AN INTELLIGENT UAV PLATFORM FOR MULTI-AGENT SYSTEMS

by

Taashi Kapoor

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Aeronautics and Astronautics

School of Aeronautics and Astronautics

West Lafayette, Indiana

May 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Shaoshuai Mou, Chair

School of Aeronautics and Astronautics

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Dr. Ran Dai

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory Blaisdell

I dedicate my thesis to my family, friends, and mentors. To my parents, Mohit and Pratibha Kapoor and my grandmother, Uma Kapoor for their endless help and support. You always strive to make me a better person and I could have never achieved this without you.

To all my friends and mentors through my undergraduate and graduate journey at Purdue, I will never forget the experiences we shared.

ACKNOWLEDGMENTS

I would like to thank Dr. Shaoshuai Mou for always supporting my ideas and for giving me the opportunity to conduct research in an area that I am passionate about. Dr. Mou not only helped guide me and support me but also presented me with new projects that helped me grow my skills throughout my degree.

I would also like to thank my committee members Dr. Dengfeng Sun and Dr. Ran Dai for serving on my committee and for helping streamline my research.

I would like to thank my fellow lab peers Gautham Vinod, Zehui Lu, and Tianyu Zhou for their help with the various project-based challenges we undertook during my time at the Autonomous and Multi-Agent Systems (AIMS) Lab.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
AI	Artificial Intelligence
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RGB-D	Red, Green, Blue-Depth
ANN	Artificial Neural Network
CCE	Categorical Cross Entropy
GPU	Graphics Processing Unit
SBC	Single Board Computer
ES	Embedded System
NCS2	Neural Compute Stick 2
IR	Infrared
FPS	Frames Per Second
GHz	GigaHertz
MHz	MegaHertz
VPU	Vision Processing Unit
YOLO	You Only Look Once
SSD	Single Shot Detection
FPN	Feature Pyramid Network
ROS	Robot Operating System
CVAT	Computer Vision Annotation Tool
mAP	mean Average Precision
IoU	Intersection over Union
COCO	Common Objects in Context
HITL	Hardware In The Loop
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network

MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
SITL	Software In The Loop
RPM	Revolutions Per Minute
Amp	Amperage
mAh	milliAmpere-hour
GPS	Global Positioning System
CSV	Comma Separated Values
QGC	QGroundControl
ADNN	Anomaly Detection Neural Network
FC	Flight Computer
ESC	Electronic Speed Controller
PDB	Power Distribution Board
ReLU	Rectified Linear Units
FFT	Fast Fourier Transform
V	Volts
LiDAR	Light Detection And Ranging

ABSTRACT

This thesis presents work and simulations containing the use of Artificial Intelligence for real-time perception and real-time anomaly detection using the computer and sensors on-board an Unmanned Aerial Vehicle. One goal of this research is to develop a highly accurate, high-performance computer vision system that can then be used as a framework for object detection, obstacle avoidance, motion estimation, 3D reconstruction, and vision-based GPS denied path planning. The method developed and presented in this paper integrates software and hardware techniques to reach optimal performance for real-time operations.

This thesis also presents a solution to real-time anomaly detection using neural networks to further the safety and reliability of operations for the UAV. Real-time telemetry data from different sensors are used to predict failures before they occur. Both these systems together form the framework behind the Intelligent UAV platform, which can be rapidly adopted for different varieties of use cases because of its modular nature and on-board suite of sensors.

1. INTRODUCTION

Since the dawn of the field of robotics, robots have drastically altered several industries by automating complex tasks and creating new avenues to perform functions that might not be as efficient if performed by human beings or might involve putting a human being at considerable risk. During the Industrial Revolution in the early 1960s, industrial robots first came to the automotive industry to negate human risk by performing dangerous tasks [1]. The main research focus from the 1960s to the 1990s included advancements in mechanical parts, such as actuators and sensors and their calibration. As the use case for robotic manipulation evolved from just the automotive industry to other industries such as food, pharmacy, and logistics, there was a greater need for robots to be more modular and to accommodate different variations in size and function. Robots needed to self-adapt to fit this new environment, and this led to the advance of intelligent robotic systems that had their own problem-solving capabilities.

The early 2000s focused more on the use of sensors to make these robots adapt to their environment by making them "smart", that is, by equipping them with sufficient intelligence and problem-solving skills in the presence of uncertainty. This led to the application of the field of Artificial Intelligence (AI) to the field of robotics [2], which opened the way to use cases that were unimaginable when the field was first established. Instead of robots just being used to manipulate objects, new use cases to automate tasks were available to every industry. As sensors got cheaper and robots got more mobile along with an increase in computing power, autonomous robots were birthed that could carry their own intelligent decision-making processes to do tasks better than human beings. These robots range from food delivery robots such as Starships to complex humanoid robots such as Boston Dynamics's Atlas.

Despite the difference in complexity of these robots, they have several common features such as their ability to perceive their environment and make complex decisions based on the information available in the environment. Perception can be used for locomotion, obstacle avoidance, and finding/determining objects to manipulate. The more unstructured an environment is, the more dependent the robot is on its onboard sensors [3]. While there are

(a) Starship Food Delivery Robot

(b) Boston Dynamics Atlas Humanoid Robot

Figure 1.1. Modern Intelligent Robots

many ways to perceive the environment through different sensors, modern robots rely on either stereo vision cameras or LiDAR (Light Detection and Ranging). The implementation of both these systems is very different depending on the mode of locomotion for the robot. Stereo cameras do not have the accuracy of LiDAR but are lightweight and inexpensive whereas LiDAR is very accurate but also very heavy and expensive. Aerial robots do not have the same luxury as ground based platforms to carry heavy systems as these systems significantly affect the operating time of the platform and rely more on stereo camera based *computer vision* to perceive the environment [4]. To overcome the drawback as compared to LiDAR systems, computer vision fuses software techniques with the output from cameras to regain accuracy and compete with LiDAR systems and also offers extended capabilities such as scene reconstruction, object detection, pose estimation, and 3D scene modeling [5].

As these robots autonomously perform more tasks in unstructured environments with significant human presence, their risk of malfunctioning and causing harm to their environment also increases [6]. Modern robots are equipped to deal with a wide array of failures and encompass several redundancies to reduce the risk of harming others present in their

environment if a failure were to occur. These systems however focus on reducing risk if the failure has already occurred instead of being able to predict a failure and stopping the system before the failure even occurs. Deep Learning techniques can be used to flag anomalies in the robot systems sensors and alert operators of impending system failure much before the failure even occurs. This significantly reduces the risk of operating such autonomous systems in heavily human-populated environments as the system is stopped before the failure can occur. This process is termed *Anomaly Detection* and is a new and upcoming addition to modern robotic systems to increase system safety and reliability.

1.1 Motivation

An Unmanned Aerial Vehicle or UAV is defined as an aircraft without any human pilot, crew, or passengers on board. UAVs were initially used for military purposes only due to their multi-role capabilities and their role in providing a huge boost to the operational efficiency of military agencies [7]. But due to the advancements in UAV technologies seen in the military sector, such as increasing the autonomy of such vehicles, their use case in the commercial, industrial and academic sectors has also boomed in the last decade as they open a new avenue to autonomous robots, the sky. Given that the primary use case for robots was to increase efficiency and perform tasks deemed dangerous for human beings - small, maneuverable robots that could entirely skip navigating rough terrain by flying, opened up the doors to novel applications that were not accessible to terrestrial, ground based robots. From search and rescue UAVs [8] to autonomous warehouse inventory management UAVs [9] to precision agriculture UAVs [10] to even UAVs that can sanitize hazardous environments [11]. Their use cases and possibilities to integrate into different industries are endless due to their extended capabilities and aerial operations.

The biggest hurdle faced by UAVs in recent years has been weight limitation. UAVs fall under the same laws of physics as other aerospace vehicles and have an inverse relationship between vehicle weight and vehicle power. To increase the flight-time of UAVs, bigger batteries are required which significantly increases the weight and lowers flight performance [12]. Autonomous UAVs add another hurdle to this problem by having a weight and size

restriction on the processing power they carry on board. This has caused each industry to carefully determine their particular use case for the UAV and optimize these parameters to fit their use case. The most general solution involves moving the processing off-board to a ground control station but this solution adds another problem of requiring a permanent link to this off-board computer to function. Current computers have not been economically miniaturized enough to provide a UAV with as much processing power as a desktop computer and so UAVs are limited to smaller Single Board Computers and Embedded Systems for their operations. While these SBCs and ESs improve their performance every year to allow UAVs to run more complex algorithms on board, software complexity increases exponentially compared to the performance of these SBCs and ESs [13] [14]. Hence, there is a need to create a modular platform which can be reequipped efficiently depending on the use case and has enough processing power to carry out complex missions requiring extensive computational capacity at an economical price.

The first capability that needs to be explored in building such a system is the problem of perception. To minimize the costs, size, and weight of the UAV, a stereo RGB-Depth camera is used instead of a LiDAR system and this also enables further computer vision applications on the UAV. A real-time object detection algorithm needs to be running on-board to further enable vision-based path planning, localization in unfamiliar environments, dynamic obstacle avoidance and 3D scene reconstruction. Real-time object detection algorithms are known for being computationally intensive [15] so a system that was computationally and financially economical, lightweight and accurate needed to be developed.

The next problem to solve was that of safety and reliability. Despite the extensive array of use cases for UAVs, if they are not safe to implement in human populated environments then the industry will never be able to adopt such aerial robots. A real-time anomaly detection algorithm also needs to be running on-board that can warn the operator of impending system failures before they occur so that safety contingencies can be executed before system failure. These algorithms are also extremely computationally heavy and require immense amounts of training data before accurate results. Therefore a system needed to be developed that could learn from real-time system utilization while not requiring extensive computational power to operate.

This thesis introduces a fully modular UAV platform that can be used for different use cases, that features real-time object detection and real-time anomaly detection running on a SBC (Raspberry Pi 4b). First, this paper compares the performance of state-of-the-art object detection algorithms running on the SBC with a custom object detection system that has up to a 7 times increase in performance. Next, this paper analyzes three different flight plans run through the anomaly detection algorithm and compares the performance of three different neural networks that were selected to run on-board the Intelligent UAV. The Intelligent UAV can be seen in Figure 1.2.

Figure 1.2. The Intelligent UAV Platform

The computational power and sensor set included on-board the UAV can also allow it to act as a leader in a leader-follower multi-robot drone swarm [16]. Cheaper and lighter UAVs can be paired with it for distributed task allocation to carry such swarm missions [17]. It can also be paired with Unmanned Ground Vehicles (UGVs) in a heterogeneous team for collaborative cooperation missions [18]. This platform opens the door for further research in different varieties of UAV use cases by establishing perception and system safety.

1.2 Literature Review

Several UAV platforms have been built to further research and develop use cases on such as a research platform for search and rescue UAVs [19], a UAV platform for Air Transportation Systems [20] and a robust UAV system for operations in constrained environments [21]. But these systems are highly specialized in their use case and can not be deployed outside of the use cases mentioned in these papers. These UAV platforms are also dated and would not be able to run custom neural networks for computer vision applications. As computer vision and AI techniques further develop every day, UAVs can be made smarter and faster and there is a need for a modular system where components can be swapped with minimal disruption to keep up with new technologies.

The Intelligent UAV platform fills in this gap with its fully modular approach so that parts can be changed according to the mission requirements or updated if new technologies can further its capabilities. The Single Board Computer paired with edge AI can handle seemingly daunting tasks such as running complex neural networks on-board and eliminates the need to exchange information with a ground station. Using real-time object detection, the Intelligent UAV can also operate safely and reliably in GPS denied environments which is an up and coming research area for autonomous UAVs [22].

1.2.1 Real-Time Object Detection on UAVs

Real-time object detection on UAVs has been a very popular research topic in the past decade. Modern object detection networks were based on AlexNet [23] which was a Convolutional Neural Network for image classification and further models for image classification were developed such as VGG [24], ResNet [25] and Inception [26]. Modern object detection algorithms use these networks as backbones and speed up classification through various techniques such as Feature Pyramid Networks [27]. A few of the most prevalent state-of-the-art object detection networks are the newer generations of the YOLO Network[28] and MobileNetSSD Network[29].

The only problem with deploying these networks on UAVs is that they are very computationally intensive and need discrete graphics cards to get higher frame rates in real-time

operations. Several research papers introduce new algorithms that have great accuracy but can rarely run on even the most powerful embedded systems with discrete graphics. DAGN which is a real time UAV remote Sensing Image Vehicle Detection Framework requires a GeForce GTX 1080Ti Graphics Processing Unit to be able to achieve real-time object detection [30]. Other solutions encompass lowering the resolution [31] and training a very small model so as to conserve computational resources [32] or running it on a very specialized SBC that is optimized for only object detection [33].

The Intelligent UAV uses a general purpose, commodity on-board SBC that does not have discrete graphics to speed up neural network inference and relies on economical inference accelerating hardware and software techniques to achieve a higher performance than expensive SBCs with discrete graphics. The Intelligent UAV is also able to achieve the same performance as most modern-day desktop computers at real-time object detection using its custom computer vision system at High Definition (HD) video resolution.

1.2.2 Real-Time Anomaly Detection on UAVs

With the recent advancement in Deep Learning and Artificial Intelligence, anomaly detection of UAV sensor data has been a very relevant research area. Several researchers have focused on receiving a telemetry stream from a live vehicle and then performing anomaly detection on the live stream through the ground control station [34]. Other anomaly detection algorithms use Recursive Least Squares to detect changes in telemetry data and report big changes as anomalies [35]. This can lead to a very large number of false positives as if a pilot takes control then deviations between the autopilot system and the pilot taking control can be viewed as an anomaly. Some methods also focus on uploading live flight data to the internet where anomaly detection is performed in the cloud [36]. More accurate methods rely on downloading the data to a desktop computer and then performing anomaly detection after the end of the flight [37].

Only a limited set of publications feature real-time anomaly detection that is performed on-board a UAV without using computational resources from the ground station. One such work attaches external sensors to calculate the variations in motor temperature data while

the UAV is in flight to determine anomalies [38]. Methods that only measure changes in the motor such as RPM and temperature or methods that rely solely on the amount of current flowing through the circuit do not account for the most common type of failure in UAVs which is sensor failure.

The Intelligent UAV is able to use data from a variety of sensors controlled by its flight computer or autopilot system and runs a deep learning algorithm on these sensor values to capture their inherent hidden features. It is then able to label anomalies according to the error in reconstruction of the input data by the neural network and exponentially reduces the amount of false positive anomalies to maximize flight time. It is able to perform anomaly detection during real time flight operations using just the on-board computer while reporting anomalies back to the ground station.

1.3 Contributions

The main contributions of this work are:

- A computer vision system that can run on embedded systems without discrete graphics to perform real time object detection and offers faster training and deployment than conventional methods using Edge Artificial Intelligence. This system can be implemented on-board any UAV platform and training results from a custom developed UAV platform are shown.
- An unsupervised anomaly detection system using neural networks that can also run on any UAV platform's companion computer and can predict major sensor failures before they occur in flight to increase system safety and reliability. The anomaly detection system can also perform real-time operations while the UAV platform is in flight.

2. OBJECT DETECTION

2.1 Introduction

In this chapter, the key ideas and the model used to classify objects for object detection used on-board the Intelligent UAV are discussed. Real world video results and gazebo image results are presented to showcase the accuracy and speed of the model.

2.1.1 Introduction to Object Detection

Object detection can be defined as determining where objects are located in a given image (object localization) and determining which category an object belongs to (object classification). Object detection models can be further divided into three distinct stages: informative region selection, feature extraction and image classification [39].

Since objects can appear in many different locations in an image and an image can have different sizes or aspect ratios, object detection algorithms scan the whole picture by dividing it into different grids and scanning the grids through a multi-scale sliding window. Optimizing this first stage can lead to less computational load per frame or image in an object detection algorithm.

The second stage consists of obtaining distinct visual features in the current grid that can be used in the image classification process to match the object in the current grid with objects that the model was trained to detect. This process can be carried out through Scale-invariant keypoint feature transforms [40], Histograms of Oriented Gradients (HOG) [41] and Haar-like features [42].

The final step in this process, involves distinguishing the current object in the sliding window among all the salient features of the objects the model was trained to detect. The current state of the art methods rely on Deep Neural Networks (DNNs) with Convolutional Neural Networks (CNNs) being the most applicable to real-time object detection [43].

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) comprise of neurons that receive an image as an input, which is converted to a two dimensional or three dimensional matrix based on the height and length of the image, which the neurons then use to try to classify images into different classes and presents probabilities of that image belonging to each class. LeNet was the first CNN [44] in 1998 which was followed by AlexNet, ZF-net, GoogLeNet and VGGNet which kept lowering the base error rate of image classification [45]. ResNet, one of the more recent CNNs, has even managed to beat the human eye's perception capabilities with an error rate of 3.6% as when compared to the human eye's error rate of 5.1% [46].

Layers of Convolutional Neural Networks

Figure 2.1 shows the framework of a typical set of network layers in a Convolutional Neural Network.

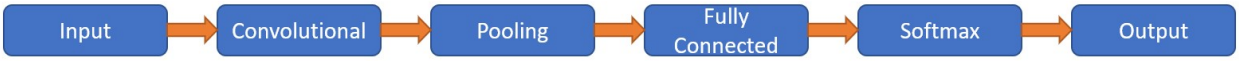


Figure 2.1. Typical Layers in a CNN. Adopted from [47].

- **Input Layer**

  The input layer comprises of an image that is converted to a two dimensional or a three dimensional matrix. The matrix is mathematically represented as $l \times w$ for a two dimensional matrix and as $l \times w \times d$ for a three dimensional matrix where l, w, and d are the length, width and pixel color of the image respectively. The pixel color comprises of the intensity and contrast of a particular pixel on the screen and is usually represented by a number which results from the combination of the RGB - Red, Green, Blue color values.

- **Convolution Layer**

  The convolution layer is responsible for generating new images which accentuate the

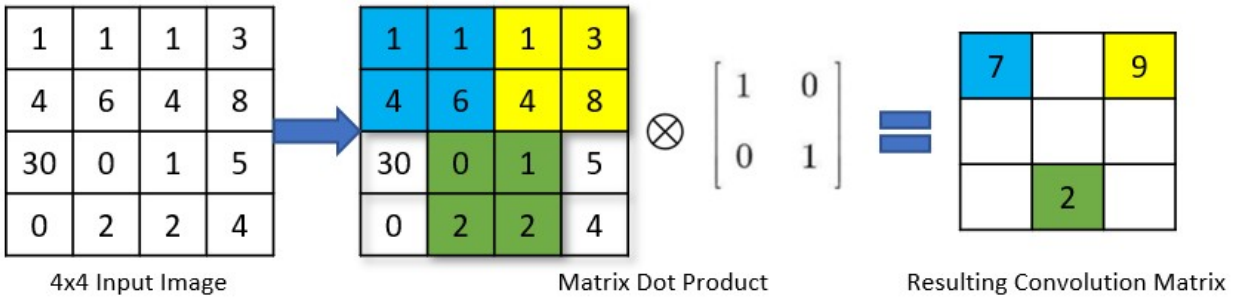


Figure 2.2. Example working of a simple convolution layer

unique features of the original image [48]. These new images are called *feature maps*. Mathematically, the convolution layer can be denoted as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

An easier way to understand this formula can be seen in Figure 2.2 where a 4x4 input image matrix is being convoluted by taking its matrix dot product with 2x2 identity matrix. The 4x4 input image is denoted by $f(t)$ and the 2x2 identity matrix, also called a kernel, is denoted by $g(t)$. The colored 2x2 matrices within the 4x4 matrix, represent the sliding kernel that is used to calculate the feature map. Important parameters to consider in this layer are the size of this sliding kernel, the number of cells it moves per operation or stride of the sliding kernel and the padding of the kernel. Padding is the process in which an outer layer of values is added to increase the size of the image matrix without influencing the dot product so that the original edge values of the image are calculated in more than one sliding kernel which gives a better approximation of the convolution.

- **Pooling Layer**

  The pooling layer is used to reduce the size of the image by combining neighboring pixels of a certain area of the image into a single representative value [48]. This process is used to down-sample the data and extract essential features from the original image while leaving out unessential features that would not help the model. Pooling can be

considered a static convolution layer where the kernel is fixed into certain grids and the convolution areas do not intersect. The two most popular types of pooling can be seen in Figure 2.3 which are mean and max pooling. 4 grids of size 2x2 are constructed into the overall 4x4 input image and for the mean pooling, the mean of the smaller 2x2 grids are extracted whereas for the max pooling, the maximum value from the smaller 2x2 grids are extracted.

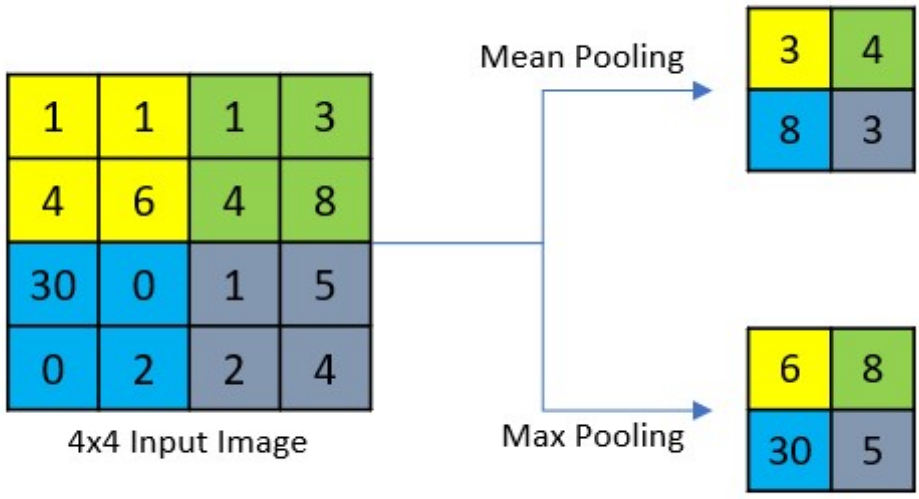


Figure 2.3. Example output of mean and max pooling

- **Fully Connected Layer**

  A fully connected layer is the standard Artificial Neural Network (ANN) model where multiple neurons are arranged in different layers which are in turn separated by a layer containing a matrix of weights. The incoming matrix of values from the Pooling layer is converted into a single vector of values, this process is called *flattening*, before it is passed through the neurons and their respective weights. After this, a non-linear transformation is applied to the product of the neurons and their weights through a

non-linear activation function f. The mathematical formula behind this calculation can be given as:

$$y_{\mathrm{i}}(x) = f(\sum_{\mathrm{i}=1}^{n} w_{\mathrm{i}} x_{\mathrm{i}} + b_0) \tag{2.2}$$

where f is the non linear activation function, n is the number of inputs from the incoming layer, w is the weight connected to the neuron, x is the value of the neuron, b_0 is the bias and y_{i} is the output value which is passed onto the next layer.

Activation functions are used to decide whether a neuron in the current run should be activated or not. The most common example of a non-linear activation function is ReLU (Rectified Linear Unit) which is given by a simple formula $R(x) = \max(0, x)$ where an output of 0 indicated that the neuron was not activated and any other output is passed into the next layer [49]. While the first layer in the fully connected layer can have any number of neurons, the last layer in the fully connected layer normally has the same number of neurons as the number of classes that the model was trained to detect.

