hspace{1cm} (3.41)

The correction (or updated error state) $\Delta \hat{\bar{x}}$ is computed as

$$\Delta \hat{\bar{x}} = K \bar{r}.$$  \hspace{1cm} (3.42)
The covariance is updated using

\[ P = (I - LH) P^-. \]  \hspace{1cm} (3.43)

We use the correction (3.42) to update the current state estimate \( \hat{x} \). A component \( a \) of the state, that is not a quaternion, is updated using

\[ \hat{a}^+ = \hat{a} + \delta \hat{a}^+. \]  \hspace{1cm} (3.44)

The quaternions in the state are updated according to

\[ \hat{q}^+ = \delta \hat{q}^+ \otimes \hat{q}, \]  \hspace{1cm} (3.45)

where

\[ \delta \hat{q}^+ = \left[ \frac{\delta \hat{q}^+_{vec}}{\sqrt{1 - \left( \frac{\delta \hat{q}^+_{vec}^\top \delta \hat{q}^+_{vec}}{2} \right)}} \right]; \]

or, if \((\delta \hat{q}^+_{vec}^\top \delta \hat{q}^+_{vec}) > 1\)

\[ \delta \hat{q}^+ = \frac{1}{\sqrt{1 + \left( \frac{\delta \hat{q}^+_{vec}^\top \delta \hat{q}^+_{vec}}{2} \right)}} \left[ \begin{array}{c} \delta \hat{q}^+_{vec} \\ 1 \end{array} \right], \]

where \( \hat{q}_{vec} \) denotes the vector portion of the quaternion, \([q_x \ q_y \ q_z]^\top\). Next, we illustrate the measurement update procedure for the laser.

**Laser Measurement Model**

The laser provides a global measurement of the altitude of the vehicle, assuming flight near hover and a flat floor. If these assumptions are violated, we would simply need to produce a different measurement model for the sensor, including, for example, consideration
of both the attitude of the aircraft and a digital elevation map. However, for the scenario considered in this paper, flight near hover and a flat floor are good assumptions. We can thus obtain an estimate of the global altitude using the position in the current state $\hat{p}$ and the global position $\hat{p}_{\text{node}}$ of the current node with respect to which we are navigating. No rotational transformation is necessary as the global down and node down directions are parallel. To compute the Jacobian $H_{\text{las}}$ of the residual with respect to the error state, we must develop an analytical expression for the residual $\Delta h_{\text{las}} = h_{\text{las}} - \hat{h}_{\text{las}}$. We have

$$h_{\text{las}} = -(p(3) + p_{\text{node}}(3))$$
$$\hat{h}_{\text{las}} = -(\hat{p}(3) + \hat{p}_{\text{node}}(3)) - \Delta z_{\text{las}} + b_{\text{las}},$$

where $\Delta z_{\text{las}}$ is the z-offset of the laser from the vehicle center of mass, and $b_{\text{las}}$ is a known laser bias term. Then the analytic residual is

$$\Delta h_{\text{las}} = h_{\text{las}} - \hat{h}_{\text{las}} = -p(3) - p_{\text{node}}(3) + \hat{p}(3) + \hat{p}_{\text{node}}(3) + \Delta z_{\text{las}} - b_{\text{las}}$$

$$= -\delta p(3) - \delta p_{\text{node}}(3) + \Delta z_{\text{las}} - b_{\text{las}}.$$  \hfill (3.46)

The Jacobian $H_{\text{las}}$ of the residual is trivially

$$H_{\text{las}} = \begin{bmatrix} 0 & 0 & -1 & 0_{1 \times 3} & 0_{1 \times 3} & 0_{1 \times 3} & 0_{1 \times 2} & 0_{1 \times 3} & 0_{1 \times 3} \end{bmatrix}$$  \hfill (3.47)

The covariance of the innovation is then found by (3.20) to be $C = R_{\text{las}} + HP - H^T$. This measurement update is used with the fault-detection method described in Section 3.4. Details of the experimentation are given next.

### 3.6 Experimental Results

One of the main focuses of this research was the generation of hardware-based experimental results. These help to illuminate important differences between theory and practice and show that the theoretical results are useful in practice. This section gives details of how the results were obtained and discusses the significance of those results. Section 3.6.1
briefly reviews the experimental setup described in Chapter 2 and gives additional details. Section 3.6.2 then discusses implementation details including deviations from the theoretical fault-detection algorithm described in 3.4 and from the estimator outlined in Section 3.5. Presentation and discussion of the results of these hardware experiments are included in Section 3.6.3. We show that the proposed fault-detection algorithm has been successfully adapted and used to detect faults in a height above ground sensor onboard an autonomous rotorcraft in real time.

3.6.1 Experimental Setup

Hardware tests of the algorithm were performed to test its capabilities. As mentioned, the tests followed the HAG scenario described in Chapter 2. To isolate the sensor of interest, other sensors were assumed to be free from faults. The estimator was run at 100 Hz, the update rate of the IMU. Measurement updates from the laser rangefinder and the visual odometry algorithm were applied at 10 Hz and 15 Hz, respectively.

Software was developed to perform these tests using the ROS (Robot Operating System) [38] framework. Testing consisted of first collecting timestamped IMU, VO, laser, and truth data from the flight computer using ROS as the hexacopter was flown autonomously in an indoor environment. To get a comprehensive view of the detection characteristics of the algorithm, sixteen datasets were collected that are representative of proper system operation, as determined by comparison of fault-free state estimates with truth data. Through the rosbag tool, this data was then played back in real time to the estimator and fault-detection module, as shown in Figure 3.2. Note that the laser datastream passes through a block that introduces faults into the data before reaching the estimator.

3.6.2 Implementation Details

In implementing the fault-detection algorithm in hardware, certain adaptations to both the algorithm and to the estimator were made to ensure proper functionality. This section discusses the modifications that were made to apply the theoretical algorithm to the physical system.
The first change presented here is the tuning of the filter necessary to achieve reliable fault-detection performance. In the process of experimentation, it was discovered that the assumed covariance of the laser altimeter needed adjusting. The original noise characteristics had been obtained through careful experimentation as having a standard deviation of approximately 0.007 m. The use of this value in the estimator, however, caused a very undesirable effect. The assumed sensor noise was so low that at the end of a measurement update, the state estimate of down position was adjusted heavily toward the laser. As a result, bias and drift were detected at most once, after which the estimate followed the faulty value. These detections would then simply be flagged as false-alarms and the state estimates would follow the faulty sensor. This indicated that an increase in perceived measurement uncertainty was in order for the algorithm to function. However, one must be careful not to increase the uncertainty of the sensor to the extent that it has no effect on the estimator,
as this would defeat the purpose of having the sensor at all. It was thus necessary to care-
fully tune the perceived covariance of the laser altimeter measurements to achieve a balance
between these two extremes. The final assumed standard deviation used in the experiments
was 0.04 m. This increase in measurement covariance for a sensor of interest is intuitive in
the sense that we are uncertain whether the sensor is functioning properly. Thus, there is an
increased level of uncertainty about the measurements it is providing. We cannot, however
claim to be so uncertain that we effectively dismiss the sensor. We would be losing valuable
information. We must learn to operate under some level of uncertainty.