- **Softmax**

  The softmax layer is present at the end of the fully connected layer and outputs the probability values of objects matching the classes that the model was trained to detect. The class with the highest probability value is deemed to be the detected object and the sum of all the detected classes probabilities equals 1.

- **Output**

  The final output of a CNN is the class of the object that is detected in the image and the whole object detection process can be visualized in Figure 2.4.

- **Loss Function**

  The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the machine learning model [50]. In multi-object detection problems, the most commonly used loss type is *Categorical Cross*

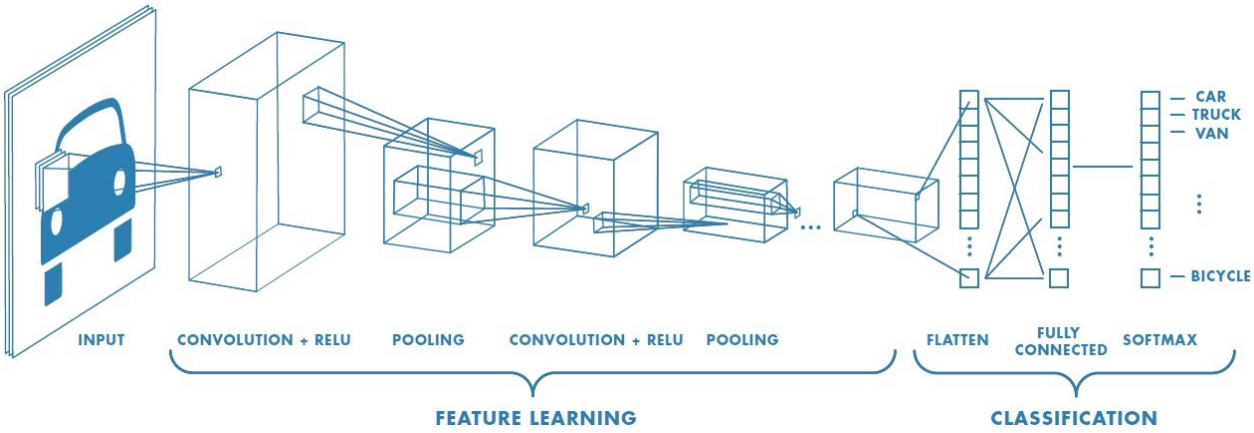


Figure 2.4. Example run of a CNN

Entropy(CCE). The mathematical formula for Categorical Cross Entropy loss is given below:

$$CCE = -\frac{1}{N}\sum_{n=0}^{N}\sum_{\mathrm{j}=0}^{J} y_{\mathrm{j}} * log(\hat{y}_{\mathrm{j}}) + (1 - y_{\mathrm{j}}) * log(1 - \hat{y}_{\mathrm{j}}) \tag{2.3}$$

where y_{j} is the original value, $\hat{y}_{\mathrm{j}}$ is the predicted output, J is the number of classes to predict and N is the number of observations.

2.2 Problem Formulation

Object detection algorithms can be classified into one of two types, one-stage object detection models and two-stage object detection models. A one-stage object detection model is able to detect and predict objects with high speed which makes it the default model for real-time use cases while the two-stage object detection model uses a preliminary stage to highlight regions of importance in an image and then tries to detect objects within those regions, giving it high accuracy but very low speed, making it ideal for static use cases. While several different object detection algorithms exist, only a handful of algorithms are capable of running on embedded systems and out of these few, almost all of them require a discrete GPU of some sort to be able to achieve real-time object detection. For the Intelligent UAV, a one-stage object detection model was required to enable real-time object detection for vision-based navigation along with the added constraint of not having a discrete GPU on-board to simulate an embedded system.

2.2.1 Hardware Setup and Constraints

The Intelligent UAV's object detection setup encompasses an Intel D435 RGB-D (Red, Green, Blue-Depth) camera, a Raspberry Pi 4b SBC (Single Board Computer) and an Intel Neural Compute Stick 2. All three of these devices can be seen in Figure 2.5

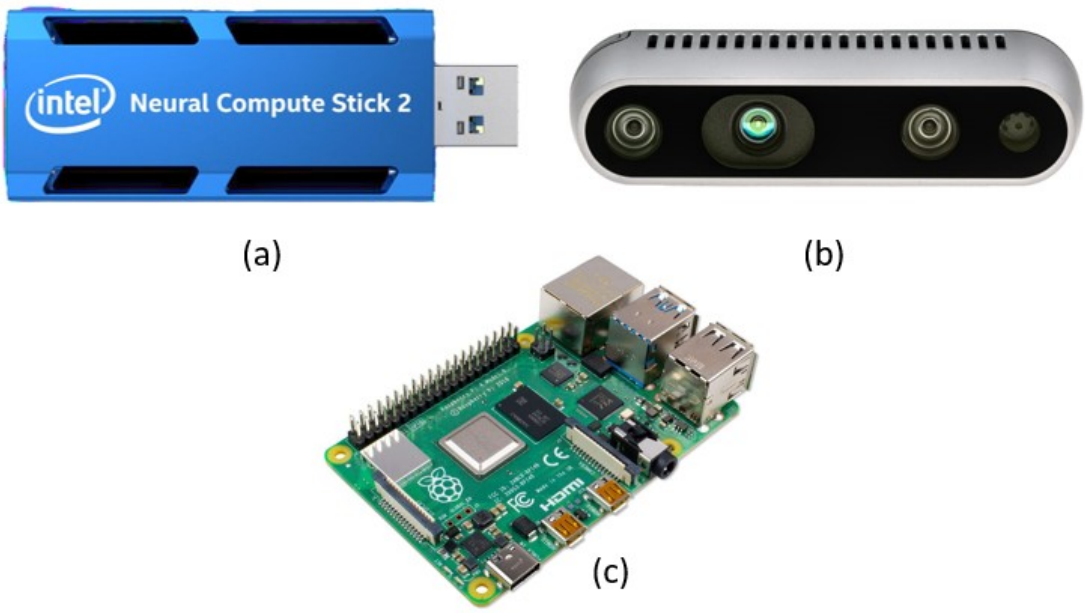


Figure 2.5. (a) Intel Neural Compute Stick 2, (b) Intel D435 RGB-D Camera, (c) Raspberry Pi 4b

.

RGB-D Camera

The Intel RealSense D435 is an active stereo depth camera that pairs the Semi Global Matching (SGM) algorithm with an infrared (IR) projector to calculate depth using its Intel RealSense Vision Processor D4 [51]. The D435 can get up to 90 frames per second (FPS) at 1280x720p with its depth sensors and 30 frames per second at 1920x1080p with its rolling shutter RGB sensor [52]. It also only weighs 72 grams at a form factor of 9 centimeters length and 2.5 centimeters height and depth which is much smaller than its popularly used counterpart the Microsoft Kinect v2. It has a less than 2 percent error rate at detecting

depth at 2 meters and its root mean square error can be calculated using the following formula:

$$\mathrm{e} = \frac{d^2 \times s}{f \times b} \tag{2.4}$$

$$f = \frac{0.5 \times p_x}{tan(\frac{V_h}{2})} \tag{2.5}$$

where f is the focal length, s is the sub-pixel error, b is the baseline, d is the distance in millimeters, V_h is the horizontal field of view and p_x is the resolution in pixels of X.

The Intel RealSense D435 RGB-D camera was chosen due to its small form factor, affordable price, light weight and high performance as compared to all other RGB-D cameras available in the market. The small factor and lightweight characteristics played the biggest part in selecting a camera due to the inverse relationship between payload weight and battery life on a UAV.

Companion Computer

The Raspberry Pi 4b is a Single Board Computer that acts as an Embedded System (ES) aboard the Intelligent UAV. It weighs just 46 grams at a form factor of 8.8 centimeters length, 5.8 centimeters width and 1.95 centimeters height while running a Quad core Cortex-A72 (ARM v8) 64 bit SoC (System on Chip) at a base clock of 1.5 GigaHertz (GHz) and 8GB LPDDR4-3200 SDRAM [53]. It can display video at 60 frames per second at 1920x1080p which made it an ideal pairing with the Intel RealSense D435 camera. The Raspberry Pi 4b also only requires a 5 Volt, 3 Ampere connection to function which reduces the drain on the battery while it functions. Its lightweight, small form factor, low energy cost and performance made it an ideal candidate for the Intelligent UAV when compared to other Single Board Computers in the market.

Hardware Inference Accelerator

The Intel Neural Compute Stick 2 is a USB 3.0 plug and play device for deep learning inference at the edge with its Intel Movidus Myriad X Vision Processing Unit with 4GB RAM

and a processor base clock of 700 MegaHertz (MHz). It is a dedicated hardware accelerator for deep neural network inferences and uses the OpenVino toolkit to optimize performance and boost neural network performance by up to 8 times [54]. The Intel Movidius Myriad X Vision Processing Unit (VPU) network architecture is given by Figure 2.6 and shows where it connects to existing models to boost performance. It was primarily designed to increase frames per second (FPS) of Convolutional Neural Network (CNN) Algorithms running on systems without a Graphics Processing Unit (GPU) which makes it a viable candidate for the Intelligent UAV.

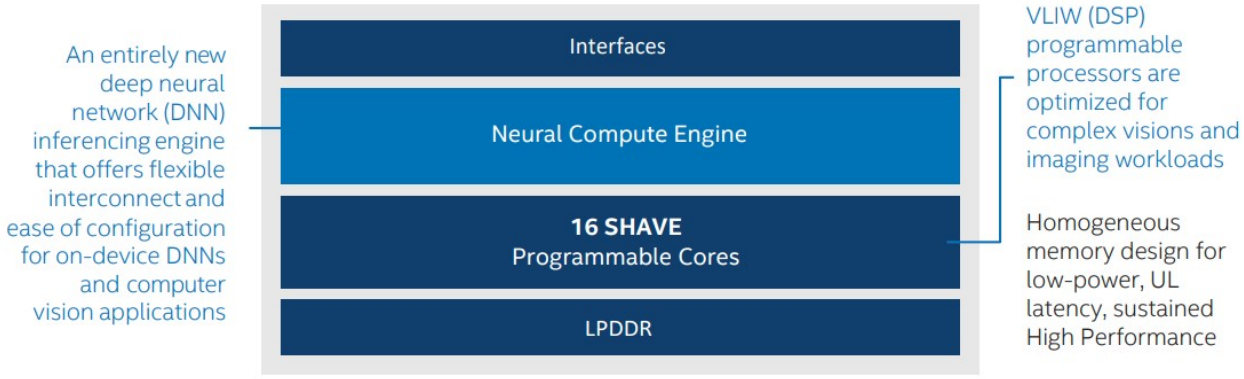


Figure 2.6. The Vision Processing Unit Architecture of the Intel Movidius Myriad X VPU. Adopted from [54]

2.2.2 You Only Look Once (YOLO)

You Only Look Once or YOLO is a state-of-the-art, real-time, one-stage object detection algorithm created by Ultralytics [55] and its latest iteration YOLOv5 is based on the works of Joseph Redmon's YOLOv1-YOLOv3 [28][56][57] and Alexey Bochkovskiy's YOLOv4 [58]. The network architecture of YOLOv5 is shown in Figure 2.7. YOLOv5 comprises of three different stages, it combines Cross Stage Partial Network (CSPNet) [59] into Darknet, creating CSPDarknet as its backbone[60]. CSPDarknet ensures good inference speed and accuracy while reducing model size by solving repeated gradient information problems in large-scale backbones and integrating the gradient changes into the outputted feature map. The neck of YOLOv5 is based on PANet [61] which uses a Feature Pyramid Network (FPN) which

in turn boosts information flow through lower level features in the network. PANet passes detected features in the lower levels directly into the following layer to achieve this boost in performance. The head of YOLOv5 generates 3 different sizes of feature maps which enables multi-scale predictions across objects of different sizes, namely (18x18, 36x36 and 72x72 pixels), which in turn improves object detection performance in the network.

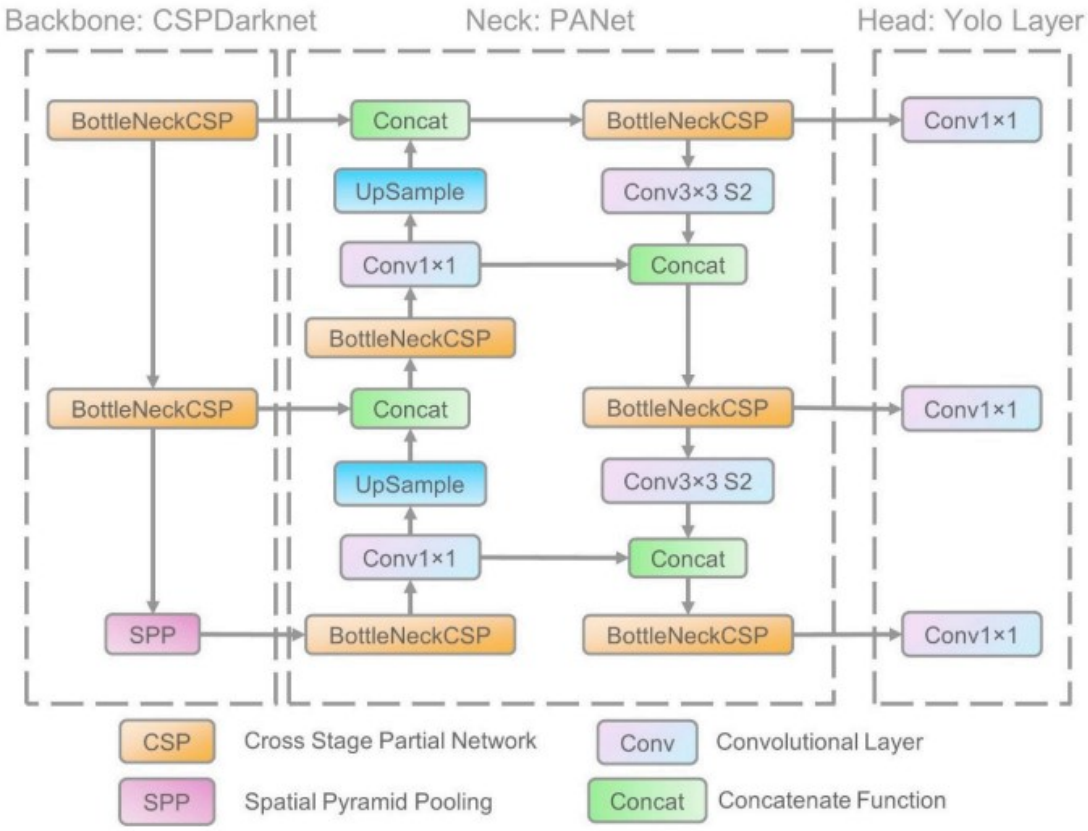


Figure 2.7. The Network Architecture of YOLOv5. Adopted from [60]

.

2.2.3 Mobilenet-SSD

Mobilenet-SSDv2 is another real-time, state-of-the-art, one stage object detection algorithm created by Yu-Chen Chiu et al., based on the popular backbone of Mobilenet-v2 with a Singe Shot Detection (SSD) head as the last layer in the network. Mobilenet-SSDv2 was designed to be as lightweight as possible with little computational load to run on embedded

autonomous systems which made it a viable choice for the Intelligent UAV. The network architecture of Mobilenet-SSDv2 can be split into two different modules, Mobilenet-v2 and the Feature Pyramid Network (FPN). Mobilenet-v2 features a standard convolution layer and 17 inverse residual modules, with each inverse residual module comprising of a 1x1 convolutional layer, a 3x3 depth-wise separable 1x1 convolutional layer, batch normalization and ReLU6 activation functions [29]. The inverse residual modules solve the gradient vanishing problem faced by other models and are able to correctly transfer the gradient information from feature maps into deeper network layers during the backpropagation process. In the FPN module, multi-scale feature maps obtained from the Mobilenet-v2 backbone are fused by unifying the number of channels in each feature map and by resizing feature maps of different scales to improve performance.

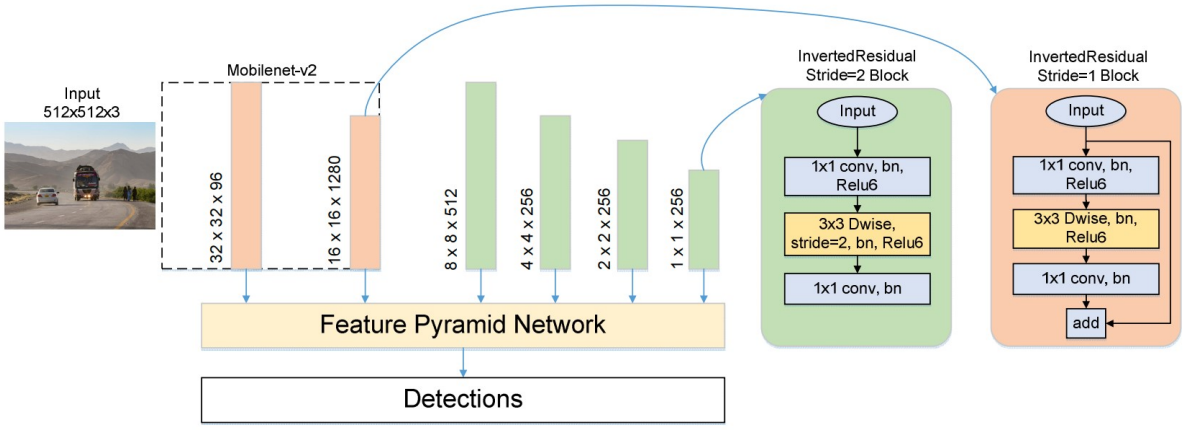


Figure 2.8. The Network Architecture of Mobilenet-SSDv2. Adopted from [29]

.

2.3 Method

Mobilenet-SSDv2 and YOLOv5 were both trained to compare their performance onboard the Intelligent UAV. Training images were taken through a simulated environment in ROS Gazebo [62] and both algorithms were trained through Google Colaboratory [63]. Both algorithms were also exported to OpenVINO format to compare their base performance on the Raspberry Pi 4b with an accelerated version using the Intel Neural Compute Stick 2.

2.3.1 Gathering Training Data

A ROS Gazebo environment was created with several different types of desks, chairs and school bags to simulate common objects in a classroom. A UAV with an Intel RealSense D435 camera plugin was created in the environment and flown manually to capture images of the objects from different angles that the UAV might encounter. Figure 2.9 shows the different object's images taken from this setup to train the network.

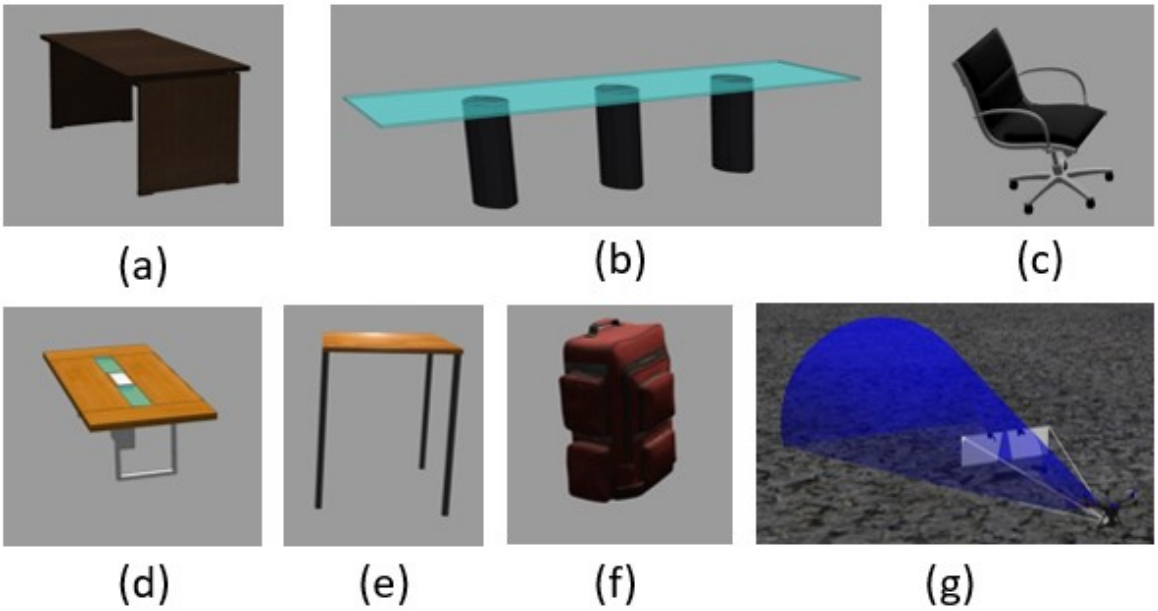


Figure 2.9. Simulated Environmental Classes in Gazebo.
(a,b,d,e) Desk, (c) Chair, (f) Bag, (g) UAV with D435 Camera Plugin

Table 2.1 shows the number of images taken from the UAV through the simulated environment for each class type before they were augmented and fed into both Convolutional Neural Network models. Each of these images was then uploaded to the Computer Vision Annotation Tool (CVAT) [64] for labeling. The process of labeling involves drawing a rectangular box around the object that the CNN model is supposed to detect and manually classifying it into the category of object that you want the CNN to predict. This method is performed on all images that will be fed into the network to provide ground truth to the model. An example of this process being performed on one frame taken from the simulation can be observed in Figure 2.10. This is a very popular approach and is termed "Supervised

Learning" as the model is given some ground truth and labels to know how well it is learning the different classes it is being trained to detect.

Table 2.1. Number of Training Images for Each Class

Object Name	Number of Images
Desk	157
Chair	86
Bag	18
Total	261

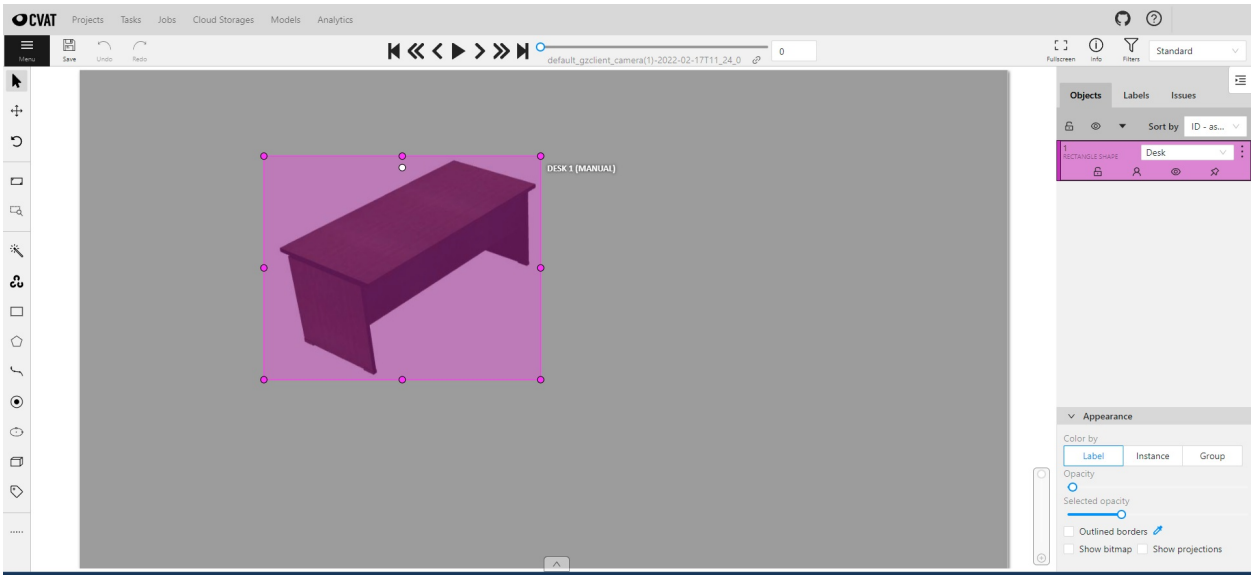


Figure 2.10. Labeling a Desk Using CVAT

Data Augmentation

Most object detection models need at least 150-500 images of each class to see exponential improvement in model accuracy for image classification [65]. The video feed from a UAV is also prone to swaying because of aerodynamic effects such as wind, lack of focus, intermittent pixelating and viewing objects from different angles while in operation. To account for these effects, data augmentation was performed on all the original images captured through the simulation. Data Augmentation refers to the process of applying transformations, filters, resizing and physical augmentation to the original images fed into the model to reduce over-

fitting during the learning process. This not only improves the model's performance in real-world scenarios but also improves performance in scenes that the model was not trained on [66].