The second deviation from the theory presented here is the determination of fault-
detection thresholds for the test of mean and the test of covariance. In our experiments,
a window size of eight measurements was utilized, resulting in theoretical thresholds of ap-
proximately 3.84 for the test of mean and 15.51 for the test of covariance for an allowable
false-alarm rate of five percent. These thresholds were set and the sixteen fault-free datasets
collected for these tests were analyzed. However, these thresholds did not yield satisfactory
results. The threshold for the test of mean was too low, causing an increased number of
false-alarms. The threshold for the test of covariance was much too high resulting in de-
creased sensitivity to faults. The thresholds for each test were tuned using the same fault-free
datasets to yield individual false-alarm rates of approximately five percent. The resulting
thresholds were approximately 4.61 for the test of mean and 1.62 for the test of covariance.
The reason for the large discrepancy in the theoretical and practical thresholds for the test
of covariance is the inflation of assumed sensor noise necessary to detect faults, as discussed
earlier in this section. The assumed sensor covariance was raised to achieve an appropriate
balance of trust in the sensor measurements. This larger covariance increases the covariance
of the residual term. Since the normalized innovation term is inversely proportional to this
covariance, the actual covariance of the innovation sequence will be lower than if no adjust-
ments had been made. Thus, to remain sensitive to faults that change the covariance of the
normalized innovation sequence, the resulting threshold must be lower than the theoretical
value.

Also, though not a deviation from the theoretical fault-detection method, it seems
appropriate to mention here that the threshold for the threshold test was set to six. Thus,
six of eight, or seventy five percent of the measurements in the window being flagged caused the algorithm to declare a sensor fault.

Another point that needs mentioning is that the computation of the residual for use in fault-detection of necessity relies on the covariance of the estimate of the global down position. During the experiments performed for this test, the covariance used in this computation was not the covariance of the estimate of the global down position but instead was the covariance of the estimate of the down position with respect to the most recently defined node. It is hypothesized that the use of the covariance of the global rather than the local down position estimate will yield improved fault-detection results as it matches the theoretical formulation of the algorithm. This is left as future work. We move next to a discussion of the results.

3.6.3 Discussion Of Results

The datasets were first analyzed with no faults injected. Due to uncertainty in system parameters and the tuning of laser uncertainty, it was necessary to tune the thresholds given in (3.25) and (3.28) to achieve the desired false-alarm rate. The window size and threshold for the thresholding test were also tuned so the overall fault-detection rate was approximately five percent. Testing yielded an average false-alarm rate of 5.4%. The resulting thresholds were then used for the remaining tests.

It was determined that the algorithm should detect bias, drift, and increased sensor noise within a 20 second interval from the time of fault inception. Varied magnitudes of these faults were injected to show the severity of faults that can be reliably detected. The detection characteristics of the algorithm were then evaluated by computing detection rate within the given window and delay to detection for each test. Detection rate is given by

\[
r_d = \frac{n_d}{n_f},
\]
where \( n_d \) is the number of detections and \( n_f \) is the number of faulty measurements, both within the window of interest. Delay in detection was simply calculated as

\[
d = t_d - t_i,
\]

where \( t_d \) is the time of detection and \( t_i \) is the time of fault inception. Averages of these quantities for each fault are shown in Tables 3.1 - 3.3.

**Table 3.1: Bias Results**

<table>
<thead>
<tr>
<th>Magnitude (m)</th>
<th>Detection Rate (20 s)</th>
<th>Detection Rate</th>
<th>Detection Delay (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>9.1%</td>
<td>5.2%</td>
<td>8.43</td>
</tr>
<tr>
<td>0.10</td>
<td>36.6%</td>
<td>13.7%</td>
<td>0.50</td>
</tr>
<tr>
<td>0.15</td>
<td>59.2%</td>
<td>22.5%</td>
<td>0.50</td>
</tr>
<tr>
<td>0.20</td>
<td>69.0%</td>
<td>27.8%</td>
<td>0.50</td>
</tr>
<tr>
<td>0.25</td>
<td>77.5%</td>
<td>32.9%</td>
<td>0.50</td>
</tr>
</tbody>
</table>

**Table 3.2: Drift Results**

<table>
<thead>
<tr>
<th>Magnitude (m/s)</th>
<th>Detection Rate (20 s)</th>
<th>Detection Rate</th>
<th>Detection Delay (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.44%</td>
<td>5.5%</td>
<td>37.88</td>
</tr>
<tr>
<td>0.005</td>
<td>9.8%</td>
<td>36.8%</td>
<td>14.99</td>
</tr>
<tr>
<td>0.010</td>
<td>45.8%</td>
<td>73.6%</td>
<td>8.63</td>
</tr>
<tr>
<td>0.015</td>
<td>68.8%</td>
<td>88.1%</td>
<td>5.18</td>
</tr>
<tr>
<td>0.020</td>
<td>77.0%</td>
<td>92.1%</td>
<td>4.02</td>
</tr>
</tbody>
</table>

**Table 3.3: Increased Noise Results**

<table>
<thead>
<tr>
<th>Magnitude (m)</th>
<th>Detection Rate (20 s)</th>
<th>Detection Rate</th>
<th>Detection Delay (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005</td>
<td>0.28%</td>
<td>3.9%</td>
<td>34.07</td>
</tr>
<tr>
<td>0.015</td>
<td>6.4%</td>
<td>9.9%</td>
<td>1.26</td>
</tr>
<tr>
<td>0.025</td>
<td>51.1%</td>
<td>54.3%</td>
<td>0.96</td>
</tr>
<tr>
<td>0.035</td>
<td>86.0%</td>
<td>88.1%</td>
<td>0.77</td>
</tr>
<tr>
<td>0.045</td>
<td>94.9%</td>
<td>96.8%</td>
<td>0.65</td>
</tr>
</tbody>
</table>
Figure 3.3: Results of fault-detection for the case of a 0.25 m bias. The uppermost plot shows the truth, estimate, and laser-based estimates together. The two center plots show the values of the mean statistic and covariance statistic, respectively, with their thresholds. The bottom plot shows the flag indicating sensor health. On each plot, the vertical dashed line indicates the time of fault inception. Note the quick but fleeting fault flag for this type of fault.