To ensure that both object detection models received the same images to train on, image augmentation was performed using a flow-based visual programming language, Roboflow [67], which also divided the image data into training, validation and testing data sets. The image augmentations performed for this section and their description are presented below. An example of augmented images that were fed to the CNN models can be seen in Figure 2.11.

- **Flipping**: Images were flipped along their horizontal and vertical axis.
- **Rotation**: Images were rotated on an axis between -21°and 21°. Rotation above ±21°leads to errors in the data labels.
- **Translation**: Images were morphed to move the objects into different corners of the frame to avoid positional bias.
- **Resizing**: Typical object detection algorithms favor the input data to be of size 640x640 pixels. Images were resized from 1920x1080 pixels to 640x640 pixels.
- **Noise Injection**: A matrix of random values drawn from a Gaussian Distribution is superimposed onto the original image to imitate static noise.

Roboflow was used to divide the entire dataset into three different parts, training, validation and testing after image augmentation was applied. The training dataset contains the images that the object detection models are trained on. The validation dataset is used to compare how well the trained dataset is able to detect images after each epoch, or run of the dataset. We can judge the performance of the dataset by looking at the mAP (mean Average Precision), IoU (Intersection over Union) and loss. The testing dataset is the final dataset on which the trained object detection algorithm is run on to present results.

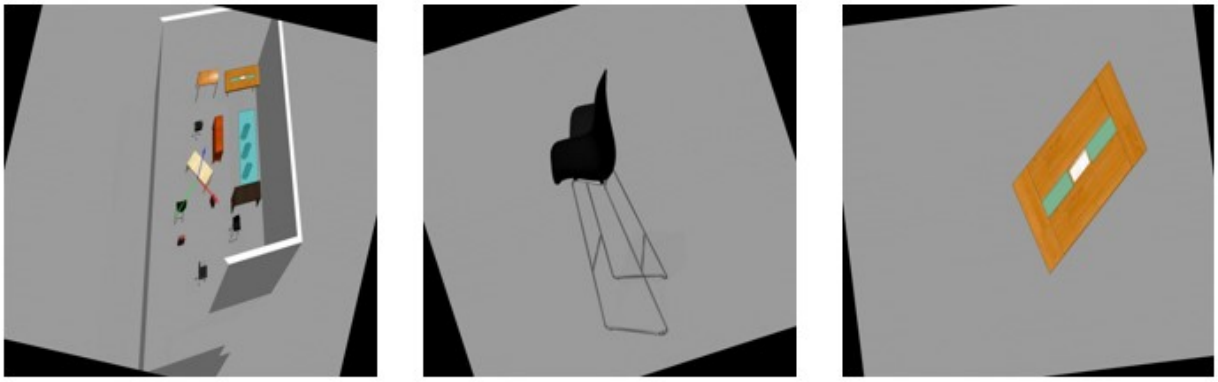


Figure 2.11. Augmented Images

2.3.2 Training the Model

The YOLOv5 model was trained on Google colab using PyTorch [68] whereas the Mobilenet-SSDv2 was trained on Google colab using TensorFlow Hub [69]. While both models were trained on different machine learning frameworks, they were executed on the same instance of Google colab using an Nvidia GeForce Titan X 12GB GPU, 16GB RAM and an Intel i9 processor.

Both models were trained using transfer learning based on their pre-trained weights of the 2017 Microsoft Common Objects in Context (COCO) dataset. The Microsoft COCO dataset is a vast dataset that contains 91 different classifications of objects along with their images and image labels that a four year old human being could potentially identify [70]. CNN models evaluate their performance based on runs of this dataset and their pre-trained weights files are readily available with the source code of the models. Transfer Learning is the process in which a pre-trained network's weights are frozen during training while the last few CNN layers are allowed to learn the new images through training. The first few layers of the CNN are already trained to extract important features from the image and remain the same in the new training network as well but the last few layers that are used to classify the images are re-trained to detect the inputted training objects. This allows for faster training of the network and increased performance due to the vast amount of data it is trained on.

Using hyper-parameter tuning and after several test-runs of the models, it was determined that the learning rate of both networks should be set at $\alpha = 0.0001$. This is the rate at which the last few layers of the CNN are trained. The number of epochs for MobilenetSSD-v2 was set to 300 and for YOLOv5 was set to 250. One epoch is defined as one whole run of the whole dataset. These numbers were chosen after fine-tuning each network to avoid overfitting of the data. The batch size for both models was set to 30. The batch size is defined as the total number of training images present in a single batch sent into an epoch. Table 2.2 shows the training time to train both the algorithms on the same system using the same number of images.

Table 2.2. Comparison of Training Time

Object Detection Model	Number of Epochs	Batch Size	Time Taken to Run
MobilenetSSD-v2	300	30	157.81 Minutes
YOLOv5	250	30	42.34 Minutes

Precision in this section can be defined as the percentage of correct predictions or how accurate the model's predictions are. Recall is defined as how well the model finds the right objects. Their mathematical formulae are given below:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{2.6}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{2.7}$$

IoU (Intersection over Union) measures the overlap between the ground truth bounding boxes and the bounding boxes determined by the model over the total area of both bounding boxes. This is an important characteristic that helps determine if the object predicted by the model is true positive or false positive. An IoU threshold of 0.5 and above is considered a true positive in most cases. The mathematical formula for IoU is given below:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{2.8}$$

The mAP or mean Average Precision is defined as the average of the area under the Precision-Recall curve. The mathematical formula for Average Precision and mean Average Precision are given below:

$$AP = \int_0^1 p(r)dr \tag{2.9}$$

$$\text{mAP} = \frac{1}{n}\sum_{k=1}^{k=n} \text{AP}_k \tag{2.10}$$

where $p(r)$ is the precision-recall curve, n is the number of classes and AP_k is the AP of class k.

Figure 2.12 compares the mean Average Precision (mAP) of both the trained YOLOv5 and MobilenetSSD-v2 models at 0.5 IoU over each epoch for the YOLOv5 model and over each step for the MobilenetSSD-v2 model. A step is defined as one run over one batch size of training data. The YOLOv5 algorithm reached a maximum mAP of 0.985 at 0.5 IoU whereas the MobilenetSSD-v2 reached a maximum mAP of 0.792 at 0.5 IoU.

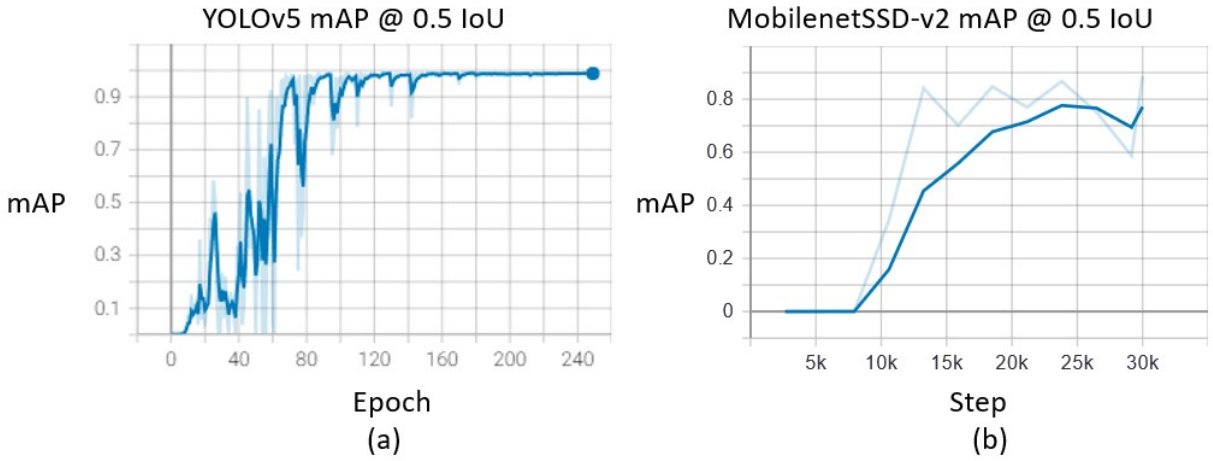


Figure 2.12. mAP Comparison of YOLOv5 and MobilenetSSD-v2 at 0.5 IoU

Figure 2.13 compares the mean Average Precision (mAP) of both models with an incremental increase of 0.05 IoU from 0.5 IoU to 0.95 IoU. The YOLOv5 algorithm reached a

maximum of 0.679 while the MobilenetSSD-v2 reached a maximum of 0.438. These graphs show the average confidence level of true positive matches for the dataset when evaluated.

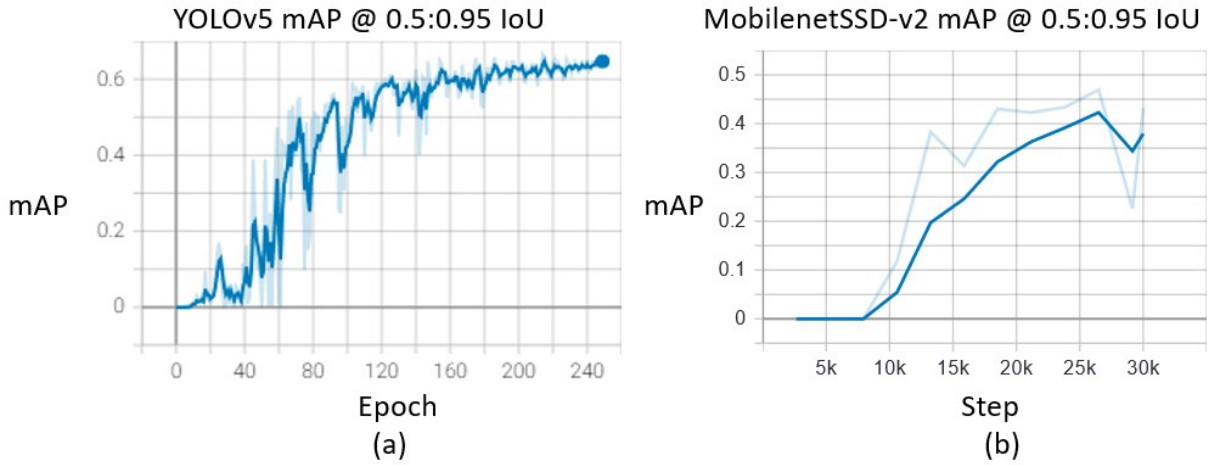


Figure 2.13. mAP Comparison of YOLOv5 and MobilenetSSD-v2 at 0.5:0.05:0.95 IoU

Figure 2.14 compares the total precision of both models over their total training duration. The YOLOv5 algorithm reached a maximum of 0.998 while the MobilenetSSD-v2 algorithm reached a maximum of 0.619. These graphs compare the number of objects correctly detected over the number of total objects detected over time.

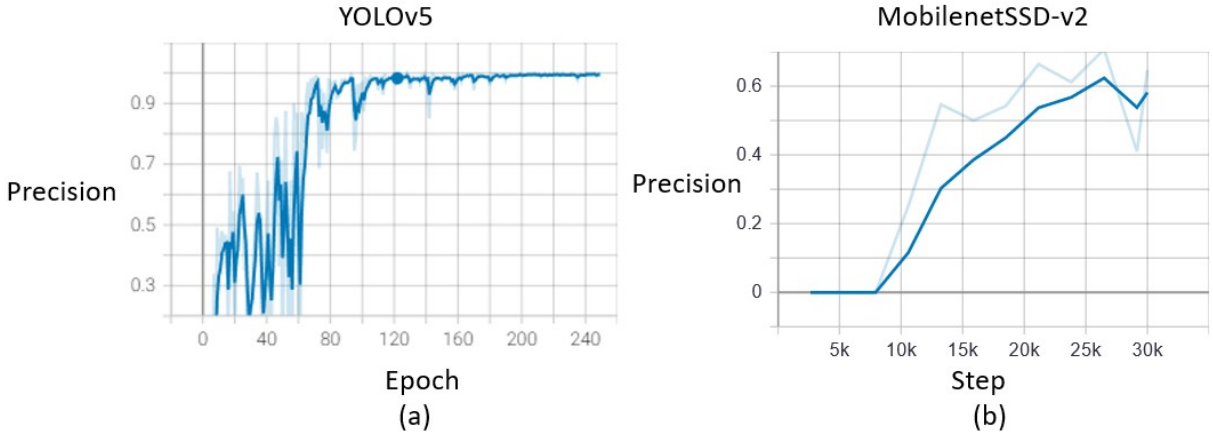


Figure 2.14. Precision Comparison of YOLOv5 and MobilenetSSD-v2

Figure 2.15 compares the total recall of both models over their total training duration. The YOLOv5 algorithm reached a maximum of 0.965 while the MobilenetSSD-v2 algorithm

reached a maximum of 0.631. These graphs compare the number of objects correctly detected over the number of actual objects over time.

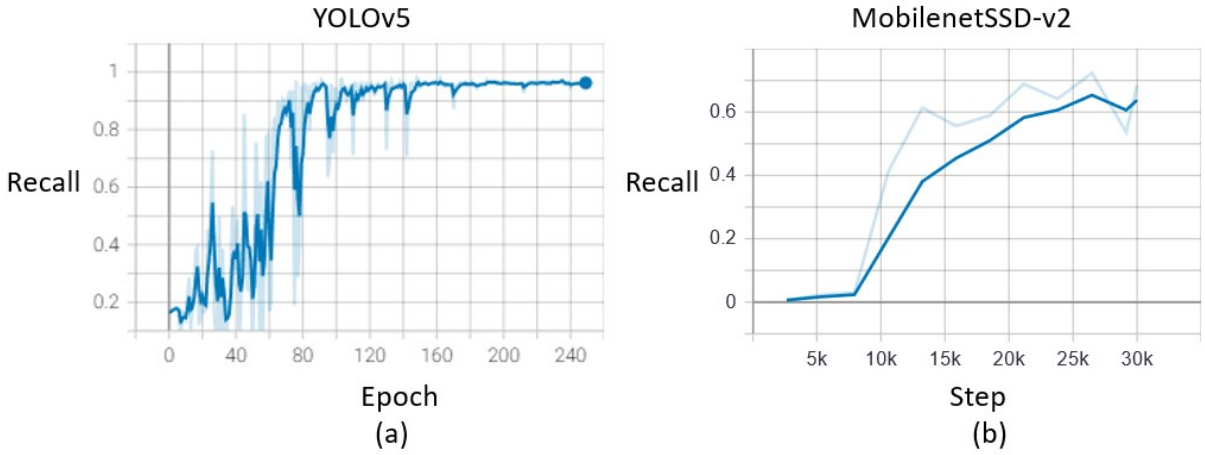


Figure 2.15. Recall Comparison of YOLOv5 and MobilenetSSD-v2

The faint lines in the background of the model comparison figures shown above are the true values before smoothening was performed on the graphs to make them more linear for plotting. The smoothening coefficient was set to $\gamma = 0.6$. The results, as shown in Table 2.3, clearly prove that the YOLOv5 algorithm not only beats the comparatively older MobilenetSSD-v2 algorithm in training time but also performs 30-35% better with a reduced dataset of images for object detection.

Table 2.3. Comparison of Training Results

Object Detection Model	mAP at 0.5 IoU	mAP at 0.5:0.95 IoU	Precision	Recall
MobilenetSSD-v2	0.792	0.438	0.619	0.631
YOLOv5	0.985	0.679	0.998	0.965

2.4 Main Results

This section compares the performance of both algorithms in their gazebo model test phase and their respective FPS on the Intelligent UAV's companion computer and hardware inference accelerator setup.

2.4.1 Gazebo Images Test Run

The MobilenetSSD-v2 and YOLOv5 models that were trained with custom weights described in the section above were run with the same set of testing images and their results can be seen in Figure 2.16 and Figure 2.17 respectively.

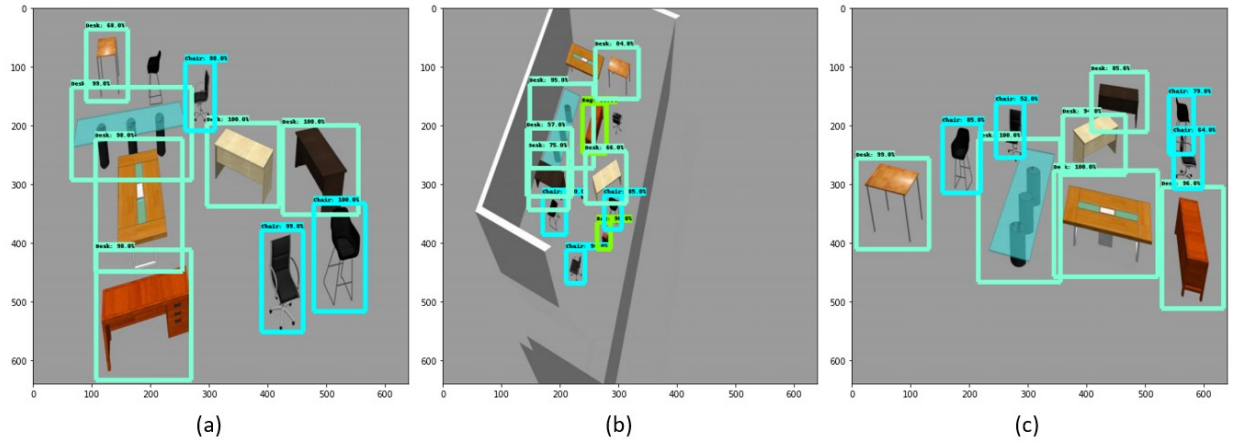


Figure 2.16. MobilenetSSD-v2 Model Deployed on Gazebo Images

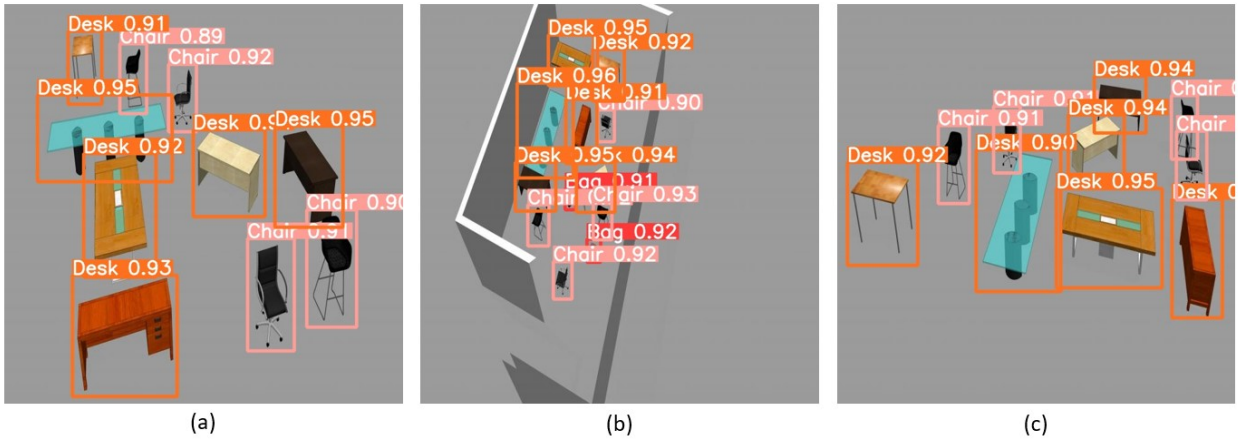


Figure 2.17. YOLOv5 Model Deployed on Gazebo Images

Table 2.4 provides complete details on both the model's performances on the three test images. Through the test results it was found that the YOLOv5 model classified all objects perfectly (38 objects detected out of 38 objects present) in all test images without any false positives or false negatives while the MobilenetSSD-v2 model classified 34 objects out

of the total 38 objects present in the test images with 4 false negative detections. Thus the YOLOv5 algorithm achieved a 100% accuracy in our test while the MobilenetSSD-v2 algorithm achieved a 89.47% accuracy.

Table 2.4. Comparison of Gazebo Testing Results

Model	Image	Object	Found	Total	Max Confidence	Min Confidence
MobilenetSSD-v2	(a)	Desk	6	6	99%	68%
YOLOv5			6	6	97%	91%
MobilenetSSD-v2		Chair	3	4	100%	88%
YOLOv5			4	4	92%	89%
MobilenetSSD-v2	(b)	Desk	5	6	95%	66%
YOLOv5			6	6	96%	91%
MobilenetSSD-v2		Chair	3	4	90%	70%
YOLOv5			4	4	93%	90%
MobilenetSSD-v2		Bag	1	2	90%	90%
YOLOv5			2	2	92%	91%
MobilenetSSD-v2	(c)	Desk	6	6	100%	85%
YOLOv5			6	6	95%	92%
MobilenetSSD-v2		Chair	4	4	85%	52%
YOLOv5			4	4	93%	91%

In the custom object detection test, the YOLOv5 algorithm outperformed the MobilenetSSD-v2 algorithm. Object detection is an important feature in the Intelligent UAV as it allows the UAV to perform visual object detection based mission planning and execution while also actively avoiding obstacles through active obstacle avoidance running through the cameras. If the number of epochs, the batch size and the number of training images were increased on MobilenetSSD-v2 then it could perform better than these initial results but this would also significantly increase the computation load to run the algorithm on the Companion Computer and increase the training time to several hours.

2.4.2 Intelligent UAV Test Run

The Intelligent UAV uses both MobilenetSSD-v2 and YOLOv5 algorithms as a backbone for object detection but requires conversion of the trained weights file into OpenVINO format so that the hardware inference accelerator can boost performance of both models. For this test, both the models were downloaded pre-trained on the Microsoft COCO dataset and

no custom training of the network was performed. The initial weights for the YOLOv5 algorithm are in .pt format which is only understood by YOLOv5 but after conversion to OpenVINO format the weights file is distributed into 3 file types, a .bin file, a .mapping file and a .xml file. The same applied to the MobilenetSSD-v2 model as well which is initially in .pb format and it is then converted into 3 file types, a .bin file, a .mapping file and a .xml file.