These tables contain the detection rates for the 20 second window after fault inception and also for the entire dataset after fault inception. Comparison of these values indicates that the algorithm is very quick to detect bias, though the detection rate lessens with time (see Figure 3.3). This is due to the static nature of the fault. Because the faulty data is still
used to update the estimator, estimates converge to the biased state, after which the fault is no longer detected. Table 3.1 also indicates that bias is quickly detected in most cases. This fast detection is necessary for bias because the fault is not detected for very long. The highest level of bias tested is only 6.25% of the maximum sensor range of 4 m, indicating good detection characteristics.

The results for drift show an increase in detection rate and decrease in detection time as fault magnitude is increased, as expected. Unlike in the case of bias, however, the detection rate is higher when we consider detections after the prescribed 20 second window. The average values for delay to detection indicate that this is because drift takes longer to detect drift than bias, however, the estimator does not accommodate drift, so the fault is more continually detected after the initial detection. An example of drift fault-detection results is shown in Figure 3.4. The highest magnitude drift tested represents a change of 0.5% of maximum sensor range per second, also indicating good detection.

The test for increased sensor noise also showed an increase in detection rate and a decrease in detection time as fault magnitude increased. Because of the abrupt nature of noisy measurements, the detection rates for the 20 second window and the remainder of the dataset do not differ to a large degree. However, being a non-static fault, the estimator does not converge to the faulty state, thus maintaining good detection characteristics for the duration of the fault. See Figure 3.5 for a sample results plot depicting algorithm performance during the onset of an increase in sensor noise. Maximum fault magnitude for this test was additive white noise with a standard deviation of less than two percent of the maximum sensor range, which also seems very good.

These tests were performed offline. To verify that this implementation works onboard during flight, tests were performed with the estimator running during flight. Detection results were streamed via Wi-Fi to a monitor for visual inspection. Results were comparable to the tests where the estimator was run on replayed data. See Figure 3.6 for photos of the hardware tests.
Figure 3.4: Results of fault-detection for the case of a 0.02 m/s drift. The uppermost plot shows the truth, estimate, and laser-based estimates together. The two center plots show the values of the mean statistic and covariance statistic, respectively, with their thresholds. The bottom plot shows the flag indicating sensor health. On each plot, the vertical dashed line indicates the time of fault inception. Note the delay in detection characteristic of drift and the persistence of fault-detection, as contrasted with the bias detection results.
Figure 3.5: Results of fault-detection for the case of 0.035 m standard deviation additive white noise. The uppermost plot shows the truth, estimate, and laser-based estimates together. The two center plots show the values of the mean statistic and covariance statistic, respectively, with their thresholds. The bottom plot shows the flag indicating sensor health. On each plot, the vertical dashed line indicates the time of fault inception. Note the quick and persistent fault-detection associated with this type of fault.
Figure 3.6: Hardware Test Photos. The rotorcraft took off and hovered above the ground before moving over a raised platform to hover. It then moved off the platform to hover above the ground again. The monitor near the bottom of the photos shows the fault-detection status at a given time.

3.7 Conclusion

As demonstrated, the fault-detection algorithm described in Section 3.4, coupled with the robust estimator described in Section 3.5 is capable of detecting bias, drift, and increased noise of a non-redundant sensor real time in hardware. This is accomplished by establishing analytical redundancy using an accurate system model.

Future work should include studying the effects of changing window sizes for each of the three tests and tuning the threshold for the thresholding test. A greater understanding of the fault-detection capabilities could also be gained by increasing the levels of faults tested. In addition, the successful integration of information from the different tests to determine a level of confidence in declaring a fault should be examined. Also, as mentioned in Section 3.6.2, it is hypothesized that the use of the covariance of the global rather than the local down position estimate will yield improved fault-detection results as it matches the theoretical formulation of the algorithm. This should be explored. Finally, another area that deserves future work is the development of theoretical limits of performance for this method.
CHAPTER 4. PARTICLE FILTER METHOD

One alternative to the EKF for estimation involving nonlinear systems is the particle filter. Particle filtering is quickly becoming more popular for localization and other estimation tasks related to robotics. This is due to its robustness and ability to model complex distributions. These desirable properties led to this investigation of the potential of using a particle filtering technique to detect faults in non-redundant sensors. Section 4.1 gives the motivation behind the use of particle filters in general and for fault-detection specifically. Next, Section 4.2 presents the particle filter algorithm, with an example following in Section 4.3. Section 4.4 discusses some drawbacks to using particle filters, while remedial methods to those problems are outlined in Section 4.5. The particle filter based fault-detection method used for this investigation is detailed in Section 4.6. Details about the setup of the simulation experiments is given next in Section 4.7. The results of those experiments are then presented in Section 4.8. This chapter then concludes with Section 4.9, which summarizes the results and discusses recommended future work.

4.1 Motivation

The fault-detection algorithm described in Chapter 3 relies on the use of system models to determine whether a sensor is likely to yield particular measurements. Specifically, the normalized residual of a Kalman filter is used to detect and identify sensor faults. One weakness of Kalman filters\footnote{In this chapter, the use of the term Kalman filter will typically refer to both the standard Kalman filter and the extended Kalman filter.} is that they use a parameterized representation of the state estimate based upon the mean and covariance. This can yield poor results for a system with noise that is not easily parameterized (e.g., non-Gaussian). An additional drawback is that the standard Kalman filter is incapable of dealing with system nonlinearities explicitly. An EKF partially solves this problem by using the nonlinear update equations to propagate
the mean of the estimate and using the linearization of those equations to propagate the estimate’s covariance. This, however, introduces some linearization error and can cause the system to become inconsistent. According to [39], this can also lead to an increase in the false-alarm rate and a reduction in detection performance for a fault-detection method based on the EKF.

A particle filter, like a Kalman filter, is a recursive Bayesian filter. Unlike a Kalman filter, however, a particle filter is nonparametric. Whereas a Kalman filter parameterizes a distribution by both a mean vector and a covariance matrix, a particle filter describes a distribution by a set of random samples drawn from the distribution. Figure 4.1 shows the two representations of a Gaussian distribution. The benefits of using a particle representation are discussed below.

![Figure 4.1: Parametric and Particle Representations of a Gaussian Distribution](image)

The particles in a particle filter are described in [40] to be hypotheses of the true state at time $t$. Through the filtering algorithm, these hypotheses are evaluated and the set of hypotheses adapts to inputs and new information (measurements). This same principle underlies Kalman filtering. Let us now discuss some of the benefits of particle filters that are, in large part, due to their nonparametric representation of distributions.

Because a particle filter is nonparametric in nature, it can operate on a wide range of distributions. This enables the implementation of particle filters for a large class of problems for which a Kalman filter would be ill-suited, as Kalman filters are only capable of describ-
ing Gaussian distributions. The particle representation of the probability density function (pdf) affords a second advantage. It allows the filter to handle nonlinear transformations of probability distributions. Kalman filters cannot fully deal with nonlinear update equations because these transformations distort the shape of the distribution and the Gaussian assumption no longer holds. As discussed previously, an EKF attempts to solve this problem but does not account for the distortion of the distribution due to nonlinear transformations. This can cause the filter to become inconsistent and diverge. Another benefit of particle filters is their ability to track multiple hypotheses with ease. This is again due to the simplicity of representing arbitrary distributions. This allows the particle filter to solve global localization problems, one of its advantages when compared to Kalman filters.

The ability of the particle filter to describe arbitrary probability distributions, including those resulting from nonlinear transformations of Gaussian distributions, allows the particle filter to more accurately represent the state of the system at a given time. In Chapter 3, the importance of an accurate model for a fault-detection algorithm is highlighted. Thus, it is hypothesized that a particle filter based method of fault-detection will yield better detection characteristics than an EKF based method for nonlinear systems. The particle filter algorithm is described next.

4.2 The Algorithm

This section is intended to be an accessible introduction to particle filters. For those interested, more rigorous introductions can be found in [40–42]. The standard particle filter algorithm comprises the following steps:

1. initialization of the belief distribution
2. model-based prediction
3. measurement update
4. extraction of an estimate (optional)
5. resampling.
Each of these steps will be discussed in detail in the following sections. An example will then be given which illustrates the use of the algorithm.

4.2.1 Initialization

The initial probability distribution of the system state, \( \mathbf{x} \), is approximated by sampling \( M \) times from \( p(\mathbf{x}_0) \), denoted

\[
\mathbf{x}_0^{[m]} \sim p(\mathbf{x}_0), \quad m \in \{0 \ldots M\}
\]

where \( M \) is the desired number of particles, which is determined by the needs of the application, and \( p(\mathbf{x}_0) \) is the initial probability distribution. This requires only a way to sample from the desired initial probability density function.

For a Kalman filter, initialization of the probability distribution is constrained because a Gaussian distribution is used for its representation. A particle filter, as mentioned above, is not subject to that constraint. For example, consider a localization problem in which the only initial information available is that the UAV is located within the occupiable space of a given map. One would then want to initialize belief to be uniformly distributed throughout occupiable space. In another case, perhaps information suggest greater probability of being located near a corner. With a particle filter, it would be simple to place greater numbers of particles near corners and fewer particles farther from corners. Neither the uniform distribution nor the complex multi-modal distribution representing higher probability near corners could be handled with a simple Kalman filter. In contrast to the above scenarios, sometimes, a much more accurate idea of the UAV’s initial position is available. This situation is ideal for a Kalman filter. A particle filter can be used in this situation as well. As can be seen, a particle filter adds a great deal of flexibility in initializing the probability distribution.

4.2.2 Prediction

The prediction step propagates particles from the previous time step to the current time step using knowledge of the system model and the inputs to the system. In this
step, samples are drawn from the distribution of current possible states conditioned on both the previous state represented by that particle and the inputs at the previous time step. Mathematically, this is written

$$\bar{x}^{[m]}_{k} \sim p(\bar{x}^{[m]}_{k}, u_{k-1}),$$

where $u$ represents the vector of inputs. Sampling from this distribution is equivalent to computing Equation (4.1) for each of the $M$ particles:

$$\bar{x}^{[m]}_{k} = f(\bar{x}^{[m]}_{k-1}, u_{k-1}, q^{[m]}_{k-1}),$$

where $f(\bar{x}^{[m]}_{k-1}, u_{t-1}, q^{[m]}_{k-1})$ is the nonlinear time update equation for the system and $q^{[m]}_{k-1}$ is a random sample drawn from $Q_{k-1}$, which is the pdf of the process noise, potentially modified to promote particle diversity. The resulting set of particles represents the a priori (prior to a measurement update) probability distribution of the UAV’s state. This is then updated by considering received measurements.

4.2.3 Measurement Update

The measurement update step of a particle filter includes the computation of importance factors or weights for each particle. A particle’s weight is proportional to the probability of receiving the current measurement given the hypothesized state the particle represents, $\bar{x}^{[m]}_{k}$. The normalized weight, $w_k$, is calculated for each particle as

$$w_k^{[m]} = \frac{p(z_k|\bar{x}^{[m]}_{k})}{\sum_{m=1}^{M} p(z_k|\bar{x}^{[m]}_{k})},$$

where $z_k$ is the measurement received at time $k$. The determination of $p(z_k|\bar{x}^{[m]}_{k})$ is accomplished by using the measurement equation to find the pdf of the predicted measurement given the current particle, $x_k$. This pdf describes the probabilities associated with different measurement values. The likelihood of receiving the actual measured value is then obtained by evaluation of the pdf at that value. A low weight indicates that the particle represents
a hypothesis that does not explain the current measurement. This suggests either a poor estimate or a faulty sensor, as in the Kalman filter.

In some cases, multiple sensors are used during the same measurement update step of the filter. In this case, the weights must be computed using the joint probability distribution

\[ w_k[m] = p(z_{k_1}, z_{k_2}, \ldots, z_{k_s}|x_k[m]) \quad (4.2) \]

where \( s \) is the number of sensors reporting measurements during the current filter cycle. Often, however, it can be assumed that the sensors are conditionally independent of one another. This simplifies Equation (4.2) to

\[ w_k[m] = p(z_{k_1}|x_k[m])p(z_{k_2}|x_k[m])\ldots p(z_{k_s}|x_k[m]), \]

which can also be written as

\[ w_k[m] = w_{k_1}[m]w_{k_2}[m]\ldots w_{k_s}[m]. \]

Thus, one computes particle weights for an individual measurement source and incrementally updates the particle weights by simple multiplication. To ensure this set of weighted particles is a valid probability mass function (pmf), the weights are again normalized. The resulting set of weighted particles represents the a posteriori (after measurement update) probability distribution of the system state.

4.2.4 Extraction of Estimate

The application to which the particle filter is applied determines whether it is necessary to extract a point estimate from the resulting posterior distribution. Often, as in the case of an estimator used for feedback control, it is important that a point estimate of the state be obtained. According to [42], the three most common methods of extracting an estimate from the weighted set of particles are:

1. best particle
2. weighted mean
In the first approach, the particle with the highest weight is chosen as the estimate of the state at time $k$. The second approach uses the weighted mean of all particles as the estimated state. The third method is a hybrid of the two previous methods. It uses the weighted mean of particles near the particle with highest weight as the estimate. Each of these has advantages and disadvantages. The best particle approach is robust when multiple modes are present, however, it also introduces a discretization error. This problem increases as the number of particles used to represent distributions decreases or as the spread of the particles increases (as we become less certain of the state or as additional dimensions are added to the state space). The weighted mean approach reduces discretization error, as the estimate does not have to be co-located with any of the particles. However, this method will usually result in poor estimates if the distribution we are considering is multi-modal, as shown in Figure 4.2. The dashed line in this figure represents the weighted mean for this distribution. Notice that the estimate lies in an area of very low probability. The most accurate and robust approach is the third, where a weighted mean is used to avoid discretization error and a small window size eliminates the effect of particles far from the most prominent mode. The drawback to this method is the computational complexity of finding which particles of...
the entire set are near the particle of highest weight, as distances of each particle from the particle of highest weight must be computed.