Table 2.5 shows the tested average FPS (Frames Per Second) of both algorithms deployed on the Intelligent UAV on a Raspberry Pi 4b conducting real time object detection through the Intel RealSense D435 RGB-D camera with and without hardware inference acceleration through the Intel Neural Compute Stick 2. It is very clear that the Intel NCS2 helps boost the performance of both models during real time inference. A resolution of 1280x720p was considered the bare minimum to ensure that the depth stream from the RGB-D camera was at the same resolution as the incoming color stream from the same camera. The base FPS at 720p resolution without hardware acceleration was 1.9 FPS for the MobilenetSSD-v2 model and 4.3 FPS for the YOLOv5 model whereas with hardware acceleration the FPS was significantly increased to 14.6 FPS for MobilenetSSD-v2 and 32.7 FPS for YOLOv5. For a resolution of 1080p, the base FPS without hardware acceleration was 0.7 and 1.6 for MobilenetSSD-v2 and YOLOv5 respectively whereas with hardware acceleration the FPS was increased again to 4.2 and 10.88 for MobilenetSSD-v2 and YOLOv5 respectively. At 720p the NCS2 was able to boost the performance of both models by about 7.64 times whereas at 1080p the NCS2 was able to boost the performance of both models by about 6.4 times. A screenshot of the real-time object detection test with the MobilenetSSD-v2 model at a resolution of 1280x720p can be seen in Figure 2.18.

Table 2.5. Comparison of Object Detection Results on the Intelligent UAV Setup

Object Detection Model	Resolution	Base FPS	Accelerated FPS
MobilenetSSD-v2	1280x720p	1.9	14.6
YOLOv5	1280x720p	4.3	32.7
MobilenetSSD-v2	1920x1080p	0.7	4.2
YOLOv5	1920x1080p	1.6	10.88

Figure 2.18. Screenshot of Real-Time Object Detection using MobilenetSSD-v2 at 720p Resolution

In conclusion, the YOLOv5 algorithm has the shortest training time and achieves a significant mAP value of classes in the 0.5 IoU region with very high overall precision while also being significantly less taxing on computational power while being deployed for real-time object detection. If the YOLOv5 algorithm is paired with an embedded system without a discrete graphics card, with passive cooling and with a hardware inference accelerator then it can reach close to real time object detection FPS reached by the most powerful mobile processors available in the market.

3. ANOMALY DETECTION

3.1 Introduction

In this chapter, the key ideas and the models used to classify anomalies in sensor data sent to the control system used on-board the Intelligent UAV are discussed. Real world sensor values and simulated HITL (Hardware In The Loop) sensor values are used to demonstrate the system's capability to predict a failure based on anomalous sensor values to increase the overall safety of the system.

3.1.1 Introduction to Anomaly Detection

Anomaly detection can be defined as the process of detecting data instances that significantly deviate from the majority of data instances [71] and can thus be dubbed outliers or anomalies. One of the biggest criteria for human-robot interaction is safety and early detection of faults in a robot is critical to ensure its reliability and safety in its environment. These faults can present themselves as hardware or software related flaws or glitches so it is of utmost importance to be able to spot these flaws before they can escalate into the cause of a major accident. There are several ways to detect these faults such as redundant hardware to compare the same sensor values but these systems increase system cost and weight [72], by setting threshold values on sensor data that the system can not exceed but this can limit system performance [73] and by modeling the system in its environment [74] but these systems can not account for unknown failure [75] or operation in an environment not seen by the system before [76].

Anomaly detection approaches can be divided into three different methods, model based anomaly detection, knowledge based anomaly detection and data driven anomaly detection or more commonly known as deep learning based anomaly detection. Model based anomaly detection algorithms are very accurate but require a very good reconstruction of the system to achieve this accuracy which decreases exponentially as the complexity of the system increases. Knowledge based anomaly detection algorithms diagnose symptoms in the faulty sensor values to compute anomalies and can only deal with anomalies that have been prede-

fined. This work focuses on deep learning based anomaly detection which relies on statistical information to detect outliers and then labels them as faults [77]. The conceptual framework of the working behind such deep-learning based anomaly detection models can be seen in Figure 3.1. The first step of this process as shown in Figure 3.1(a), involves using machine learning to extract the most important features from a high-dimensional dataset. The next step as shown in Figure 3.1(b) involves teaching the model what normal sensor values look like and the final step as shown in Figure 3.1(c) separates small anomalies that could be false positives and detects the highest scoring anomalies as trained in the previous steps.

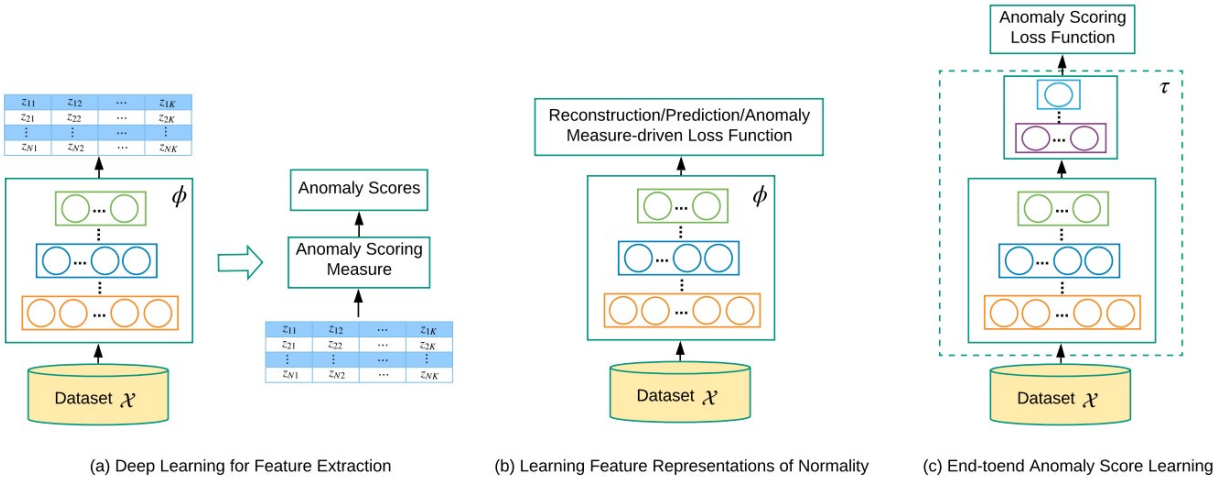


Figure 3.1. Conceptual Frameworks for Deep Learning Anomaly Detection Approaches. Adopted from [71]

The next two sections describe Deep Learning models that were chosen to detect anomalies on-board the Intelligent UAV to create a safer system for human-robot interaction and to increase monitoring and reliability of the system without affecting system performance or capabilities.

3.1.2 Long Short-Term Memory (LSTM) Model

LSTM Models differ from traditional neural networks as they incorporate feedback in their learning compared to most other neural networks that are only feedforward as seen in the previous chapter. This enables LSTM models to overcome long-term dependency

problems faced by other Recurrent Neural Networks (RNNs) due to the vanishing gradient problem [78]. When a large network with multiple layers is used, the network's weights are assigned from the last layer to the first layer by taking the derivative and multiplying this derivative with the layer's number. This causes the gradient to decrease exponentially at the initial layers of the model which are usually the core layers for feature detection [79]. LSTM networks solve this issue by being comprised of several non-linear layers such as a cell state, an input gate, an output gate and a forget gate.

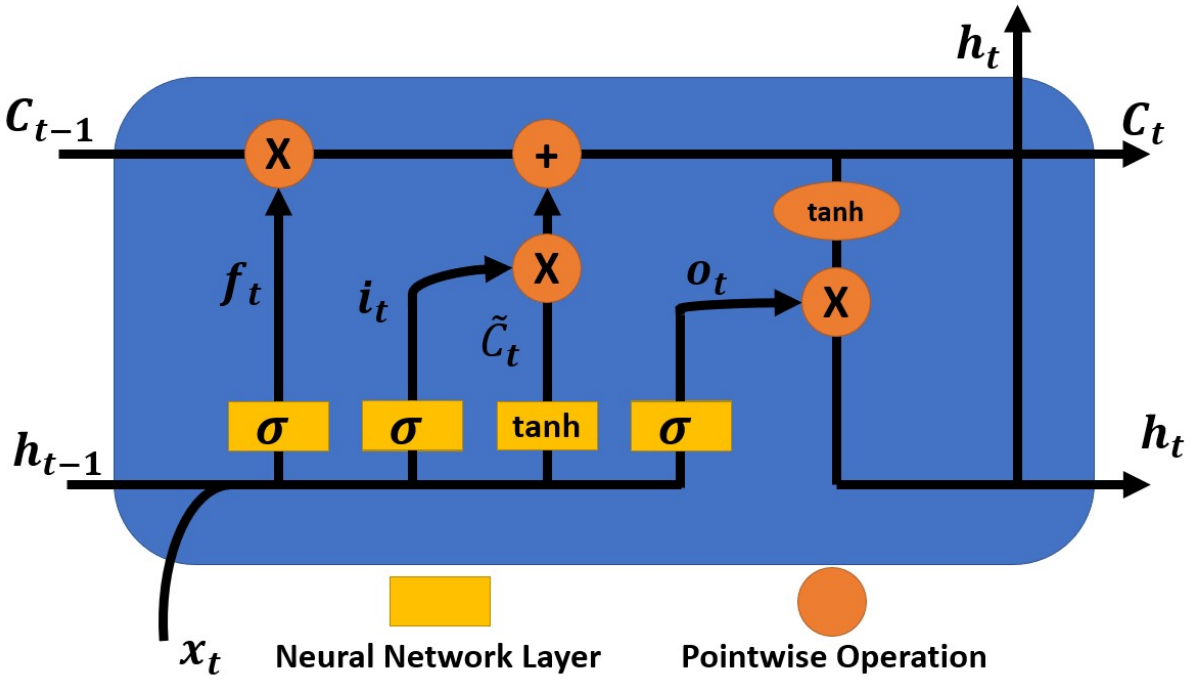


Figure 3.2. Conceptual Framework for LSTM Networks

Figure 3.2 shows the basic working of a LSTM network. The most important feature in an LSTM model is given by the top horizontal line where C_{t-1} is being brought to C_t [79]. This is the cell state of the LSTM and is where the main information of the network flows through like a conveyor belt. The three vertical lines below this horizontal line comprise of three gates (each one being its own neural network) that control the ability to add or modify the information contained in the cell state.

- **Forget Gate**

  The forget gate consists of the lines beginning from h_{t-1} and x_t that then pass through

σ or the sigmoid activation function and get multiplied into the cell state. The sigmoid activation function is responsible for normalizing the values that pass through it between 0 and 1. This gate decides what information from the feedback loop should be added to the cell state where a value closer to 0 ignores the incoming information and a value closer to 1 adds the information to the cell state. Mathematically this gate can be represented as:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.1}$$

where f_t is the forget gate, σ is the sigmoid activation function, W_f is the weight of that layer, h_{t-1} is the previous hidden state, x_t is the current input and b_f is the bias of that layer.

- **Input Gate**

  The input gate consists of two parts, an input gate layer signified by the second σ function and a candidate value layer that generates new candidate values that can be added into the cell state through the tanh layer. The tanh layer functions similarly to the sigmoid activation function and normalizes values going through it between -1 and 1. These two functions work in conjunction through the pointwise multiplication block with the tanh function regulating the values that are going to be added into the network and the sigmoid functions choosing which values to keep from the tanh function before they are all added to the cell state. Mathematically this gate can be represented as:

$$\mathrm{i}_t = \sigma(W_\mathrm{i} \cdot [h_{t-1}, x_t] + b_\mathrm{i}) \tag{3.2}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.3}$$

  where i_t given by Equation 3.3 is the forget gate equation seen above in Equation 3.1 and $\tilde{C}_t$ is the new candidate value layer that uses a tanh function instead of the sigmoid function with the same inputs.

- **Cell State Update**

  The next step in the process is updating the cell state. The old cell state undergoes

pointwise multiplication with the forget gate so that only the important values from the previous cell state are passed onto the process. The output of the input gate then undergoes pointwise addition to add new values that the neural network found relevant to the cell state. Mathematically this update process can be represented by:

$$C_t = f_t * C_{t-1} + \mathrm{i}_t * \tilde{C}_t \tag{3.4}$$

where C_t is the new cell state, f_t is the output of the forget gate, C_{t-1} is the old cell state, i_t is the input gate layer and $\tilde{C}_t$ are the new candidate values.

- **Output Gate**

  The last step in the process is the output gate. The output gate is primarily responsible for updating the next hidden state h_t. The hidden state carries information regarding the previous inputs and is used for predictions alongside x_t at the start of the process at the forget gate [80]. The x_t and h_{t-1} inputs from the left side are inputted to a sigmoid function while the new cell state is passed into a tanh function. The output of both these functions is multiplied to determine the next hidden state. This process is repeated across every LSTM layer. This process can be represented mathematically by:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.5}$$

$$h_t = \tanh(C_t) * o_t \tag{3.6}$$

  where Equation 3.6 is the output of the sigmoid activation function and h_t is the next hidden state found by multiplying the new cell state, C_t by the output of the sigmoid function, o_t.

3.1.3 Autoencoder Model

An autoencoder is an unsupervised artificial neural network that accepts input data, efficiently compresses this data by encoding it and then learns how to reconstruct the data

or decode the compressed data to a representation that is as close to the original input data as possible [81]. It is very commonly used to filter noise or disturbances from the original data by learning a lower-dimensional representation for a higher-dimensional data set and reconstructing it through correlation in the training process [82]. An autoencoder usually consists of 3 main layers, an input layer, one or multiple hidden layers and an output layer. It also consists of 4 pain parts, en encoder, a bottleneck, a decoder and an activation function. The layers and conceptual framework of a simple autoencoder can be seen in Figure 3.3.

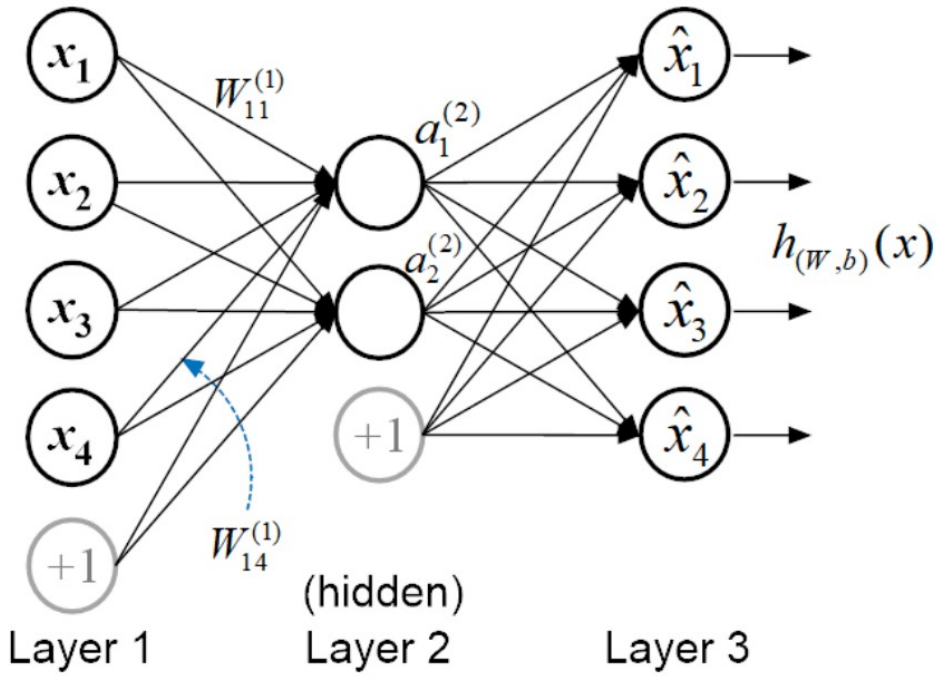


Figure 3.3. Conceptual Framework for Autoencoder Neural Network. Adopted from [83].

- **Input Layer**

  The input layer comprises of an N-dimensional vector that represents the input data with a time series representation of sensor values [83]. Time-series data is a sequence of data that is indexed at the time the measurement was recorded over successive intervals from the same source [84][85]. The input layer can be mathematically represented as:

$$\mathbf{x} = (x_1, x_2, \ldots, x_N) \tag{3.7}$$

- **Output Layer**

  The output layer comprises of the same N-dimensional vector that represents the input data but with reconstructed data values. Unlike other neural networks where the output is a different variable, it is denoted as a modified version of the input [83]. The output layer is mathematically represented as:

$$\hat{\mathbf{x}} = (\hat{x_1}, \hat{x_2}, \ldots, \hat{x_N}) \tag{3.8}$$

- **Hidden Layers**

  The hidden layers are present between the input and output layers and work to extract pertinent information about the relationship between values to encode the input. the hidden layer can be mathematically represented as:

$$\mathbf{h} = f(W * x + b) \tag{3.9}$$

  where $\mathbf{h}$ is the hidden layer output, f is an activation function that represents the neural network layer, W is a weight matrix, x is the input vector and b is the bias vector.

- **Encoder**

  The encoder comprises of a set of convolutional blocks and pooling modules that compress the input data into a lower dimensional version before feeding it into the bottleneck [82].

- **Bottleneck**

  The bottleneck captures all the pertinent information from the input data and restricts the flow of trivial data from the encoder [82]. This block also helps reduce overfitting and over learning of the input data by the neural network.

- **Decoder**

  The decoder comprises of a set of upsampling and convolutional blocks [82] that recre-

ate the downsampled input data into the same dimensions as the input of the encoder block.

- **Activation Functions**

  Each neuron in the hidden layer represents an operation with an activation function as denoted by f in Equation 3.9. The three most commonly used activation functions are Rectified Linear Units (ReLU), tanh and sigmoid activation functions. Figure 3.4 shows the graphical representation of all three of the commonly used activation functions in autoencoders. The ReLU function returns a value between 0 and the maximum value of the input value, the Sigmoid function returns a value between $(0, 1)$ and the Tanh function returns a value between $[-1, 1]$ Their mathematical representation is also given below:

$$\text{ReLU}(z) = \max(0, z) \tag{3.10}$$

$$\text{Sigmoid}(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{3.11}$$

$$\text{Tanh}(z) = \frac{\mathrm{e}^{z} - \mathrm{e}^{-z}}{\mathrm{e}^{z} + \mathrm{e}^{-z}} \tag{3.12}$$

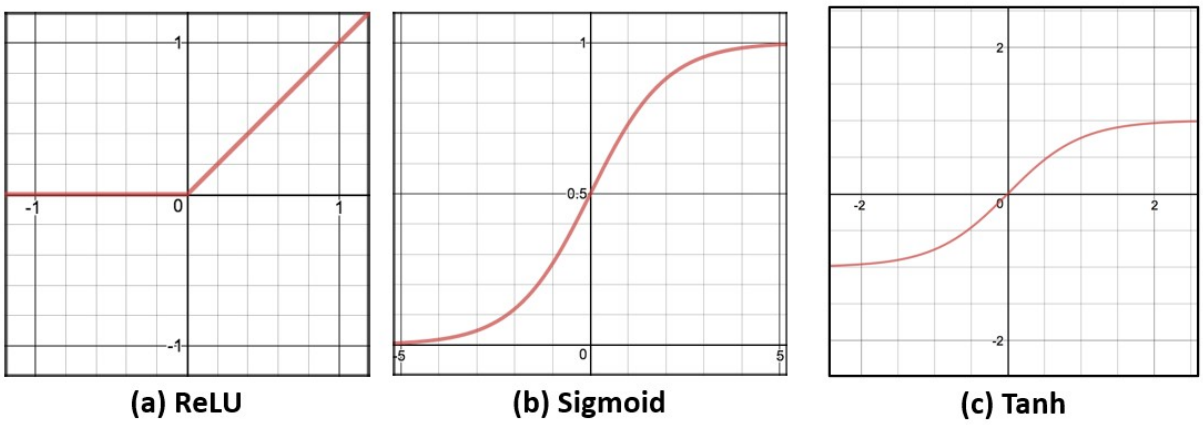


Figure 3.4. Graphical Working of Activation Functions. Adopted from [86]

.

3.2 Problem Formulation

Anomaly detection neural networks work to recreate the most salient features of the input data and are only trained with normal input data that does not have any faults or failures. This enables them to reconstruct normal data values that are within the range of the input data's normal functioning data values even when there is a failure in the dataset. The deviation of the input data's normal functioning data values and the predicted values outputted by the autoencoder can be used as an anomaly score to detect failures in the original dataset. This deviation is also termed reconstruction error [87]. For the Intelligent UAV, anomaly detection was run on-board the flight computer to monitor data from various sensors such as motor RPM (Revolutions Per Minute), motor temperature, battery voltage and current, altitude, thrust, roll angular rate and pitch angular rate to try to predict failure before it occurred and flag anomalies in the flight log. The case for network intrusions aboard the Raspberry Pi 4b was not considered for this section due to limitations on the computing power on-board the Intelligent UAV.

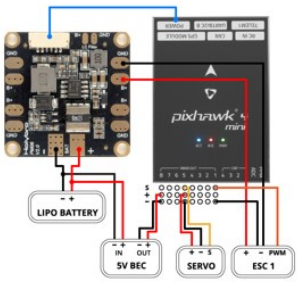
3.2.1 Hardware Setup and Constraints

The Intelligent UAV's electronic system encompasses a Pixhawk 4 Mini Flight Controller, 4x BR2212 980 kV Brushless Electric Motors, 4x 2-4S 20 Amp Electronic Speed Controllers and a 5000 mAh 60C Lithium Polymer Battery. These devices are pictured in Figure 3.5.

Flight Controller

The Pixhawk 4 Mini is an open-source autopilot/avionics flight system that provides a feature-rich, low-cost autopilot for academic, hobby and developer communities. It supports multiple flight software stacks like PX4 [88] and ArduPilot [89]. The Intelligent UAV uses the PX4 flight stack due to its BSD license as compared to the ArduPilot flight stack's GPL license. Technical specification for the Pixhawk 4 Mini can be viewed in Table 3.1.

The Pixhawk 4 Mini interfaces with the companion computer (Raspberry Pi 4b) on-board the Intelligent UAV through a UART port that provides it with power and data transfer



(a) Pixhawk 4 Mini Flight Controller

(b) BR2212 980kV Brushless Electric Motors

(c) 20 Amp 2-4S Electronic Speed Controller

(d) 5000 mAh 60C LiPo Battery with XT60 Connector

Figure 3.5. Electronic Systems used for Anomaly Detection on the Intelligent UAV.

.

Table 3.1. Pixhawk 4 Mini Technical Specifications. Adopted from [90].