4.2.5 Resampling

The resampling step for a particle filter is technically optional as well, though it is nearly always included. Perhaps the best motivator for including it is that it places more particles in areas of greater probability. This improves resolution in these areas and reduces the number of particles in areas of low probability. If this step is not performed, many more particles are typically needed for the algorithm to produce suitable results. The most basic type of resampling draws with replacement $M$ particles from the a posteriori probability distribution with probability proportional to their weights. Other resampling methods have been developed and a number of these are discussed in [42].

The basic particle filter algorithm is summarized in Algorithm 2.

4.3 Example

In this section, an example is given that illustrates the operation of the particle filter. This example and explanation is adapted from a similar example in [40]. Let us assume that a UAV is operating in a GPS-denied environment and must rely on road recognition for localization. Suppose further that the vehicle has a map of its environment, as seen in Figure 4.3, and is able to detect both position and orientation of roads with a certain degree of accuracy. Assume also that the heading of the vehicle is known, but that the only initial location information available is that the vehicle is located above the main road shown in the map.

As discussed in Section 4.2, the first step is to initialize the probability distribution representing the vehicle’s belief about its position along the road and its orientation. Because of the global uncertainty in initial position, this distribution will be initialized by sampling $M$ times from a uniform distribution over the entire state space. The resulting particle set is shown in Figure 4.3.
Algorithm 2 Standard Particle Filter Algorithm

1: procedure Particle Filter
2:     procedure Initialization
3:         \( k = 0 \)
4:     for \( m = 1 : M \) do
5:         \( \bar{x}_0[m] \sim p(x_0) \) % Sample from initial distribution
6:         \( w_0[m] = \frac{1}{M} \) % Initialize weights uniformly
7:     while True do % until no more data is coming in
8:         \( k = k + 1 \)
9:         procedure Time Update(\( x_{k-1}, u_{k-1}, q_{k-1} \))
10:            for \( m = 1 : M \) do
11:                \( q_{k-1}[m] \sim p(q_{k-1}) \) % draw samples from process noise distribution
12:                \( \bar{x}_k[m] = f(\bar{x}_{k-1}[m], u_{k-1}, q_{k-1}[m]) \) % propagate particles forward in time
13:     if \( z_k \neq \text{None} \) then
14:         procedure Measurement Update(\( x_k, z_k \))
15:             \( \bar{w}_k[m] = \frac{p(z_k|x_k[m])}{\sum_{m=1}^{M} p(z_k|x_k[m])} \) % compute normalized particle weights
16:     if desired then
17:         procedure Extract Estimate(\( x_k, w_k \))
18:             if method == bestParticle then
19:                 \( \hat{x}_k = \bar{x}_k[m] \) s.t. max(\( w_k \)) = \( w_k[m] \)
20:             else if method == weightedMean then
21:                 \( \hat{x}_k = \sum_{m=1}^{M} \bar{x}_k[m] w_k[m] \)
22:     if desired then
23:         procedure Resample(\( x_k, w_k \))
24:             generate new particle set by sampling \( x_k \) with \( p(x_k[m]) \propto w_k[m] \) (with replacement)
Figure 4.3: Initialization - Under global uncertainty, the distribution for UAV position, $x$, is initialized by drawing $M$ particles from a uniform distribution over the state-space. The true location of the vehicle is shown above the particle set.

Assume that at this initial time, the UAV then receives a measurement from its road sensor indicating that it is over an intersection with a road perpendicular to the road along which it is traveling. The particle sets and probability distributions associated with this step are shown in Figure 4.4. The first plot again shows the a priori distribution obtained through the initialization step. The second plot shows the conditional probability of detecting a perpendicular intersection given the vehicle’s location. This pdf is used to assign weights to the particles to obtain the weighted particle set that is shown in the third plot. Because the vehicle detected a perpendicular intersection and there are four such intersections in the map, these each receive equal probability (we assume the roads are otherwise indistinguishable). The non-perpendicular intersecting road is also assigned a small probability due to the possibility of misclassification of intersection angle. Also, notice that a uniform distribution underlies the distribution, $p(z|x)$. This is due to the small possibility of falsely detecting the presence of a road where none exists. This also represents the weights that will be assigned to the particles, as can be seen by comparing the first two plots to the third plot in Figure 4.4. Notice that the locations of particles in the set have not changed. The particles in the third plot are simply scaled versions of the a priori particle set. The particle weights are proportional to $p(z|x)$ and sum to one to create a valid pmf. This weighted particle set represents the posterior probability distribution for vehicle position.

Then, to focus particles in areas of higher probability, a resampling step is applied. Figure 4.5 depicts this process. $M$ particles are sampled with replacement from the weighted
Figure 4.4: Measurement Update- The first particle set represents the prior probability distribution. \( p(z|x) \) is the probability of detecting a perpendicular intersection given the position, \( x \), of the vehicle. The second particle set shows the a posteriori probability distribution particle set, the probability of selecting each particle being equal to its weight. This yields a new set of particles, all with equal weights.

Figure 4.5: Resampling- A new particle set is obtained by sampling the weighted particle set obtained from the measurement update.
Next, the prediction step takes place. This step propagates the particles forward in time using the motion model, including the addition of appropriate process noise to encourage particle diversity and to account for unmodeled behavior. This step is depicted in Figure 4.6.

The entire process is then repeated for a second iteration. This new prior distribution is again updated through a measurement update (see Figure 4.7). At the time of this measurement update, the vehicle believes with a high degree of confidence that it is above an intersection with a road that is not perpendicular to the road along which it is traveling. Thus, a high portion of the conditional probability distribution is centered over the only non-perpendicular intersection in the map. Small amounts of probability are assigned at the other intersections to account for the possibility of misclassification of intersection angle. Because a large amount of the probability density from the conditional distribution is centered near one location, the particle weights are very high in that narrow band and low elsewhere. Note that because a large number of particles were already located in this region, this region represents the most likely location of the UAV. This causes the resampling step (see Figure 4.8) to cluster...
Figure 4.7: Measurement Update - This measurement indicates with high probability that the vehicle is over an intersection with a road that is not perpendicular to the road it is following. Thus, highest probability is placed near the only such road in the map.

a large portion of the new set of particles in this region. This encourages better resolution in this area, which has a higher likelihood. Notice that the majority of resampled particles are now co-located with the vehicle, suggesting that we are converging on a correct estimate of the vehicle’s location along the road.