Manufacturer	Holybro and Auterion
Model	Pixhawk 4 Mini
FMU Processor	STM32F765 (32 Bit Arm Cortex-M7, 216 MHz, 512KB RAM)
Sensors	Accelerometer, Gyroscope, Barometer, Magnetometer
Power Input	4.75-5.5VDC
Interfaces	8 PWM Servo outputs 2 I2C Ports Micro USB Port 3 general purpose serial ports 3 SPI buses 1 CANBus SD Card

capabilities. The control and output of the entire system is viewed through a GUI called QGroundControl [91]. QGroundControl also provides a live telemetry monitoring stream from the Pixhawk 4 Mini that is fed into the Raspberry Pi 4b for anomaly detection. The

telemetry data that is fed into the anomaly detection algorithm includes the following time-series data:

- Altitude Estimate: The height above the ground the vehicle is believed to be at.
- Roll Angle: The angle made through rotations on the longitudinal axis. Denoted by ϕ in the body frame and x in the inertial frame in Figure 3.6.
- Pitch Angle: The angle made through rotations on the lateral axis. Denoted by θ in the body frame and y in the inertial frame in Figure 3.6.
- Yaw Angle: The angle made through rotations on the vertical axis. Denoted by Ψ in the body frame and z in the inertial frame in Figure 3.6.
- Roll Angular Rate: The measure of roll rotation rate with relative to resting roll angle.
- Pitch Angular Rate: The measure of pitch rotation rate with relative to resting pitch angle.
- Yaw Angular Rate: The measure of yaw rotation rate with relative to resting yaw angle.
- Velocity: Measure of speed of the object relative to the ground.
- Acceleration: Measure of rate of change of velocity.
- Power: A matrix containing Battery Voltage and Battery Current.
- Thrust: The amount of power supplied to the motors for flight. This is limited to a scale of 0 to 1 which corresponds to no power and full power respectively.

3.2.2 Predicting Anomalies

The first step in predicting anomalous behavior through Neural Networks is by training the network with healthy sensor data values. The network will learn to reproduce these healthy values and will learn the correlations and interactions between the inputted variables.

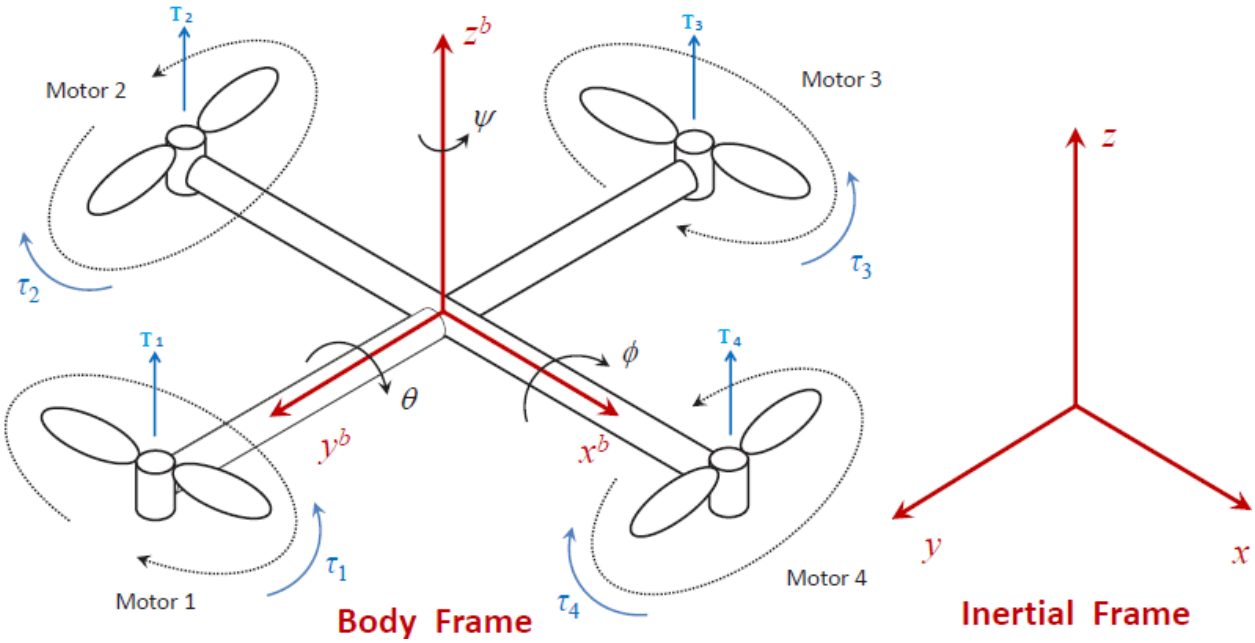


Figure 3.6. Quadrotor Free-Body Representation of Roll, Pitch and Yaw Angles. Adopted from [92].

The network then produces output values that match the healthy input values and if there is an anomaly then the network produced output value and the anomalous data point will have an abnormal reconstruction error. This reconstruction error can be used to predict anomalies in the sensor data. The reconstruction error can be calculated in multiple ways, some of the ways used by this work are given below:

- **Mean Squared Error:** The mean squared error (MSE) is the average of the squared difference between the original data values and the predicted data values calculated by the neural network and it measures the variance of the residuals of the data. It can be mathematically denoted by:

$$\text{MSE} = \frac{1}{N}\sum_{\text{i}=1}^{N}(y_{\text{i}} - \hat{y})^2 \tag{3.13}$$

  where N is the number of data points, y_{i} is the original data value and $\hat{y}$ is the predicted data value.

- **Mean Absolute Error:** The mean absolute error (MAE) is the average of the absolute difference between the original data values and the predicted data values calculated by the neural network and it measures the average of the residuals of the data. It can be mathematically denoted by:

$$\text{MAE} = \frac{1}{N}\sum_{\text{i}=1}^{N} \mid y_{\text{i}} - \hat{y} \mid \tag{3.14}$$

  where N is the number of data points, y_{i} is the original data value and $\hat{y}$ is the predicted data value.

- **Root Mean Squared Error:** The Root Mean Squared Error is the square root of the Mean Squared Error and measures the standard deviation of the residuals of the data. It can be mathematically denoted by:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{\text{i}=1}^{N}(y_{\text{i}} - \hat{y})^2} \tag{3.15}$$

  where N is the number of data points, y_{i} is the original data value and $\hat{y}$ is the predicted data value.

After determining the reconstruction error, a threshold value can be set to encompass values of data that are healthy which correspond to values under the threshold value and values of data that are anomalies or values that are above the threshold. There are again multiple routes to determining this threshold value depending on the input data and the neural network's performance. In this work, the two ways to monitor anomalies depend on whether the training data contains on healthy sensor values or if it also contains anomalous sensor values.

- **Healthy Sensor Training Values:** When the training data only comprises of healthy sensor values and the neural network has converged to minimize training losses, the threshold can be calculated as the maximum value of the Mean Absolute Error or Mean Squared Error times half the standard deviation of the Mean Absolute Error or Mean Squared Error. This technique ensures that the maximum reconstruction error is

contained within the bounds of half a standard deviation of reconstruction loss above the maximum reconstruction loss and the minimum reconstruction error. Any value above this threshold value can be deemed anomalous.

$$\text{Threshold}_{\text{Healthy-MAE}} = \text{MAX(MAE)} + \frac{1}{2}\text{STD(MAE)} \tag{3.16}$$

$$\text{Threshold}_{\text{Healthy-MSE}} = \text{MAX(MSE)} + \frac{1}{2}\text{STD(MSE)} \tag{3.17}$$

- **Mixed Sensor Training Values:** When the training data comprises of both healthy and anomalous values or if the training data doesn't have a common or discernible trend then the threshold value can be calculated as the sum of the mean of the Mean Absolute Error or Mean Squared Error and n times the standard deviation of the Mean Absolute Error or Mean Squared Error. The n term depends on the dataset's reconstruction loss and can be manually fitted to ensure that only healthy sensor data is within the reconstruction bounds. For a dataset that has a marginal increase in anomalous data compared to healthy data, the n term is contained between $(0, 10]$ and for datasets that have exponential increase in anomalous data compared to healthy data, the n term is contained between $(0, 1]$. It is highly advised to train the network only on healthy sensor training values to automatically optimize reconstruction losses and set an appropriate threshold.

$$\text{Threshold}_{\text{Mixed-MAE}} = \text{MAX(MAE)} + (n * \text{STD(MAE)}) \tag{3.18}$$

$$\text{Threshold}_{\text{Mixed-MSE}} = \text{MAX(MSE)} + (n * \text{STD(MSE)}) \tag{3.19}$$

3.3 Method

An Autoencoder model, a LSTM model and an Autoencoder-LSTM model were trained to compare their performance on-board the Intelligent UAV. Pixhawk 4 Mini training data was collected through several ROS Gazebo [62] Hardware-In-The-Loop simulations and real world training logs were downloaded from PX4 Autopilot's Flight Log Review repository [93] which contains publicly available data of Pixhawk flight logs of UAVs. Flights logs that

contained anomalies were simulated through ROS Gazebo with a PX4 HITL [94] simulation that was induced with various failures in flight.

3.3.1 Gathering Training Data

This section describes the process through which training data was acquired through the SITL Simulation and the PX4 Autopilot's Flight Log Review repository.

QGroundControl Parameters

A Pixhawk 4 Mini ROS Gazebo simulation was created to gather flight log data for anomaly detection. QGroundControl was used as a Graphical User Interface (GUI) to create flight mission plans that the simulated UAV would follow and a PX4 command shell was called to inject failures in flight at a desired time. More details on how to setup a PX4 SITL simulation using ROS Gazebo can be found in [94].

Before planning missions on QGroundControl, two important parameters need to be changed within QGroundControl for data logging and failure injection:

- **Enabling Telemetry Logs from Vehicle:** Under Application Settings, in the Telemetry Logs from Vehicle Section, Save Log after each flight and Save CSV log of telemetry data needs to be enabled.
- **Enabling Failure Injection:** Under Vehicle Setup, in the Parameters section, SYS_FAILURE_EN needs to be enabled to inject failures using the PX4 command shell.

QGroundControl Mission Planning

To further challenge the Anomaly Detection Neural Network and to test performance close to real world conditions, two different types of missions were setup. The first mission plan followed a simple circle with an entry and exit close to the takeoff and landing position. The altitude for the UAV was set to 20 meters with random 5 meter increments and decrements while flying in the circle. This mission plan was created as it followed an easy

to determine pattern which would help the Neural Network learn the flight characteristics of the UAV and recreate data with comparatively less reconstruction error. The flight plan can be seen in Figure 3.7. Failures in this flight plan were added to the accelerometer, gyroscope and magnetometer in Waypoint 22 as seen in Figure 3.7. The failures included publishing the same values for all three sensors at random intervals and then publishing seemingly uninitialized sensor values of the UAV which were still within the threshold of operation.

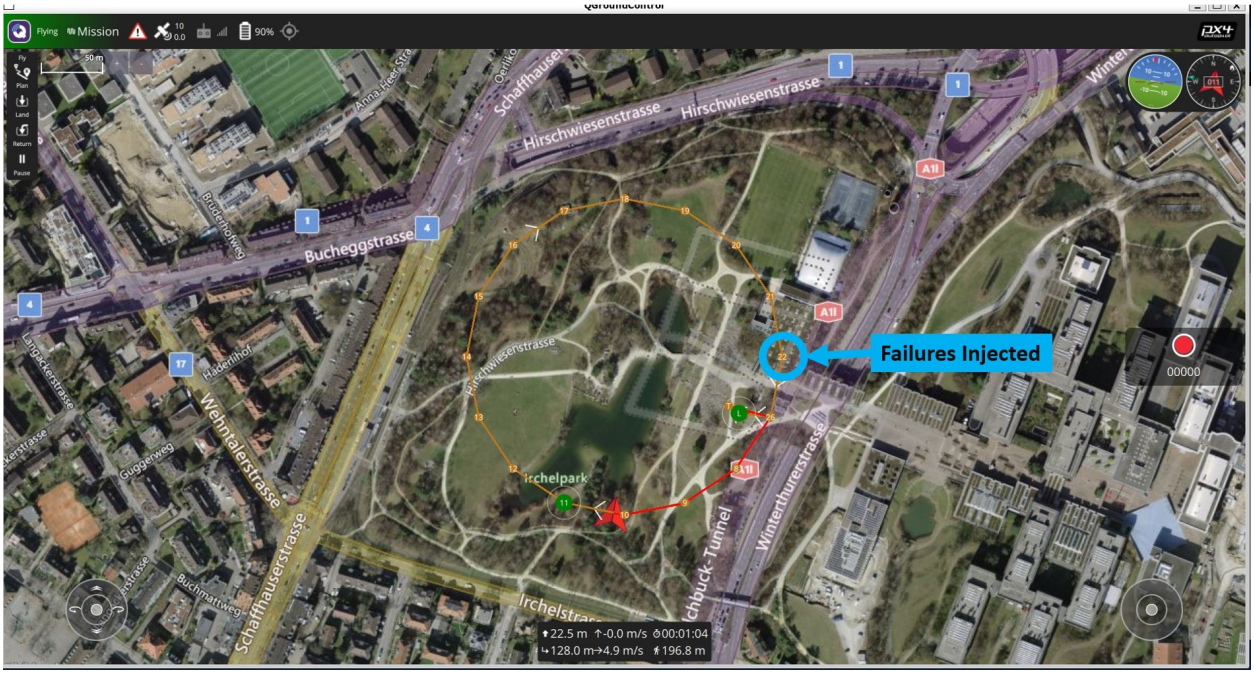


Figure 3.7. Circular Mission Plan on QGroundControl

The second mission plan followed a random zig-zag flight path with changes in altitude, speed and big yaw adjustments along with flying in straight lines in between the zig-zag pattern as seen in Figure 3.8. This mission plan would test the Neural Network as there was no repeating pattern in the flight plan. Each zig-zag maneuver was performed at different velocities and included a random yaw adjustment. This would cause a much higher reconstruction error when the Neural Network tried to recreate the sensor values as compared to the circular flight plan and would test if the Neural Network would falsely identify these changes as anomalies. Failures in this flight plan were added to the accelerometer, gyroscope and magnetometer in Waypoint 20 as seen in Figure 3.8.The failures included publishing

the same values for all three sensors at random intervals and then publishing seemingly uninitialized sensor values of the UAV which were still within the threshold of operation.

Figure 3.8. Zig-Zag Mission Plan on QGroundControl

Failures were only added to an assortment of sensors on board the UAV such as the Accelerometer, Barometer and Gyroscope as failures that were induced in the motor and other critical flight systems would cause the UAV to crash immediately and only create a single, easy to identify anomaly for the Neural Network. Figure 3.9 shows the PX4 command shell where the failures were injected during a simulation in the last couple waypoints before the vehicle landed. Simulating sensor failures in flight surprisingly did not create major deviations from the flight mission plan and could only be noticed through thorough examination of the flight telemetry data. The Anomaly Detection Neural Network was only run on data that included sensor values as these were the only values affected by the failure injection.

PX4 Autopilot Public Flight Logs

The flight telemetry data captured in the previous section contained data for autonomous flights only. A use case for manual human-controlled flight and a flight in real-world test

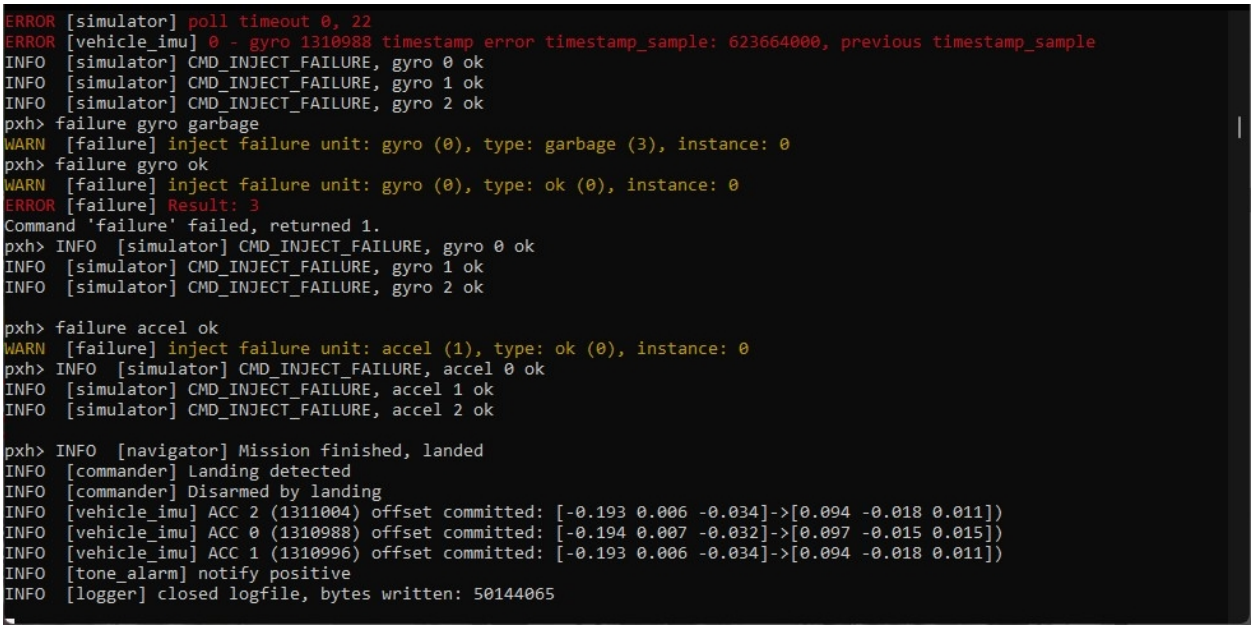


Figure 3.9. PX4 Command Shell Showing Failure Injection in Flight

conditions was also sought to test the performance of the Anomaly Detection Neural Network. This was considered a bonus objective rather than a mission critical objective as the Intelligent UAV would only fly autonomously. The flight log data was gathered from [95] and can be viewed through Figure 3.10.

The flight log data contains telemetry data from a Generic Quadrotor that was controlled by a human pilot through the Pixhawk 4 Mini Flight Computer. It was a 14 minute and 41 second flight on a relatively new Quadrotor build as it had only flown for 3 hours 1 minute and 23 seconds in total throughout its life. The pilot described the winds as calm and claimed that the crash was caused only due to pilot error. The flight's GPS logs show a octagonal shaped flight path and the crash occurs towards the end of the flight at around 13 minutes of flight time. The sensor data from this flight will be described in the next section through the training process of the Neural Network.

Conversion of Flight Telemetry Data Log

QGroundControl outputs the flight telemetry data log as a file with extension .ulog which can only be opened and analyzed through QGroundControl but the PX4 Autopilot GitHub

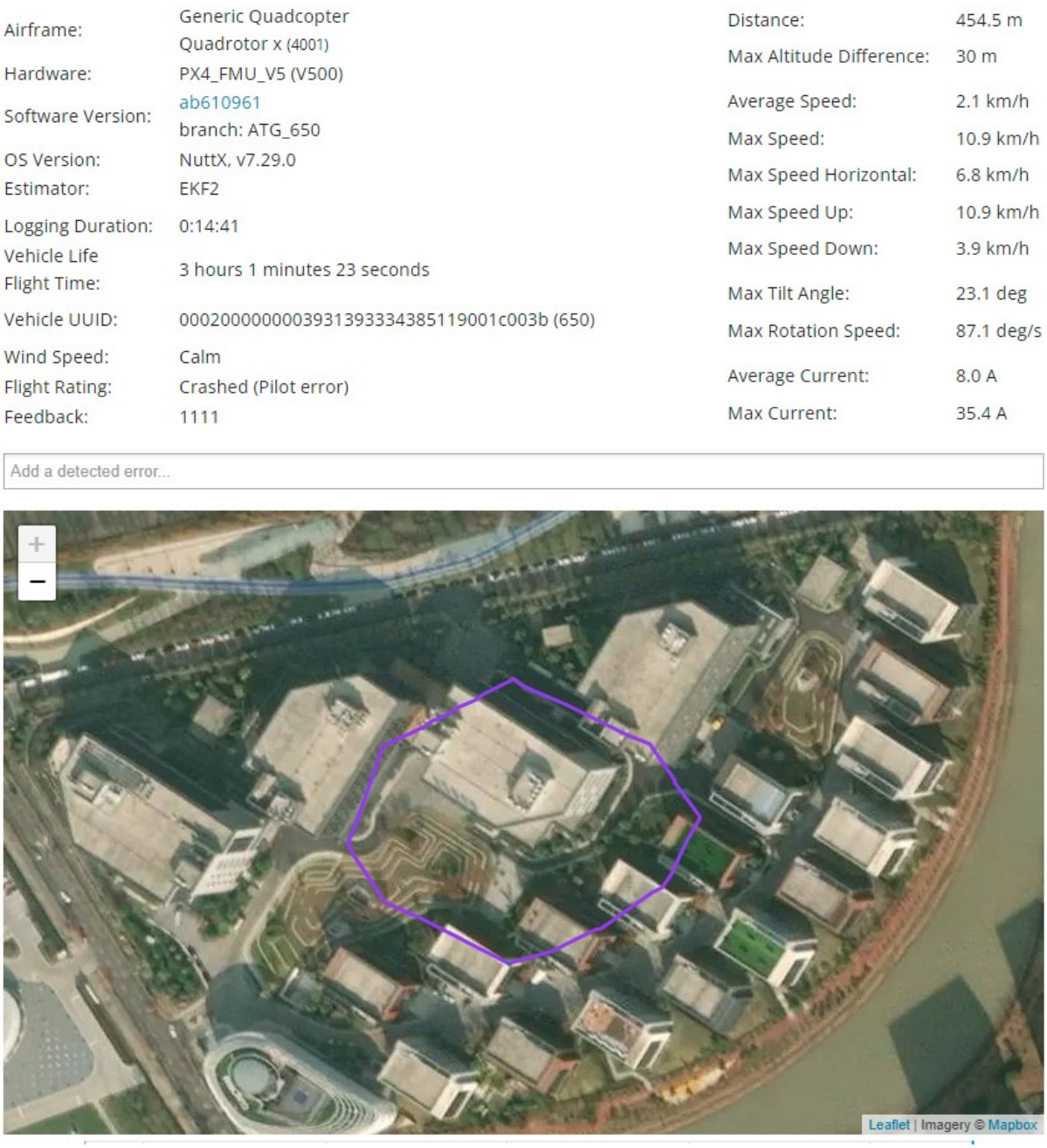


Figure 3.10. Real-World PX4 Autopilot Flight Repository Flight Log. Adopted from [95].

repository [96] offers a python script that can convert the .ulog file to a CSV file that can be inputted to the Anomaly Detection Neural Network. Flight logs scripts were downloaded through QGroundControl using the inbuilt Analyze Flight Tool and were converted to files

with extension .csv using the ulog2csv.py script. This command outputs multiple data files that offer detailed telemetry data collected through the sensors on board the UAV but the only files of relevance to the Anomaly Detection Neural Network are the combined sensor data files that contain sensor data collected throughout the flight.

3.3.2 Training the Model

This section provides an analysis of the initial sensor data that was fed into the different Neural Network models, summaries and architectures of the different NN models and training results of the different NN models on the different types of telemetry data described above.

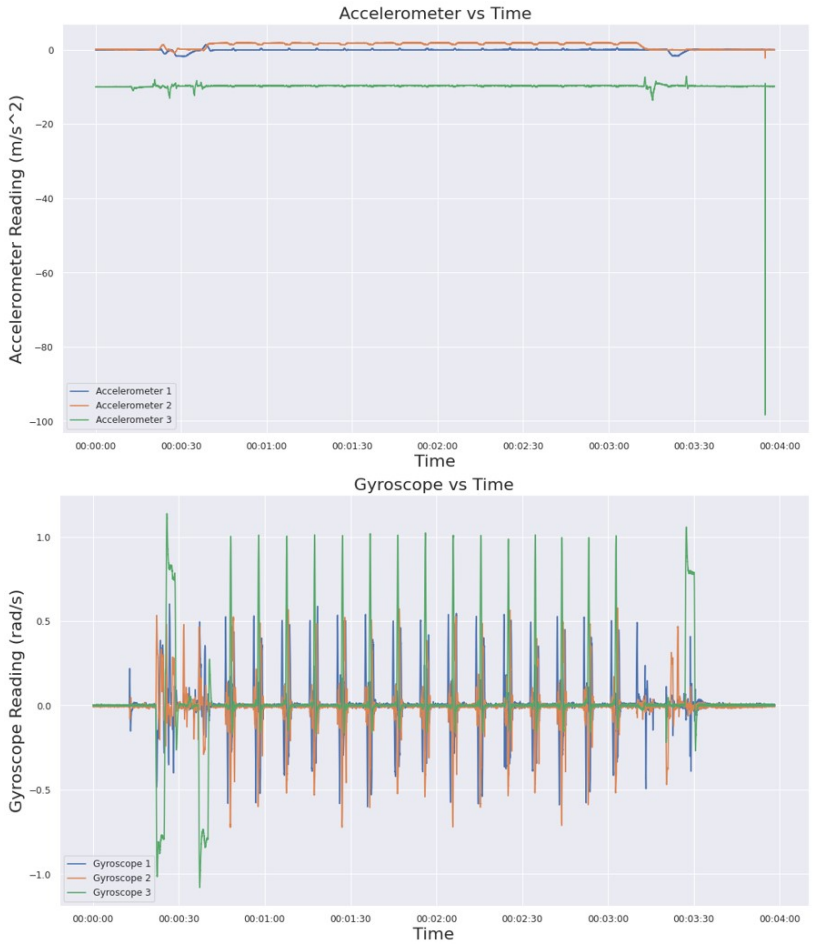


Figure 3.11. Accelerometer and Gyroscope Data for Circular Pattern Flight

Combined Sensor Data

Figure 3.11 contains the whole simulated circular flight's telemetry sensor data which included the injected anomalies for the flight. While looking at the Accelerometer vs Time graph, it is not inherently clear as to which sequence of points are anomalies and which sequence of points are normal operation. The circular flight path begins at about 40 seconds into the flight and the format for the time column is given as $hh - mm - ss$ where hh is the hours in flight time, mm is the minutes in flight time and ss are the seconds in flight time. The circular flight plan ends at about 3 minutes and 10 seconds into the flight and anomalies were inserted just before the circular flight plan ended. While looking at the Gyroscope vs Time graph, several sequences of gyroscope data don't follow a particular pattern but only data after 3 minutes included the anomaly. The initial flight data from the start of the graphs to 30 seconds had variations as compared to the seemingly similar data up to 3 minutes due to the UAV configuring it's path to begin the circular flight plan.

Figure 3.12 contains the whole simulated zig-zag flight's telemetry sensor data which included the injected anomalies for the flight. While looking at the Accelerometer vs Time graph, it is again not inherently clear as to which sequence of points are anomalies and which sequence of points are normal operation. The zig-zag flight path begins at about 30 seconds into the flight and the format for the time column is given as $hh - mm - ss$ where hh is the hours in flight time, mm is the minutes in flight time and ss are the seconds in flight time. The zig-zag flight plan ends at about 2 minutes and 45 seconds into the flight and anomalies were inserted at around 2 minutes and 38 seconds in the flight. While looking at the Gyroscope vs Time graph, several sequences of gyroscope data don't follow a particular pattern but only data after 2 minutes and 38 seconds included the anomaly. The initial flight data from the start of the graphs to 30 seconds only had variations due to the UAV taking off and trying to reach the safe mission altitude of 20 meters above ground level. Flight data till just up to 2 minutes included the zig-zag maneuvers and then until 2 minutes and 12 seconds, the UAV realigned itself for the next set of zig-zag maneuvers.

Figure 3.13 contains the whole real world test flight's telemetry sensor data which included the pilot error crash for the flight. While looking at the Accelerometer vs Time

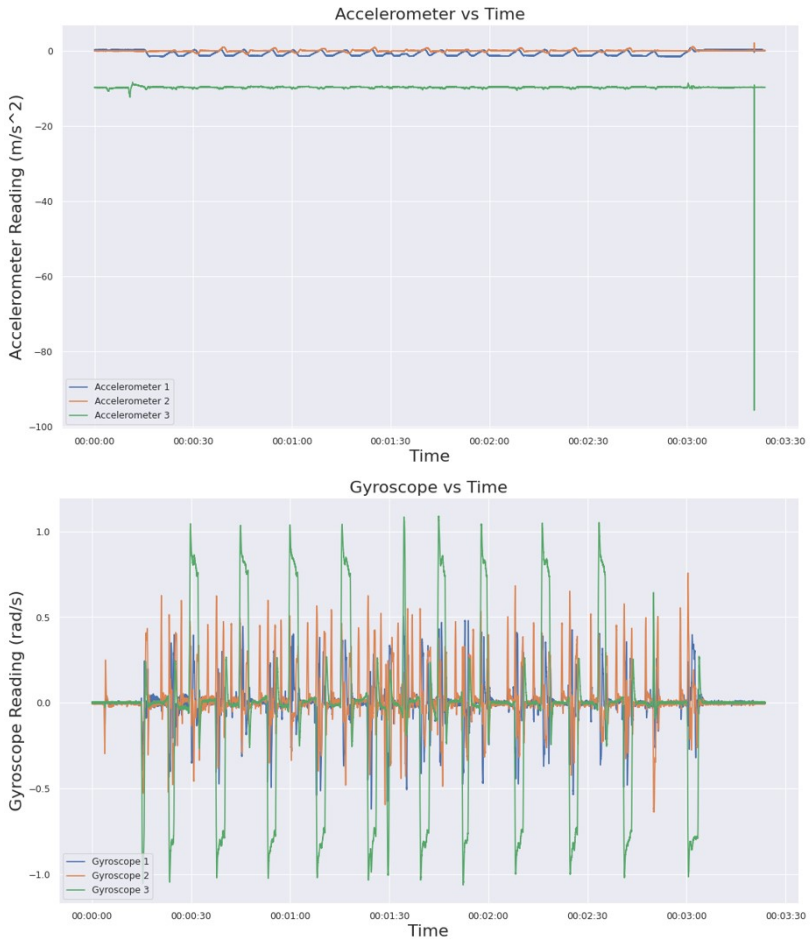


Figure 3.12. Accelerometer and Gyroscope Data for Zig-Zag Pattern Flight

graph, it is again not inherently clear as to which sequence of points are anomalies and which sequence of points are normal operation. The pilot of the UAV starts performing major maneuvers only 8 and a half minutes into the flight. Up till that point the pilot is only raising the altitude of the UAV to get to a safe mission height of 68 meters. As the pilot described that the crash only occurred towards the end of the flight it is easy to say that the last couple datapoints are the anomalies that occurred moments before the crash, but there are several major spikes in accelerometer data throughout its operation which could have been precursors to the crash. The format for the time column is given as $ddhh - mm$ where dd is the date of the flight, hh is the hours of the flight and mm are the minutes into the flight. While looking at the Gyroscope vs Time graph, it is again easy to say that the incident at around 14 minutes into the flight was the crash but the gyroscope data for the

points between 13 and 14 minutes show a very different pattern as compared to the while flight. This will be another factor that the Anomaly Detection Neural Network will have to compensate for.

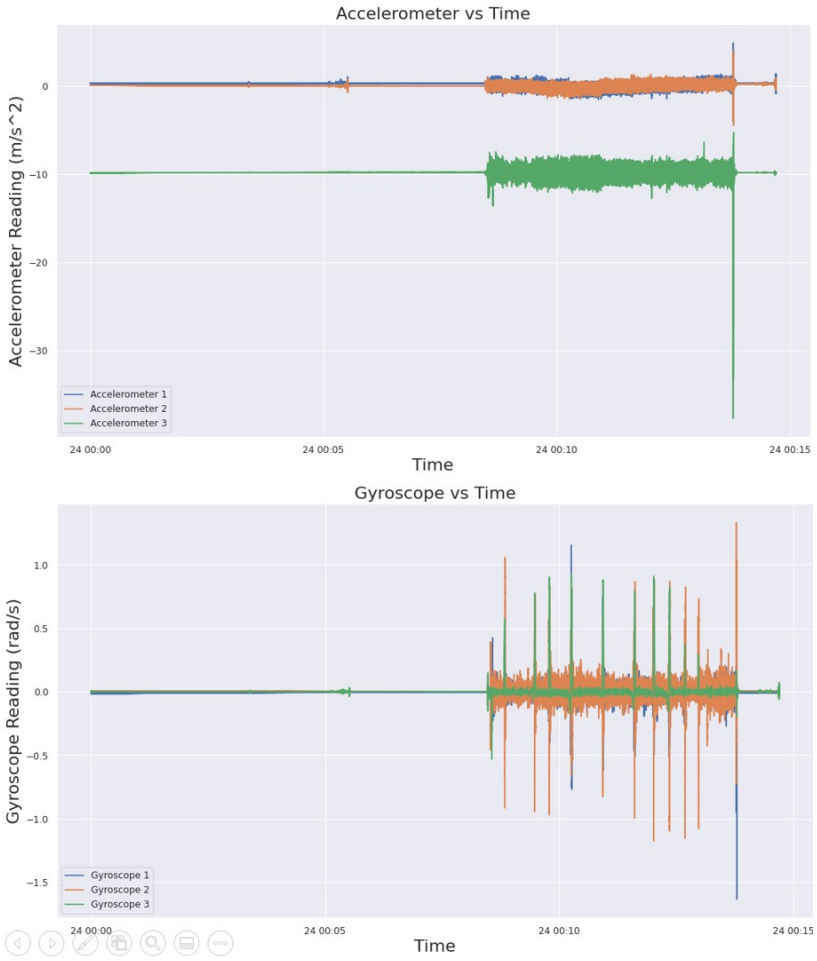
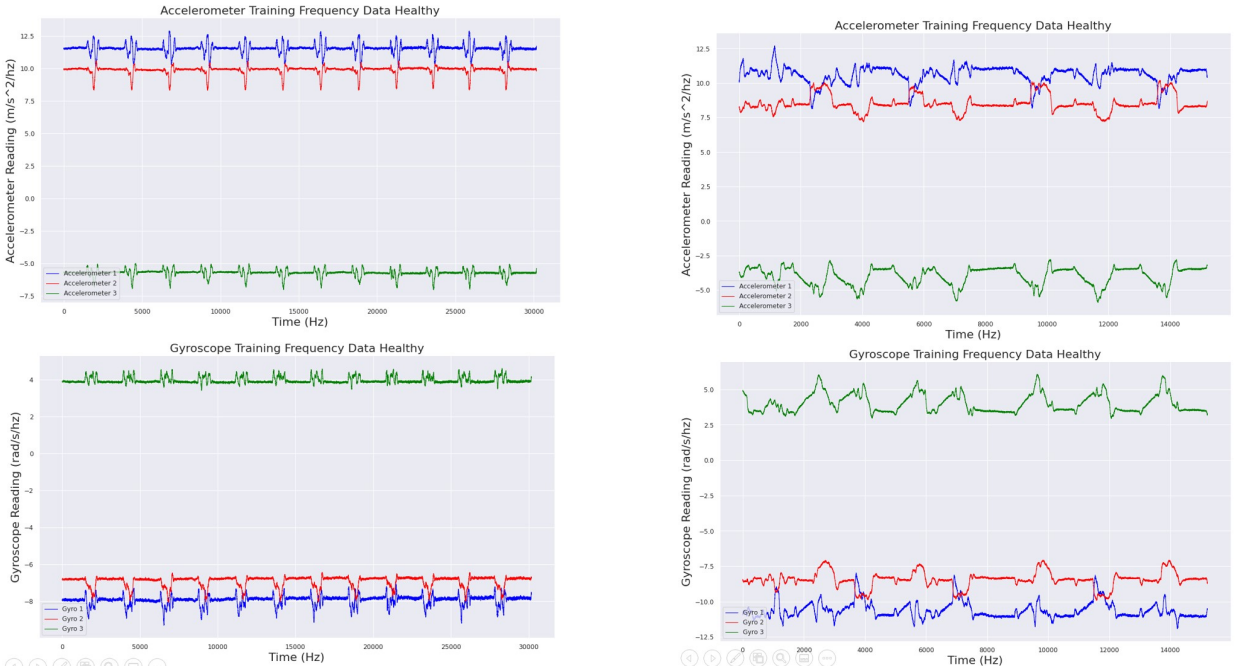


Figure 3.13. Accelerometer and Gyroscope Data for Real-World Test Flight

Healthy Sensor Data

The initial input data as described above was converted to data that had been Fast Fourier Transformed (FFT) in order to better extract the characteristics of the time series to be predicted. Figure 3.14 shows the same input data as before but it is much easier to discern differences between this data after performing Fast Fourier Transform on it [97]. Healthy sensor data contained the data before anomalies were added to the flight. The model

was only trained on healthy sensor data so that the resulting reconstruction of the inputted data in the Neural Network models contained as low of a reconstruction error as possible. This would help the Neural Network models classify anomalies in flight as compared to a model that was trained on healthy and unhealthy sensor data.



(a) Training Data for Circular Pattern Flight

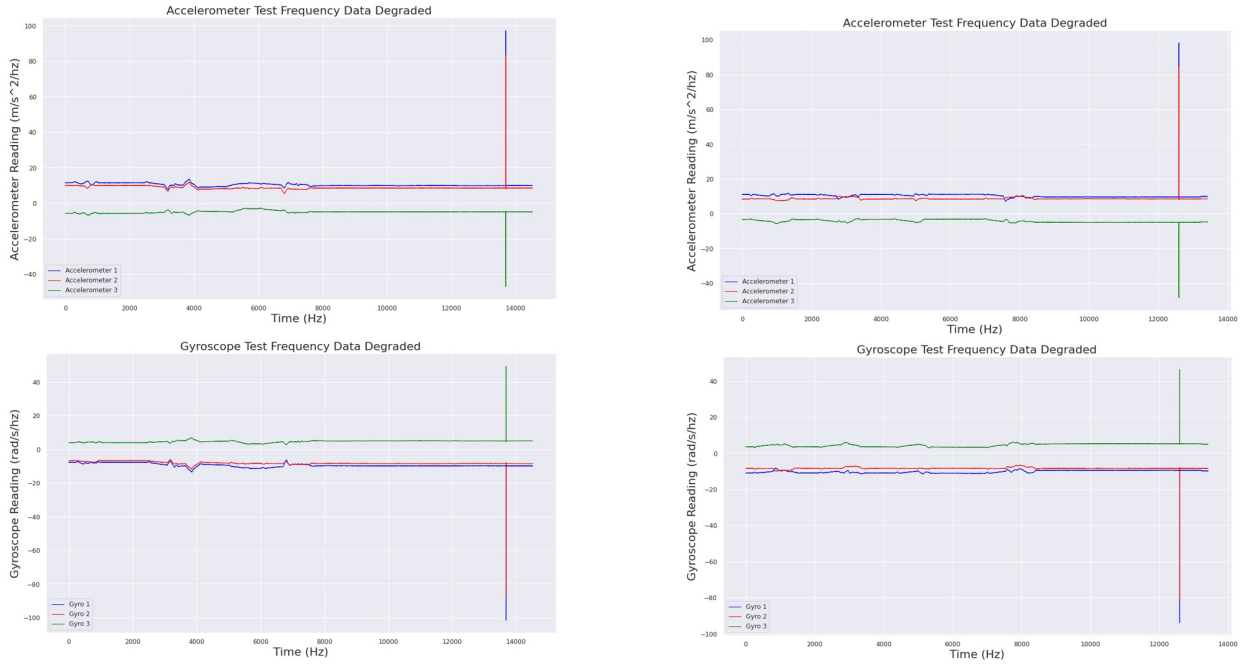
(b) Training Data for Zig-Zag Pattern Flight

Figure 3.14. Healthy Accelerometer and Gyroscope Training Data

Mixed Sensor Data

Figure 3.15 shows the last part of the input data with Fast Fourier Transform applied on it and it is again much easier to discern the differences between the previous section's normal input data and the FFT input data. Mixed sensor data contained some flight data before any anomalies occurred in flight and the data after anomalies were applied in flight all the way until the UAV landed and data logging was stopped. Towards the end of the graphs, a huge spike in accelerometer and gyroscope readings is apparent as the error between both sensors is magnified due to the UAV maneuvering back into position to start the landing process.

The subtle differences in between the different types of zig-zag and circular maneuvers in flight are also clearly discernible through these graphs.



(a) Testing Data for Circular Pattern Flight

(b) Testing Data for Zig-Zag Pattern Flight

Figure 3.15. Mixed Accelerometer and Gyroscope Testing Data

LSTM Model

The LSTM Model was kept very simple so as to not drain the computational resources of the companion computer on-board the Intelligent UAV and was limited to a single LSTM layer. It used the Adam Optimizer which is an algorithm for first-order gradient-based optimization of stochastic objective functions based on estimates of lower-order moments [98] and calculated loss according to Mean Absolute Error. The number of neurons in the hidden LSTM layer was calculated using this formula:

$$N_h = \frac{2}{3} * (N_\mathrm{i} + N_o) \tag{3.20}$$

where N_h is the number of neurons in the hidden layer, N_i is the number of input neurons and N_o is the number of output neurons. This formula follows all the rule of thumbs to determine

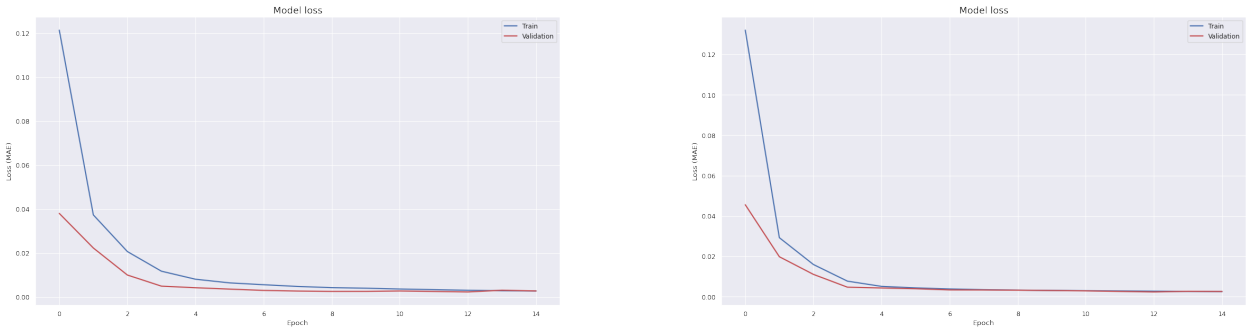
the correct number of neurons in hidden layers [99]. The overall model architecture along with the number of parameters can be seen in Figure 3.16 and a detailed model description can be viewed in Table 3.2.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 1, 6)]            0
_________________________________________________________________
lstm_10 (LSTM)               (None, 1, 8)              480
_________________________________________________________________
time_distributed_10 (TimeDis (None, 1, 6)              54
=================================================================
Total params: 534
Trainable params: 534
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.16. LSTM Model Architecture for Anomaly Detection Neural Network

Table 3.2. LSTM Model Architecture

Model Name	LSTM
Number of Layers	3
Activation Function	ReLU
Layer 1	LSTM Layer with 8 Neurons
Optimizer	Adam
Training Loss	Mean Absolute Error
Epochs	15
Batch Size	75



(a) Circular Flight (b) Zig-Zag Flight

Figure 3.17. LSTM Training Results for Different Flights

Figure 3.17 shows the training results of the three different flights for the LSTM Model. All three flights reached their optimum minimum loss over 15 epochs and training was

stopped after this to avoid overfitting of the training data. The model architecture was not changed despite the different flight data and was able to adapt to new flight data without impairing performance.

Autoencoder Model

The Autoencoder Model was also kept very simple so as to not drain the computational resources of the companion computer on-board the Intelligent UAV. It contained 4 hidden layers, with 16 neurons in the first Dense layer, 4 neurons in the second Dense layer to compress the salient data features, 4 neurons in the third Dense layer and 16 neurons in the fourth Dense layer to reproduce the reconstructed input. It used the Adam Optimizer which is an algorithm for first-order gradient-based optimization of stochastic objective functions based on estimates of lower-order moments [98] and calculated loss according to Mean Absolute Error. The number of neurons in the hidden dense layers was set after hyper-parameter tuning of the model.

The overall model architecture along with the number of parameters can be seen in Figure 3.18 and a detailed model description can be viewed in Table 3.3.

Layer (type)	Output Shape	Param #
input_12 (InputLayer)	[(None, 1, 6)]	0
dense_31 (Dense)	(None, 1, 16)	112
dense_32 (Dense)	(None, 1, 4)	68
dense_33 (Dense)	(None, 1, 4)	20
dense_34 (Dense)	(None, 1, 16)	80
time_distributed_11 (TimeDis	(None, 1, 6)	102

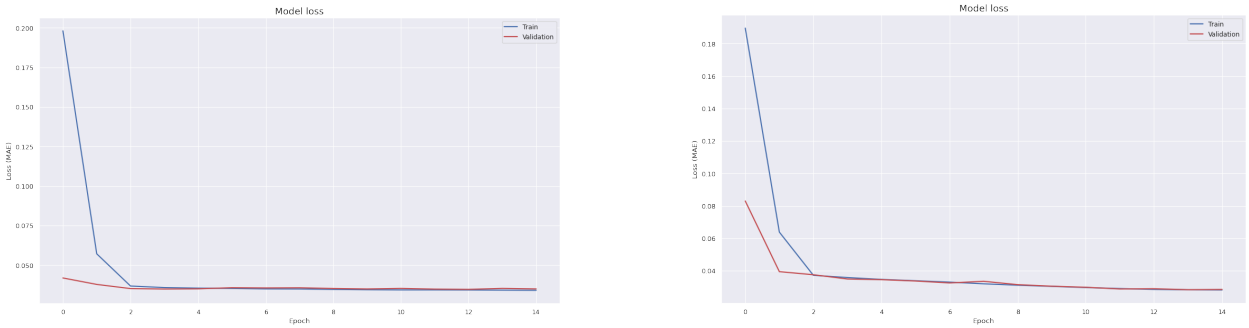
Total params: 382
Trainable params: 382
Non-trainable params: 0

Figure 3.18. Autoencoder Model Architecture for Anomaly Detection Neural Network

Figure 3.19 shows the training results of the three different flights for the Autoencoder Model. All three flights reached their optimum minimum loss over 15 epochs and training was stopped after this to avoid overfitting of the training data. The model architecture was

Table 3.3. Autoencoder Model Architecture

Model Name	Autoencoder
Number of Layers	6
Activation Function	ReLU
Layer 1	Dense Layer with 16 Neurons
Layer 2	Dense Layer with 4 Neurons
Layer 3	Dense Layer with 4 Neurons
Layer 4	Dense Layer with 16 Neurons
Optimizer	Adam
Training Loss	Mean Absolute Error
Epochs	15
Batch Size	75



(a) Circular Flight (b) Zig-Zag Flight

Figure 3.19. LSTM Training Results for Different Flights

not changed despite the different flight data and was able to adapt to new flight data without impairing performance.