From this example, it can be seen that the particle filter algorithm can easily handle non-Gaussian distributions. It is a wonderful tool in cases such as this. This characteristic allows the particle filter to track multiple hypotheses robustly without needing to propagate multiple filters in parallel. Notice that in this example, we did not extract an estimate of the UAV’s location. This is one scenario where it was not necessary to extract an estimate because no feedback was utilized. However, estimates could have been extracted if necessary. The highest-weight particle method probably would have best fit this scenario initially, due to the multi-modal nature of the a posteriori distribution, though perhaps a switch to a weighted mean method could have been adopted when the distribution became more unimodal in nature.
4.4 Drawbacks

Particle filters are able to do things that Kalman filters cannot do. They have some very nice properties due to their nonparametric nature. However, they do have some limitations which are important to understand. One of the drawbacks to particle filters is that they only approximate the true posterior distribution. In fact, a particle filter only approaches the actual posterior asymptotically as $M$ approaches infinity. Thus, the particle representation may not faithfully represent the true probability distribution. An inaccurate view of the probability distribution could cause the filter to diverge or cause other problems. Thus, a particle filter requires $M$ to be sufficiently large to adequately represent the probability distribution. Another drawback to particle filters is their computational cost. While the computational complexity of a Gaussian filter scales between linearly and quadratically in the number of dimensions, the computational complexity of a particle filter scales exponentially [40]. This makes it difficult to use a traditional particle filter for systems with high-dimensional state spaces. Another potential problem with particle filters is a loss of particle diversity. If resampling is employed, as it usually is, there is the potential that, over time, the filter could end up making $M$ copies of the same particle. This is an indication that the filter has become overconfident. This leads to filter divergence. Understanding of
these pitfalls and potential drawbacks has spurred the development of methods and techniques to remedy these problems so that particle filters can be adapted to a greater number of systems and can be implemented in real time. In the next section, methods commonly used to remedy these problems are presented and discussed.

4.5 Remedial Methods

First, I will address some of the methods used to encourage particle diversity. The first is the inclusion of additional noise in the time update, Equation (4.1). This helps to account for inaccuracies in the system model and helps spread the resampled particles out (after resampling, there are often multiple copies of the same particle). Another method is to inflate the algorithm’s sense of measurement noise when calculating particle weights. This will help to spread the weight over more particles. Thus, a higher percentage of the particles will be selected during resampling. As discussed in [40], a noiseless sensor, ideal in Kalman filters, would cause a particle filter to diverge, as particle diversity would quickly be lost. This would cause the posterior to have all probability centered at a single state. This can happen even if the sensor is modeled as having a small non-zero noise. A filter is especially susceptible to this if the number of particles in the vicinity of the measurement is too small. This again leads to a small subset of the particles receiving more weight than the majority of other particles. Another method that encourages particle diversity is the injection of random particles during the resampling step. Rather than drawing $M$ particles from the posterior, one would randomly generate some fraction of the $M$ particles according to some distribution (dictated by the application). The rest would then be resampled in the desired manner. This approach allows for there to always be some diversity in the particles regardless of how narrow the sensor’s assumed distribution is made. This technique is often used in the context of localization to recover from severe localization failures (the kidnapped robot problem). Another method that can be employed is described in [40]. It is presented in the context of localization and is called Mixture Monte Carlo Localization (Mixture MCL). The basic idea is to switch the roles of the measurement and prediction models for a small portion of particles. Thus, one would propagate a portion of the particles forward with the measurement model and then calculate their weights using the prediction model. The other
particles would be treated normally. This ensures that there is a greater number of particles in the vicinity of the most recent measurement. This method, though, can be difficult to implement.

The ability to more accurately represent the true state distribution using a particle set requires an increase in the number of particles used in the representation. However, the use of a larger number of particles exacerbates the other problem of computation time. For real-time implementation, then, $M$ can only be made so large before the computational burden becomes too great. There are a myriad of methods used to increase the efficiency of a particle filter so that more particles can be used, though there are limits as to how much of a performance gain can be achieved through these methods.

One method of reducing required computation time is to reduce the number of particles. This can be done by fixing $M$ at a smaller value, or it can be done by allowing $M$ to change to meet the speed of the system. This latter method is referred to as making the algorithm resource adaptive. This causes the algorithm to continue creating particles until a new measurement is received. Thus, $M$ will be as large as possible while still running in real-time (as determined by the rate of arrival of sensor measurements). The problem with these methods, however, is that estimates suffer as the number of particles is decreased. It may be that it is not possible to generate enough particles to get good estimates in the given time. In cases which have large state spaces or require high update rates, a variant of the standard particle filter, called a Rao-Blackwellised particle filter (RBPF), is often very useful.

The Rao-Blackwellised particle filter relies on the Rao-Blackwell theorem ([43]) to reduce the size of the state space. It does this by marginalizing out some of the state variables using an optimal filter such as a Kalman filter [44]. This then requires sampling of only a subset of the state space using a particle filter, making the filter much more efficient. A detailed explanation of RBPF is provided in [44]. One of the major benefits to this variation on the standard particle filter is that we can typically achieve better results than we could with a particle filter and with many fewer particles. This method, however, cannot always be used (see [44] for details).
4.6 Fault Detection

The purpose of this work was to evaluate a particle filter based fault-detection algorithm for a non-redundant sensor through comparison with the EKF based algorithm of Chapter 3. A number of researchers have developed fault-detection and identification methods using particle filters. Most of these include the use of multiple models in conjunction with the particle filter to determine which of a set of known faults has occurred (see [45–50]). This ability is potentially useful in many situations, however the scope of the current work included only detection of faults and not their identification. This facilitates a comparison with the EKF based fault-detection scheme, which also considers only detection. The researchers in [39] detail a particle filter based approach to fault-detection that closely parallels the EKF algorithm described in Chapter 3. They show that their approach produces a higher detection rate along with a lower false-alarm rate and lower time to detection than an EKF based fault-detection method for a given nonlinear system. These promising results prompted the adoption of their method for our system of interest.

The proposed fault-detection method in [39] uses a particle filter to predict the current state as described in Section 4.2. However, when a measurement is received, the decision variable

$$d_k = \sum_{j=k-L+1}^{k} \ln \left( \frac{1}{M} \sum_{i=1}^{M} \tilde{w}_i \right)$$

(4.3)

is computed, where $L$ is the desired sliding window length and $\tilde{w}_i$ represents the un-normalized weight, $p(z_k|x_k(i))$, of the $i$-th particle. A fault decision is reached by comparison with a threshold as

$$d_k \geq \frac{D_0}{D_1} h,$$

(4.4)

where $h$ is determined experimentally to achieve good balance between detection rate and false-alarm rate.