Autoencoder-LSTM Model

The Autoencoder-LSTM Model was also kept very simple so as to not drain the computational resources of the companion computer on-board the Intelligent UAV. It contained 4 hidden layers and 1 repeat vector, with 16 neurons in the first LSTM layer, 4 neurons in the second LSTM layer to compress the salient data features, 4 neurons in the third LSTM layer and 16 neurons in the fourth LSTM layer to reproduce the reconstructed input. It used the Adam Optimizer which is an algorithm for first-order gradient-based optimization of stochastic objective functions based on estimates of lower-order moments [98] and calculated

loss according to Mean Absolute Error. The number of neurons in the hidden LSTM layers was set after hyper-parameter tuning of the model.

The overall model architecture along with the number of parameters can be seen in Figure 3.20 and a detailed model description can be viewed in Table 3.4.

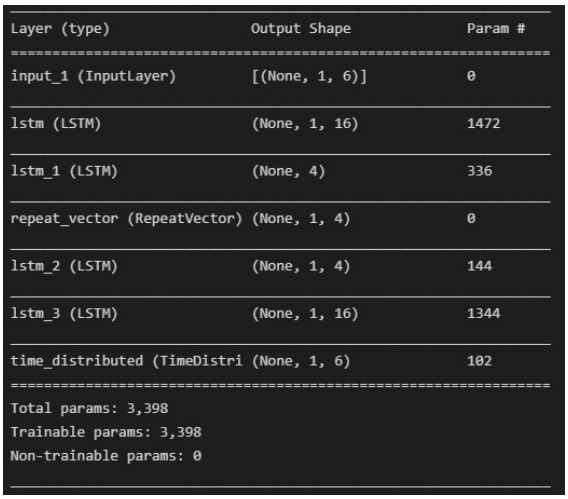
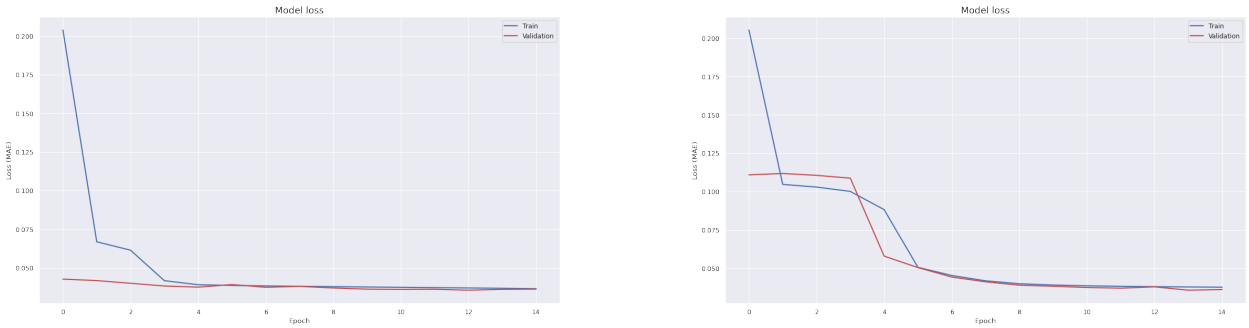


Figure 3.20. Autoencoder Model Architecture for Anomaly Detection Neural Network

Table 3.4. Autoencoder-LSTM Model Architecture

Model Name	Autoencoder-LSTM
Number of Layers	7
Activation Function	ReLU
Layer 1	LSTM Layer with 16 Neurons
Layer 2	LSTM Layer with 4 Neurons
Layer 3	Repeat Vector
Layer 4	LSTM Layer with 4 Neurons
Layer 5	LSTM Layer with 16 Neurons
Optimizer	Adam
Training Loss	Mean Absolute Error
Epochs	15
Batch Size	75

Figure 3.21 shows the training results of the three different flights for the Autoencoder-LSTM Model. All three flights reached their optimum minimum loss over 15 epochs and training was stopped after this to avoid overfitting of the training data. The model archi-



(a) Circular Flight (b) Zig-Zag Flight

Figure 3.21. Autoencoder-LSTM Training Results for Different Flights

tecture was not changed despite the different flight data and was able to adapt to new flight data without impairing performance.

3.4 Main Results

This section compares the reconstruction performance of all three Neural Network models in all three different test cases. It also compares the error threshold that each of the Neural Network models set for their respective reconstruction data along with details of their Anomaly Detection performance on-board The Intelligent UAV.

3.4.1 Reconstruction Performance

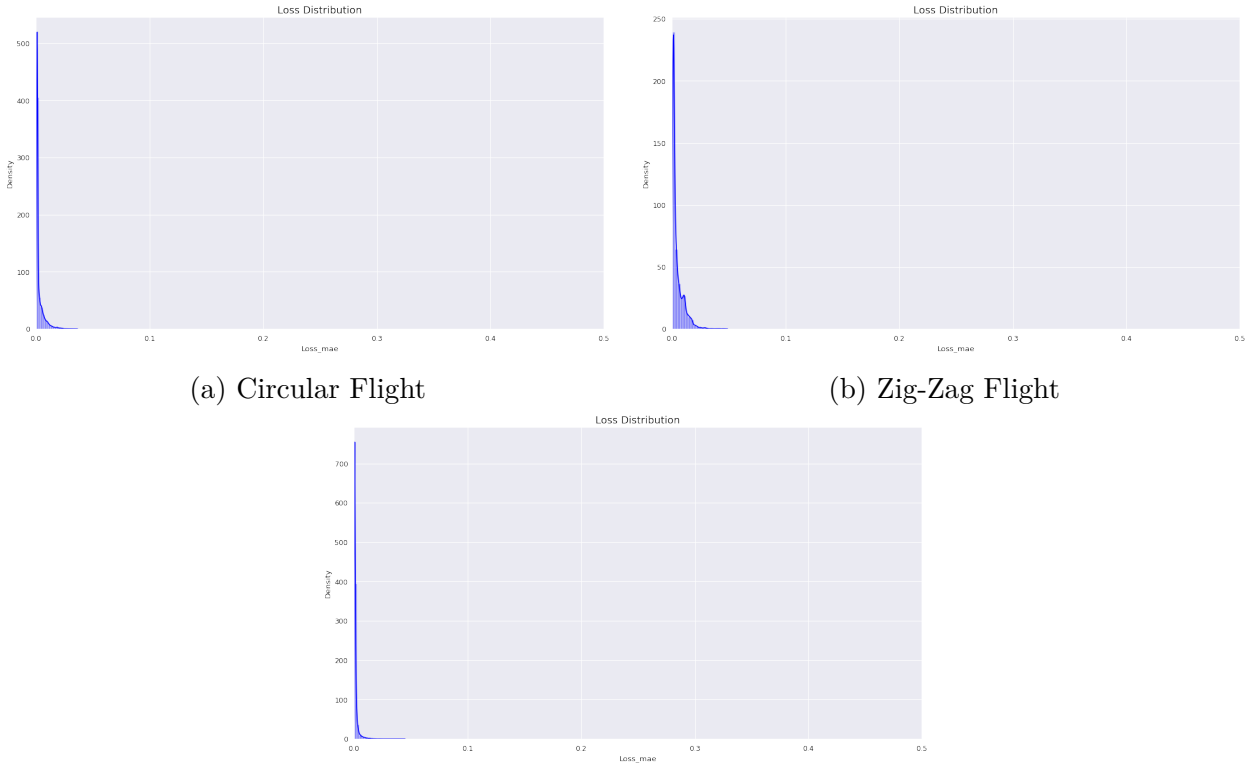
After the model is trained, the distribution of the calculated loss can be found to determine model feasibility and this loss is used to determine a threshold value for the entire model. The Mean Absolute Error between the model's prediction and the ground truth data is first calculated for the training dataset, which is the same data the model was trained to predict, to determine how accurate the model is.

Algorithm 1 shows how this process was carried out in the Anomaly Detection Neural Network to yield the distribution of the calculated loss in the training set. The same algorithm was also called on the testing set after an appropriate threshold was set.

Figure 3.22 showcases the associated distribution of the calculated losses across the different training sets for the LSTM model. The maximum loss density for the circular flight,

Algorithm 1: Reconstruction Loss Algorithm Pseudocode

1 Predict training data using the trained model
2 Reshape predictions into time series data
3 Line up predictions time series index with the training data index
4 Calculate the mean of the absolute difference between Predicted Data and Actual Data
5 Set this value as the threshold value



(a) Circular Flight

(b) Zig-Zag Flight

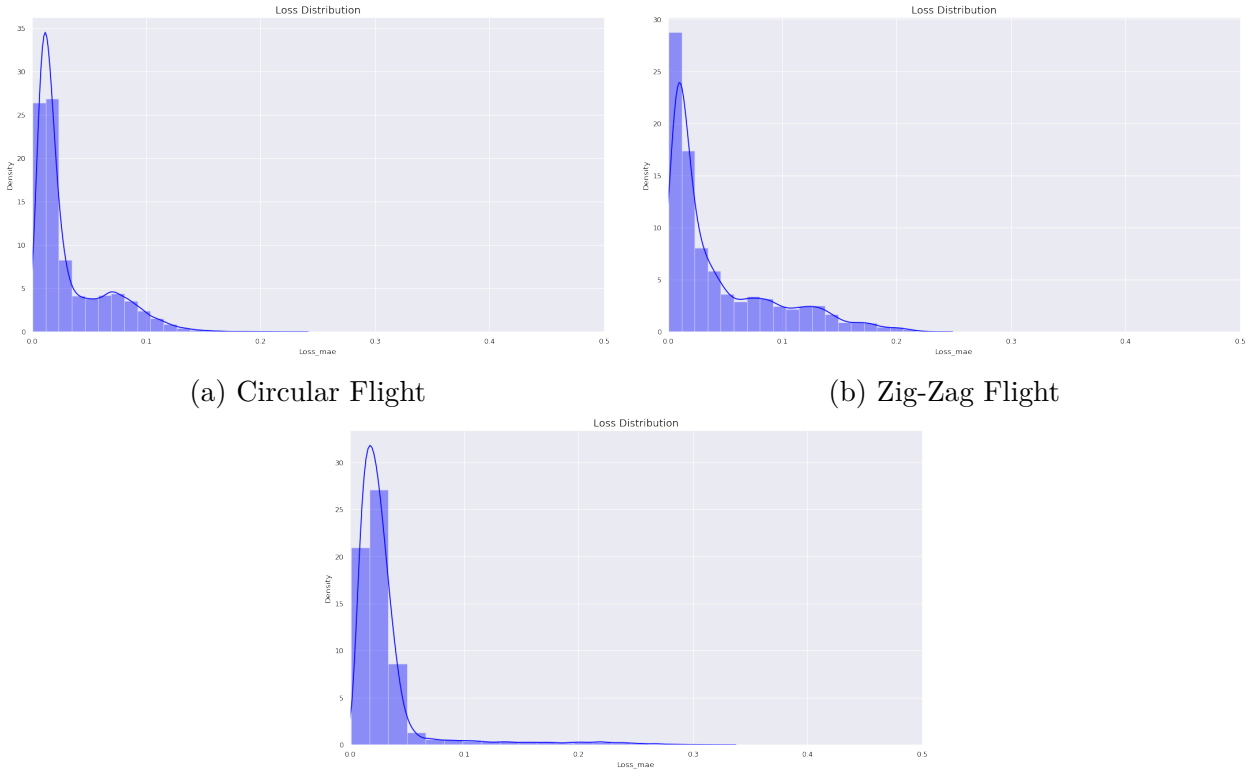
(c) Real-World Flight

Figure 3.22. LSTM Reconstruction Losses for Different Flights

as seen in Figure 3.22a, under the LSTM Anomaly Detection Neural Network (ADNN) was concentrated in the beginning of the dataset at MAE = 0.0153 and indicates a very good learning of the base model.

The maximum loss density for the zig-zag pattern flight, as seen in Figure 3.22b, under the LSTM Anomaly Detection Neural Network (ADNN) was also concentrated in the beginning of the dataset at MAE = 0.0161 and indicates a very good learning of the base model.

The maximum loss density for the real world test flight, as seen in Figure 3.22c, under the LSTM Anomaly Detection Neural Network (ADNN) was also concentrated in the beginning of the dataset at MAE = 0.0097 and indicates a very good learning of the base model.



(a) Circular Flight

(b) Zig-Zag Flight

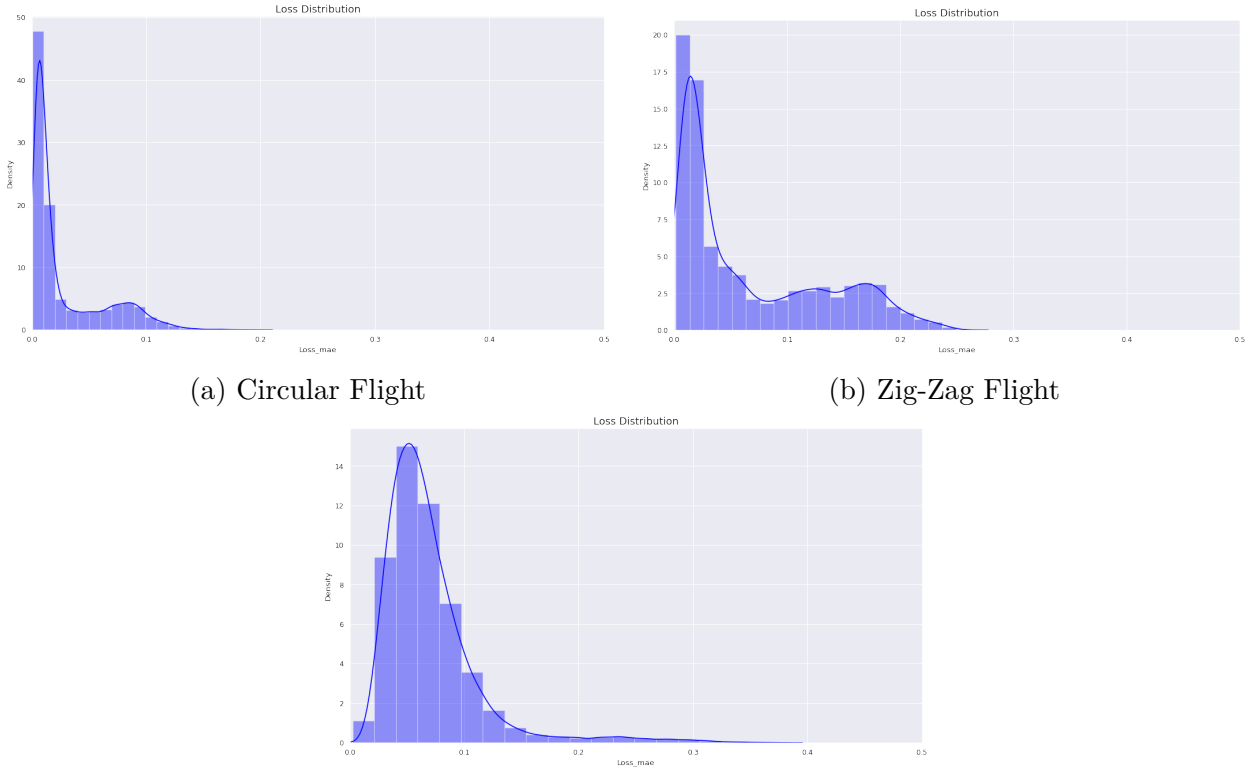
(c) Real-World Flight

Figure 3.23. Autoencoder Reconstruction Losses for Different Flights

Figure 3.23 showcases the associated distribution of the calculated losses across the different training sets for the Autoencoder model. The maximum loss density for the circular flight, as seen in Figure 3.23a, under the Autoencoder Anomaly Detection Neural Network (ADNN), was more spread out as compared to the LSTM network but is still heavily right skewed, indicating much smaller reconstruction losses.

The maximum loss density for the zig-zag pattern flight, as seen in Figure 3.23b, under the Autoencoder ADNN, shows a similar trend to the circular flight. It was more spread out as compared to the LSTM network but is still heavily right skewed, indicating much smaller reconstruction losses.

The maximum loss density for the real world test flight, as seen in Figure 3.23c, under the Autoencoder ADNN, shows a similar trend as the above two flights. It was less spread out as the previous two flights loss distribution but was more concentrated than the LSTM network.



(a) Circular Flight

(b) Zig-Zag Flight

(c) Real-World Flight

Figure 3.24. Autoencoder-LSTM Reconstruction Losses for Different Flights

Figure 3.24 showcases the associated distribution of the calculated losses across the different training sets for the Autoencoder-LSTM model. The maximum loss density for the circular flight, as seen in Figure 3.24a, under the Autoencoder-LSTM ADNN, was spread out the most with more reconstruction errors spanning more than MAE > 0.1 but it was still heavily skewed right with most of the concentration of loss at the start of the x-axis.

The maximum loss density for the zig-zag pattern flight, as seen in Figure 3.24b, under the Autoencoder-LSTM ADNN, was spread out more than the previous two neural network models and showed the least right skewed graph as compared to the previous models performance on the same flight data. The reconstruction loss maxed out at MAE $= 0.26$.

The maximum loss density for the real world test flight, as seen in Figure 3.24c, under the Autoencoder-LSTM ADNN, contained the highest reconstruction losses of any of the neural network models as the right skew was concentrated between MAE 0 to 0.1. The maximum reconstruction loss also reached till MAE = 0.39.

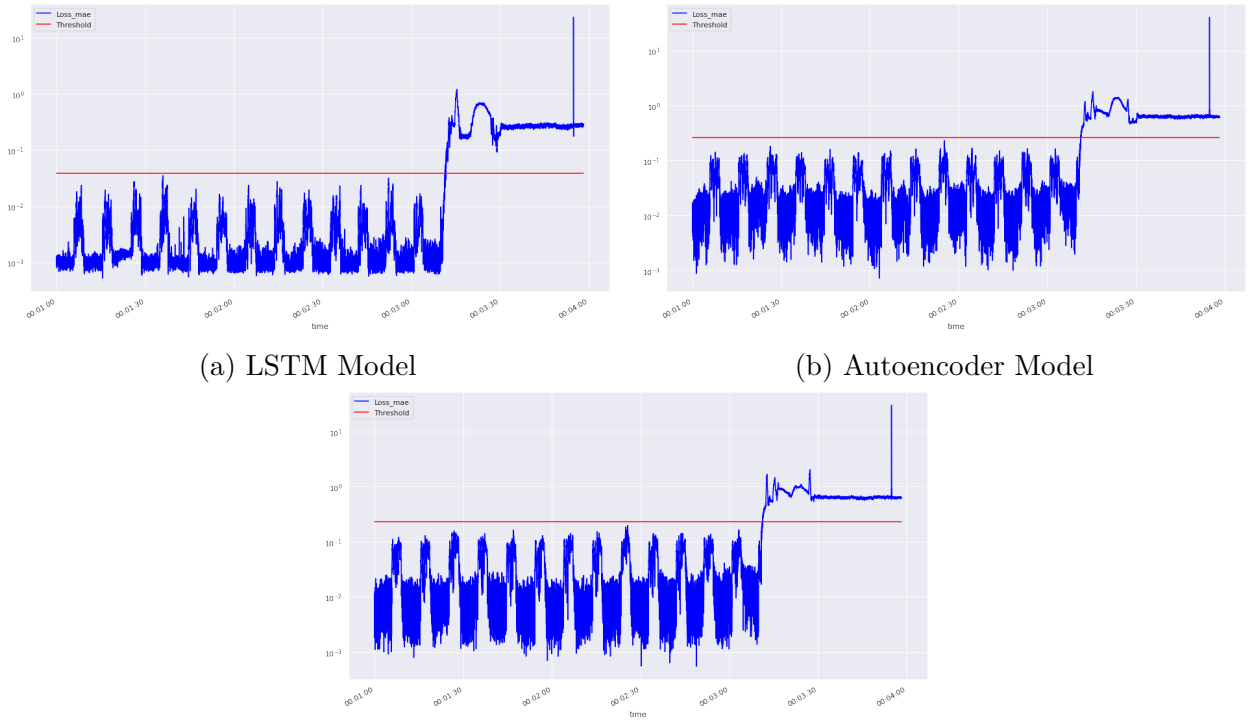
While having a large reconstruction loss might show that the model is not learning the data well, it was preferred compared to having a model that can recreate the data perfectly as this would lead to a very sensitive anomaly detection neural network that can only recognize the training data perfectly. A good anomaly detection neural network needs some kind of reconstruction loss to establish a baseline in the quality of data.

3.4.2 Error Threshold

Once the distribution of the loss function was plotted, the anomaly threshold could be set to be inclusive of the noise or disturbance in the reconstructed data values. This would ensure that the Anomaly Detection Neural Network did not catch any false positives in the dataset and would only report an anomaly once there was reconstruction loss that was deemed beyond the threshold for reconstruction loss of a particular model.

The threshold was dynamically calculated for each neural network and for each input data set to ensure that only true positive anomalies were detected in the dataset. Figure 3.25 shows the threshold value being applied across the predicted value for the circular flight path for all three types of Neural Networks.

Table 3.5 shows the error threshold values for all of the datasets. Any reconstruction error value above these determined thresholds would be marked as an anomaly by the neural networks. The error threshold can also be taken as a single parameter to compare model performance of the Neural Network learning the training dataset. The values clearly indicate that the LSTM Neural Network seemed to have the lowest error threshold and hence learned the training dataset the best. The next best neural network was the Autoencoder Neural Network which had 6 times higher reconstruction error as compared to the LSTM model. The Autoencoder-LSTM came in last with a 7 times higher reconstruction error as compared to the LSTM model. These values indicate that the LSTM model will be the most sensitive to



(a) LSTM Model

(b) Autoencoder Model

(c) Autoencoder-LSTM Model

Figure 3.25. Threshold Values Applied on Predicted Values for the Different Neural Network Models on the Circular Pattern Flight

anomalies while avoiding false positives as compared to the Autoencoder and Autoencoder-LSTM Network.