To maintain similarity with the EKF based method described in Chapter 3, the threshold test described in Section 3.4.3 was added to this proposed particle filter method as well. This reduces the number of false-alarms, as the sensor is declared faulty only when a large percentage of the most recent measurements are flagged. A window of size $W$ is used.
for this test. Letting $n$ represent the number of faults within the current window, we pick some threshold, $T < W$, such that the test

$$\frac{n_{D_1}}{n_{D_0}} \geq T$$

yields a decision about the health of the sensor. Here, $D_0$ represents failure to declare a fault and $D_1$ represents the declaration of a fault. Window size for this test can be chosen independently of the window size for the previous test. Choosing a larger window will delay fault-detection and sometimes mask legitimate fault declarations, whereas too small a window will not appreciably reduce the false-alarm rate. Again, the operator must use discretion to choose an appropriate window size and threshold for this test. These quantities will vary from system to system and should be selected in light of the considerations above.

The similar method was shown in [39] to produce better results than an EKF based method like the one described in Chapter 3 for a highly nonlinear system. To test whether this method would yield improved detection performance for our proposed scenario, simulations were performed to compare the two methods of fault-detection.

### 4.7 Simulation Setup

This section describes the setup of simulations that were performed to compare the fault-detection performance of the EKF based method described in Chapter 3 with the particle filter based method described in Section 4.6. The scenario that was used for this comparison and the reasons that this differs from the scenario used to test the EKF based algorithm are discussed in Section 4.7.1. Note that the comparison of the filters is performed on this new scenario and does not involve the system described in Chapter 3. The testing procedure is then described in Section 4.7.2.

#### 4.7.1 Scenario

Due to the high dimensionality of the state space of the scenario described in Chapter 2, the particle filter based method was impractical, as the required number of particles to adequately cover the space was prohibitively large. Rao-Blackwellisation was attempted
in an effort to reduce the number of required particles, however the structure of the system
did not admit such a solution, as certain required assumptions of independence among states
were violated. Because it was not possible to implement a particle filter on the original sys-
tem, a system was created with a smaller state space that would be amenable to a particle
filter.

To maintain similarity to the original scenario, this new scenario also involves the
detection of faults in a height-above-ground (HAG) sensor onboard an autonomous aircraft.
In contrast with the dynamic model used in Chapter 3, however, this scenario implements
a kinematic model. The states in the model are position north, position down, pitch, and
speed. The aircraft is assumed to be moving in the north-down plane only. Inputs to the
state update equation include a pitch gyro measurement and an accelerometer measurement
along the direction of travel (i.e. it is unaffected by gravity). Measurement updates included
a camera update, with the simulated camera measuring pitch and speed of the aircraft as
would be reported by an optical flow algorithm. Updates also included the HAG sensor, the
sensor of interest for the purposes of fault-detection. The state update equations are given
by
\[
\begin{bmatrix}
    p_n \\
p_d \\
\theta \\
s_k
\end{bmatrix}
= \begin{bmatrix}
p_n \\
p_d \\
\theta \\
s_{k-1}
\end{bmatrix}
+ \Delta t \begin{bmatrix}
a(s_{k-1})\cos(\theta_{k-1}) \\
-b(s_{k-1})\sin(\theta_{k-1}) + c \\
d(p_{gyro_{k-1}}) \\
e(a_{k-1})
\end{bmatrix}
\]

where \(a, b, c, d, \) and \(e\) are unknown system parameters. In generating datasets, these held
the values \(a = b = d = e = 1\) and \(c = 0\). However, to simulate an imperfect knowledge of
the system model, both the EKF and the particle filter used values which deviated slightly
from these nominal values. The camera measurement functions are given by
\[
h_{cam} = \begin{bmatrix}
\theta \\
s
\end{bmatrix},
\]
and the measurement equation for the HAG sensor is given by
\[
h_{HAG} = -p_d\cos(\theta).
\]
4.7.2 Test Procedure

Monte Carlo simulations were performed to ascertain the operating characteristics of both the EKF and the PF fault-detection algorithms. Truth data remained the same for each iteration. However, the simulated sensor measurements were populated by sampling the truth data at pre-defined sensor update rates and by adding a random amount of noise each time.

Due to the theoretical similarities between the EKF and the particle filter, the various filter parameters (e.g. process noise, measurement noise, etc.) were kept as similar as possible. This was largely achieved, with only minor differences. Additionally, the particle filter used five hundred particles for its representation of the state estimate.

The performance of the filters was evaluated through one hundred iterations of the power set of tests formed by considering: fault type, fault magnitude, and threshold levels. The fault types considered were: no fault, bias, drift, and increased noise. Each of these faults was evaluated at different levels, which were determined through initial experimentation. This determination came by finding a range of magnitudes that showed distinct performance between the low and high magnitudes. Three detection thresholds (or sets thereof in the case of the EKF based method) were tested for each algorithm, with the center threshold chosen to give a false-alarm rate as close to five percent as possible for the no fault case. The other two thresholds were chosen on either side to produce approximately three and eight percent false-alarm rates for the no fault case, respectively. Because the particle filter employs only one threshold, the thresholds for the EKF tests of mean and covariance were grouped into low, medium, and high pairs to maintain similar test conditions.

Three performance metrics were measured for each Monte Carlo iteration: false-alarm rate, detection rate, and delay to detection. False alarm rate was computed as the number of faults reported before the fault onset time divided by the total number of measurements occurring before that time. Detection rate was calculated similarly as the ratio of faults reported after the onset of the fault to the total number of measurements received after that time. Delay to detection was measured in seconds as the total time between fault inception and the first detection.
4.8 Simulation Results

As mentioned in Section 4.7, one hundred Monte Carlo simulations were performed for the power set of fault type, fault magnitude, and threshold values for both the EKF and the particle filter. Measured values for each of these tests included false-alarm rate, detection rate, and delay to detection. The run time of the filter was also recorded. The statistics of each of these measured quantities were computed for each of the input parameters and were plotted for comparison. Each plot displays the mean and covariance of a given metric by the use of error bars that extend two standard deviations from either side of the mean.

Figure 4.9 shows the resulting false-alarm rates for the case of no fault using both the particle filter and the EKF at each of their three threshold levels. This plot shows that there is no significant difference between the false-alarm rates of any of these configurations. They are also all centered very near the desired five percent level. This indicates that the probability of detection for the two methods can be compared reliably, as one does not have a higher detection rate at the expense of producing a greater number of false-alarms.

\[ \text{False Alarm Rates} \]

![False Alarm Rates](image)

Figure 4.9: False alarm rates for the particle filter and the EKF at their high, medium, and low threshold values. Note that they are not significantly different.

A comparison of the filters’ abilities to detect bias are shown next, in Figure 4.10. In
Figure 4.10: Test results for bias. The upper plot shows probability of detection versus fault magnitude. The lower plot shows delay to detection versus fault magnitude. In both metrics, the EKF outperforms the particle filter.

In the upper plot, we see that while neither detection method detects low levels of bias reliably, the Kalman filter consistently detects bias at the higher levels, whereas the particle filter does not. The lower plot shows that in addition to the difference in probability of detection, the particle filter method is slower (on average) than the EKF method at detecting bias, at
least for small levels. Note also the high variability in the PF method’s results. This is an undesirable characteristic for a fault-detection scheme.

Next, Figure 4.11 shows the results of using both methods to detect drift in the sensor of interest. In the upper plot of Figure 4.11, one can see that the EKF method again

![Drift Results](image)

Figure 4.11: Test results for drift. The upper plot shows probability of detection versus fault magnitude. The lower plot shows delay to detection versus fault magnitude. In both metrics, the EKF outperforms the particle filter.
outperforms the particle filter method in terms of probability of detection. There is also a
difference in delay to detection, as shown in the lower plot.

Finally, Figure 4.12 shows the results of the testing for the detection of increased
sensor noise. In the upper plot of Figure 4.12, one sees that both methods achieve similarly

![Noise Results](image)

Figure 4.12: Test results for increased noise. The upper plot shows probability of detection
versus fault magnitude. The lower plot shows delay to detection versus fault magnitude. In
both metrics, the EKF again outperforms the particle filter.
poor performance for small fault magnitudes. The detection rate for the EKF based method, however, rises rapidly with fault magnitude, while the detection rate for the particle filter based method rises slowly. It should be noted that the particle filter based method only achieves near perfect detection rates for faults an entire order of magnitude larger than those for which the EKF based method achieves that level of detection. Similarly, the delay to detection decreased rapidly for the EKF based method as fault magnitude increases, whereas the particle filter method’s decrease is slower.

Another comparison that is of interest in the analysis of the two algorithms is their required time per iteration. This was also recorded during the Monte Carlo simulations. The mean measured values of filter frequency were 10.3 kHz for the EKF and 115 Hz for the particle filter. Thus, in these experiments, the particle filter took two orders of magnitude longer to run than the EKF. This shows a great disparity in computational burden between the two methods. Despite its relatively slow performance, however, the particle filter still exceeded the 90 Hz required for this scenario, making it capable of real-time implementation.

It is readily apparent from the results plots that in this scenario, the EKF based method of fault-detection has outperformed the particle filter based method. This is unexpected, as the quality of estimates for a nonlinear system should improve with a full description of the pdf of the estimate. It is also unexpected as the researchers in [39] showed that for the nonlinear system they considered, the particle filter based method greatly outperformed the EKF based method. A good deal of time and effort was expended in the implementation of this particle filter and its fault-detection method, and the implementation appears to be sound. Some potential sources of this discrepancy in performance are given next. First, the kinematic equations that modeled this system were not highly nonlinear over the time between updates. Additionally, Gaussian process and measurement noise were introduced. The use of non-Gaussian noise could show a marked impact on the detection results of both methods. It is possible that performance of the particle filter based method could be improved through modifications to the underlying estimator. There are many variants and modifications of particle filters that could be explored.
4.9 Conclusion

As mentioned in this chapter, particle filters offer many advantages over Kalman-style filters in terms of flexibility of application. They are especially well suited for nonlinear systems and systems with non-Gaussian noise. They also simplify the tracking of multiple hypotheses. The algorithm is straightforward and easy to understand, as demonstrated by the example in Section 4.3. There are of course drawbacks to a particle filter as well, including the computational cost they incur. The advantages particle filters afford, however, make them an attractive target for a robust fault-detection method.

Despite the many attractive qualities of particle filters and the time spent on the implementation of the particle filter, however, the results in Section 4.8 clearly show that the EKF based fault-detection algorithm developed in Chapter 3 outperformed this implementation of the particle filter based algorithm. It is apparent that the results of these experiments differ from those in [39]. They are also at odds with the notion that a full description of the pdf of the estimate should yield better estimates than a parametric description. Much care has been taken to ensure correct implementation of the PF based fault-detection algorithm, thus future work should seek to understand and overcome these deficiencies. Thus, suggestions for future work include further exploration of the utility of a particle filter based method for detection of faults in nonlinear systems with non-Gaussian noise. This includes further exploration of the method implemented in this chapter. For example, it should be determined whether the particle filter fault-detection scheme would have performed optimally under a different set of parameters. The degree to which system nonlinearity and non-Gaussianity of noise affect the comparison should also be ascertained. Suggested future work also includes the exploration of other potential particle filter based fault-detection methods.
CHAPTER 5. CONCLUSION

This section reviews the main conclusions reached through the research in this thesis. It also gives recommendations for future work.

5.1 Summary of Main Results

One of the focuses of this research was the evaluation of the EKF based fault-detection method using a hardware test. As demonstrated in Chapter 3, the EKF fault-detection algorithm, coupled with the robust estimator described in Section 3.5 is capable of detecting bias, drift, and increased noise in a non-redundant sensor real time onboard an autonomous aircraft. Such a result has not been found in the literature, making this an important contribution. The use of this algorithm can yield more robust estimation amidst the possibility of sensor faults, thus enabling greater autonomy of unmanned aircraft. This should also extend well to other systems for which a Kalman filter or EKF can be implemented.

The other major focus of this research was the comparison of a particle filter based fault-detection method to the aforementioned EKF based method. The simulations detailed in Chapter 4 show that in this implementation, the EKF based method outperformed the particle filter based method. This is in contrast with the results obtained by other researchers in a similar comparison. It also violates the notion that a full description of the pdf should yield better estimates than a parametric description. It is difficult to conclude with certainty the cause of these discrepancies. A great deal of care was taken to ensure proper implementation of the particle filter algorithm.

5.2 Recommendations for Future Work

Due to the discrepancies in the results of the particle filter fault-detection simulations, it is recommended that future work seek to understand and overcome these deficiencies. For
example, it should be determined whether the particle filter fault-detection scheme would have performed optimally under a different set of parameters. The degree to which system nonlinearity and non-Gaussianity of noise affect the comparison should also be ascertained. Other variants of the particle filter could also be implemented to determine whether such could yield improved fault-detection performance. Suggested future work also includes the exploration of other potential particle filter based fault-detection methods.

As regards the EKF based fault-detection method, future work could include studying the effects of changing window sizes for both the tests of mean and covariance, as well as the threshold test. A greater understanding of the fault-detection capabilities could also be gained by testing a greater number of fault levels. The extension of this method to handle the possibility of multiple faulty sensors would be valuable as well. Additionally, the successful integration of information from the different tests to determine a level of confidence in declaring a fault should be examined. Finally, another area that deserves future work is the development of theoretical limits of performance for this method.

It is expected that the work presented in this thesis will be valuable to others interested in the detection of faults in non-redundant sensors and will lead to more robust autonomous systems.
REFERENCES


[34] R. C. Leishman, “A vision-based relative navigation approach for autonomous multirotor aircraft,” Dissertation, Brigham Young University, April 2013. 27, 29, 32