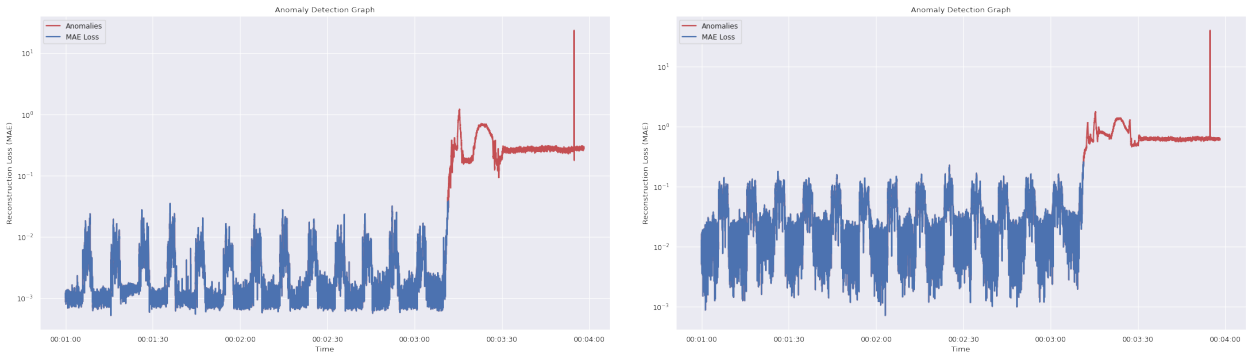
Table 3.5. Error Threshold for Neural Networks Across All Datasets

Model Name	Flight Plan Name	Error Threshold
LSTM	Circle	0.03879
	Zig-Zag	0.05199
	Real-World	0.04622
Autoencoder	Circle	0.26072
	Zig-Zag	0.2753
	Real-World	0.36107
Autoencoder-LSTM	Circle	0.23073
	Zig-Zag	0.31385
	Real-World	0.42431

3.4.3 Anomaly Detection Performance

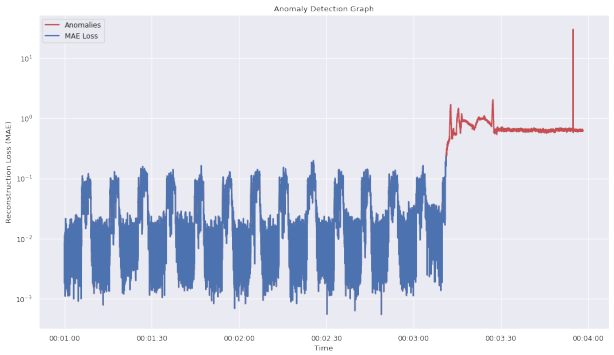
All three models correctly identified the anomalies present in all three datasets. The major difference can only be observed when the test cases are changed. When a repeating pattern is used to train the model to detect anomalies, then the anomaly detector is able to predict when the anomalies begin down to the millisecond when the sensors start to send degraded values to the flight computer. For the test case that had changing actions at every turn, the anomaly detector was able to flag when the sensors started sending anomalous values to the flight computer but then considered all the next few values as normal operation. This is due to the fact that the healthy data was random and led to a much higher threshold than a case with a predictable path. The anomaly detector detected another set of anomalies when the values spiked in this degraded sensor data setting off two alarms in the same test case. For the last test case, the data suggests that the crash in the flight was purely caused by pilot error as suggested by the pilot as there were no discernible signs of faulty sensor measurement before the crash occurred.

Figure 3.26 shows the performance of the three different kinds of Anomaly Detection Neural Networks on the circular flight pattern. These graphs plot the log of the Mean Absolute Error against the flight time. The Autoencoder (Figure 3.26b) and Autoencoder-LSTM (Figure 3.26c) models seem to reconstruct the data similarly whereas the LSTM (Figure 3.26a) model takes a different approach. The LSTM model has the least amount of reconstruction error and is able to correctly recreate the first anomaly in the circular flight path test data that occurs 3 minutes and 8 seconds into the flight and appears as the first red spike. The LSTM model tries to recreate the next few sensor values after this reading but has a much higher error in doing so and continues regarding every value after this as an anomalous sensor value. The LSTM model is then able to correctly predict the landing sequence which is signified as the large red spike towards the end of the graph. The sensor values stayed degraded till after landing in this test case and the model accurately predicted the whole test case. The Autoencoder and Autoencoder-LSTM showcase this phenomenon as well but detect the anomaly a whole 2 seconds after it had been spotted in the LSTM model at 3 minutes and 10 seconds. The graphs for both these models follow a similar



(a) LSTM Model

(b) Autoencoder Model

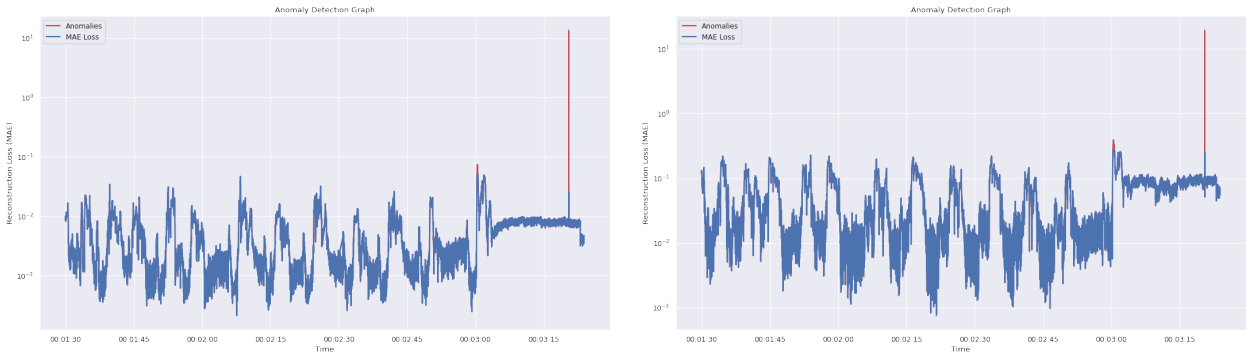


(c) Autoencoder-LSTM Model

Figure 3.26. Anomaly Detection on the Circular Pattern Flight

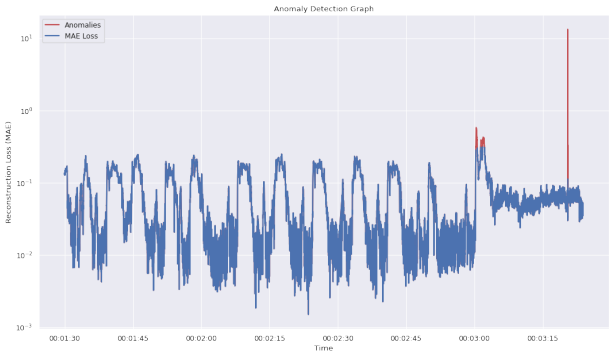
pattern and are less precise than the graphs reconstructed by the LSTM model due to their high training loss and higher reconstruction error threshold.

Figure 3.27 shows the performance of the three different kinds of Anomaly Detection Neural Networks on the zig-zag flight pattern. These graphs plot the log of the Mean Absolute Error against the flight time. The Autoencoder (Figure 3.27b), Autoencoder-LSTM (Figure 3.27c) and the LSTM (Figure 3.27a) model seem to reconstruct the data similarly as compared to the previous test case. Although the Autoencoder-LSTM model is not able to achieve a MAE of 10^{-3} as compared to the LSTM and Autoencoder models. The LSTM model again has the least amount of reconstruction error and is able to correctly recreate the first anomaly in the zig-zag flight path test data that occurs 3 minutes into the flight and appears as a small red spike. The LSTM model tries to recreate the next few sensor values after this reading but has a much higher error in doing so as compared to the values ranging from 1 minute to 3 minutes into the flight. Despite this, the next few values



(a) LSTM Model

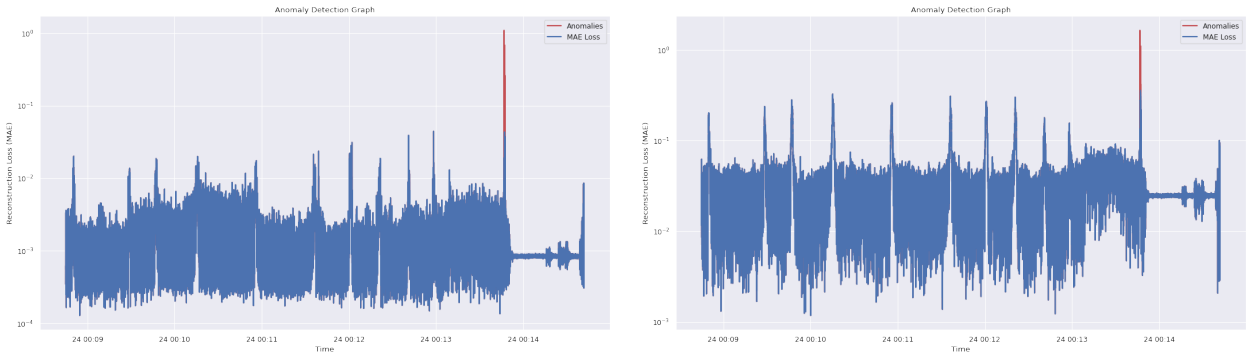
(b) Autoencoder Model



(c) Autoencoder-LSTM Model

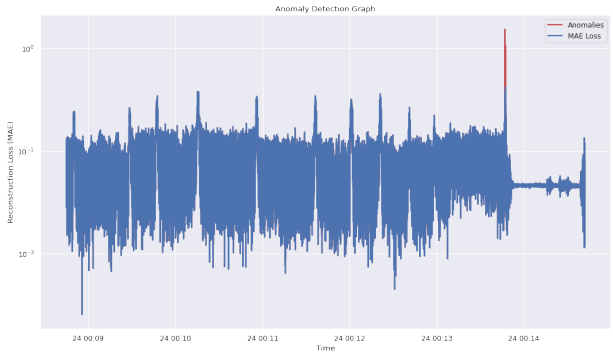
Figure 3.27. Anomaly Detection on the Zig-Zag Pattern Flight

are regarded as normal operation due to the nature of the mixed data fed into the model which caused higher reconstruction errors leading to these values not being detected. The LSTM model is then able to correctly predict the landing sequence. As the UAV begins the land, the sensor data then displays a huge anomaly which the model correctly identifies and flags as an anomaly. The Autoencoder and Autoencoder-LSTM showcase this phenomenon as well as their reconstruction error keeps rising higher and higher after the 3 minutes mark but they are able to attribute the spike to the landing sequence as well so the previous spike was not identified as an anomaly. When the biggest spike shows up towards the end of the flight, both models identify the same anomaly as the LSTM model. Due to the random flight characteristics of this dataset, the model was not able to correctly recreate the healthy data which led to it setting a higher threshold and missing an anomaly in the middle between the first anomaly and the landing sequence.



(a) LSTM Model

(b) Autoencoder Model



(c) Autoencoder-LSTM Model

Figure 3.28. Anomaly Detection on the Real World Flight

Figure 3.28 shows the performance of the three different kinds of Anomaly Detection Neural Networks on the real-world test flight. These graphs plot the log of the Mean Absolute Error against the flight time. The Autoencoder (Figure 3.28b), Autoencoder-LSTM (Figure 3.28c) and the LSTM (Figure 3.28a) model seem to reconstruct the data similarly just like the previous test case. Although the Autoencoder and the Autoencoder-LSTM models are not able to achieve a MAE below of 10^{-3} as compared to the LSTM model. The LSTM model again has the least amount of reconstruction error and is able to correctly recreate the whole flight despite it being controlled by a human pilot. The LSTM model correctly identifies when the crash takes place at around 13 mins and 50 seconds into the flight. The Autoencoder and Autoencoder-LSTM models follow a similar trend and are able to discern when the crash takes place as well due to the huge reconstruction error spike at that moment. Through this test case, the human operator's cause for the crash which was listed as pilot

error can be confirmed as none of the sensors seemed to have any anomalies except till the moment of the crash.

In conclusion, the LSTM Anomaly Detection Neural Network, despite being the most simple amongst the three neural networks with only a single hidden layer, seems to perform the best and has the least reconstruction error making it more prone to detect small anomalies in flight as compared to the Autoencoder or the Autoencoder-LSTM network. It was able to detect anomalies in the circular flight pattern, the zig-zag flight pattern and the human piloted real world test flight that ended with a crash. Due to its lower threshold, the LSTM model's sensitivity has the highest chance to pick up and report an anomaly to the user before a major failure.

4. SUMMARY

4.1 Conclusion

The problem addressed by this study was one of creating a modular and robust UAV platform that could be adapted for multiple use cases and could be used as a leader UAV in a multi-robot swarm due to its computational power and suite of sensors. The main hurdles to enable further research on this platform were to solve real-time perception to enable computer vision applications on-board the UAV such as object detection, motion estimation, obstacle avoidance and 3D scene reconstruction and to increase system safety and reliability through anomaly detection to predict system failure before it can occur.

Perception was enabled on the UAV through a Convolutional Neural Network running on the companion computer using the RGB-D camera to capture its surroundings. The performance of two different types of state-of-art object detection algorithms, YOLOv5 and Mobilenet-SSDv2, meant to run on Embedded Systems were compared along with their deployment using a hardware inference accelerator to increase frames per second of real time object detection. Both Neural Network models were also trained on a limited custom image dataset to judge their training performance and time taken to train. Data augmentation was applied to this dataset during training to further improve the performance of both of the models. The mAP, precision and recall were compared to further determine which algorithm performed better on the Intelligent UAV. Real-time video from the UAV system was also used to compare their maximum frames per second at HD resolutions including their performance increase gained by using a hardware inference accelerator. The higher the frames per second, the faster the UAV system can travel. The YOLOv5 algorithm was determined to give the highest precision with the least amount of training time along with the highest frames per second for real time object detection.

Next, anomaly detection on-board the Intelligent UAV was introduced through three different types of neural network models, LSTM, Autoencoder and Autoencoder-LSTM. The Intelligent UAV's external flight sensors and flight computer that would be sending the live telemetry data to the on-board computer were discussed along with a brief overview of the contents of the telemetry data. The three different types of reconstruction error losses,

MAE, MSE and RMSE were discussed along with methods to set a threshold for the neural network's output of the reconstructed data. Three different mission plans, one that followed an established pattern, one with a random pattern and one human controlled flight were established and used as a training dataset for each of the neural network algorithms. It was determined that the neural network would be very good at learning a fixed pattern so the latter two flight plans were used to check how consistent the neural network was at predicting anomalies even if there was no established pattern. Sensor failure was deemed as the only reasonable failure to inject into the anomaly detection system as major component failure would lead to an instantaneous crash. Input data with and without Fast Fourier Transform performed on it was compared to establish the improvement of model performance due to this step. The models architecture was shown to reestablish the simplicity of the models to ensure that they did not consume a lot of computational power. Reconstruction losses and error thresholds were compared for all three different networks on all three flights to judge the internal networks reconstruction errors. The LSTM model seemed to have the best performance despite being the simplest of all three networks. All the networks were able to detect the anomalies that occurred during the flight and even plotted the exact time the anomaly was injected into the flight.

With the establishment of these systems running in real-time using an embedded system without discrete graphics and without computational help from a ground control station, it can be concluded that this platform is ready for further development of new use-cases some of which are mentioned in the next section.

4.2 Future Work

Future work for the Intelligent UAV can be:

- To create a vision-based, GPS-denied trajectory generation and path planning algorithm that would allow the UAV to autonomously carry out flight plans. Using the object detection on board, contextual AI could also be added to the path planning algorithm so that the user only needs to tell the UAV what object it needs to look for

and what to do with it for the UAV to plan an entire mission by itself in an unknown environment.

- Train more models to reinforce the learning of the anomaly detection algorithm onboard the Intelligent UAV so that a better model representing healthy sensor data is generated. This will enable the UAV to flag anomalies that were too small to catch by the current model.
- Simultaneous Localization and Mapping (SLAM) could be paired with the RGB-D camera on the Intelligent UAV to create 3 dimensional depth maps or 3D reconstruction of the environment.
- A use case with multiple other UAVs for distributed tasks in drone swarms. The Intelligent UAV could act as a leader UAV in the swarm and distribute tasks by providing other UAVs with trajectory information.
- A use case with a heterogeneous team of robots such as UGVs and UAVs to develop inter-robot collaboration. The UGV could act as a ground transport vehicle when the UAV is not in use and the UAV could provide data on the terrain ahead of the current path of the UGV.

REFERENCES

[1] E. Garcia, M. A. Jimenez, P. G. De Santos, and M. Armada, "The evolution of robotics research," *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 90–103, 2007. DOI: 10.1109/MRA.2007.339608.

[2] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník, "Artificial intelligence for long-term robot autonomy: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4023–4030, 2018. DOI: 10.1109/LRA.2018.2860628.

[3] B. Apolloni, A. Ghosh, F. Alpaslan, and S. Patnaik, *Machine Learning and Robot Perception*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2005, ISBN: 9783540265498. [Online]. Available: https://books.google.com/books?id=VeGd71RiU3wC.

[4] D. Ballard and C. Brown, *Computer Vision*. Prentice-Hall, 1982, ISBN: 9780131653160. [Online]. Available: https://books.google.com/books?id=EfRRAAAAMAAJ.

[5] T. Morris, *Computer Vision and Image Processing*, English. United Kingdom: Palgrave Macmillan Ltd, 2004, ISBN: 0333994515.

[6] B. Dhillon, *Robot System Reliability and Safety: A Modern Approach*. CRC Press, 2015, ISBN: 9781498706452. [Online]. Available: https://books.google.com/books?id=hHd3CAAAQBAJ.

[7] C. Howard, "Uav command, control & communications," *Military & Aerospace Electronics, militaryaerospace.com*, 2013. [Online]. Available: https://www.curtisswrightds.com/news/articles/mae-jul13-uav-command-control-communications.html.

[8] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner, "An autonomous multi-uav system for search and rescue," in *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, ser. DroNet '15, Florence, Italy: Association for Computing Machinery, 2015, pp. 33–38, ISBN: 9781450335010. DOI: 10.1145/2750675.2750683. [Online]. Available: https://doi.org/10.1145/2750675.2750683.

[9] E. H. C. Harik, F. Guérin, F. Guinand, J.-F. Brethé, and H. Pelvillain, "Towards an autonomous warehouse inventory scheme," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8. DOI: 10.1109/SSCI.2016.7850056.

[10] T. Adão, J. Hruška, L. Pádua, J. Bessa, E. Peres, R. Morais, and J. J. Sousa, "Hyperspectral imaging: A review on uav-based sensors, data processing and applications for agriculture and forestry," *Remote Sensing*, vol. 9, no. 11, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9111110. [Online]. Available: https://www.mdpi.com/2072-4292/9/11/1110.

[11] D. Harfina, Z. Zaini, and W. Wulung, "Disinfectant spraying system with quadcopter type unmanned aerial vehicle (uav) technology as an effort to break the chain of the covid-19 virus," *Journal of Robotics and Control (JRC)*, vol. 2, no. 6, pp. 502–507, 2021, ISSN: 2715-5072. DOI: 10.18196/jrc.26129. [Online]. Available: https://journal.umy.ac.id/index.php/jrc/article/view/10462.

[12] B. Galkin, J. Kibilda, and L. A. DaSilva, "Uavs as mobile infrastructure: Addressing battery lifetime," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 132–137, 2019. DOI: 10.1109/MCOM.2019.1800545.

[13] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Future of Software Engineering (FOSE '07)*, 2007, pp. 171–187. DOI: 10.1109/FOSE.2007.32.

[14] L. A. Barroso, "The price of performance: An economic case for chip multiprocessing," *Queue*, vol. 3, no. 7, pp. 48–53, Sep. 2005, ISSN: 1542-7730. DOI: 10.1145/1095408.1095420. [Online]. Available: https://doi.org/10.1145/1095408.1095420.

[15] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park, *Pvanet: Lightweight deep neural networks for real-time object detection*, 2016. DOI: 10.48550/ARXIV.1611.08588. [Online]. Available: https://arxiv.org/abs/1611.08588.

[16] A. Bürkle, F. Segor, and M. Kollmann, "Towards autonomous micro uav swarms," *Journal of intelligent & robotic systems*, vol. 61, no. 1, pp. 339–353, 2011. DOI: 10.1007/s10846-010-9492-x. [Online]. Available: https://link.springer.com/article/10.1007/s10846-010-9492-x.

[17] H. Wu, H. Li, R. Xiao, and J. Liu, "Modeling and simulation of dynamic ant colony's labor division for task allocation of uav swarm," *Physica A: Statistical Mechanics and its Applications*, vol. 491, pp. 127–141, 2018, ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2017.08.094. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437117308166.

[18] C. Phan and H. H. Liu, "A cooperative uav/ugv platform for wildfire detection and fighting," in *2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing*, 2008, pp. 494–498. DOI: 10.1109/ASC-ICSC.2008.4675411.

[19] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 46–56, 2012. DOI: 10.1109/MRA.2012.2206473.

[20] A. Al-Kaff, R. Alonso, M. Osman, and A. Hussein, "Skyonyx: Autonomous uav research platform for air transportation system (atsys)," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 3550–3555. DOI: 10.1109/ITSC.2018.8569675.

[21] M. Petrlík, T. Báča, D. Heřt, M. Vrba, T. Krajník, and M. Saska, "A robust uav system for operations in a constrained environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2169–2176, 2020. DOI: 10.1109/LRA.2020.2970980.

[22] G. Balamurugan, J. Valarmathi, and V. P. S. Naidu, "Survey on uav navigation in gps denied environments," in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, 2016, pp. 198–204. DOI: 10.1109/SCOPES.2016.7955787.

[23] A. Krizhevsky, *One weird trick for parallelizing convolutional neural networks*, 2014. DOI: 10.48550/ARXIV.1404.5997. [Online]. Available: https://arxiv.org/abs/1404.5997.

[24] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556.

[25] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. DOI: 10.48550/ARXIV.1512.00567. [Online]. Available: https://arxiv.org/abs/1512.00567.

[27] S.-W. Kim, H.-K. Kook, J.-Y. Sun, M.-C. Kang, and S.-J. Ko, "Parallel feature pyramid network for object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018. [Online]. Available: https://www.mdpi.com/2078-2489/13/1/5/xml.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[29] Y.-C. Chiu, C.-Y. Tsai, M.-D. Ruan, G.-Y. Shen, and T.-T. Lee, "Mobilenet-ssdv2: An improved object detection model for embedded systems," in *2020 International Conference on System Science and Engineering (ICSSE)*, 2020, pp. 1–5. DOI: 10.1109/ICSSE50014.2020.9219319.

[30] Z. Zhang, Y. Liu, T. Liu, Z. Lin, and S. Wang, "Dagn: A real-time uav remote sensing image vehicle detection framework," *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 11, pp. 1884–1888, 2020. DOI: 10.1109/LGRS.2019.2956513.

[31] P. Nousi, I. Mademlis, I. Karakostas, A. Tefas, and I. Pitas, "Embedded uav real-time visual object detection and tracking," in *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2019, pp. 708–713. DOI: 10.1109/RCAR47638.2019.9043931.

[32] J. Deng, Z. Shi, and C. Zhuo, "Energy-efficient real-time uav object detection on embedded platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 3123–3127, 2020. DOI: 10.1109/TCAD.2019.2957724.

[33] H.-H. Wu, Z. Zhou, M. Feng, Y. Yan, H. Xu, and L. Qian, "Real-time single object detection on the uav," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 1013–1022. DOI: 10.1109/ICUAS.2019.8797866.

[34] Y. Liu and W. Ding, "A knns based anomaly detection method applied for uav flight data stream," in *2015 Prognostics and System Health Management Conference (PHM)*, 2015, pp. 1–8. DOI: 10.1109/PHM.2015.7380051.

[35] A. Keipour, M. Mousaei, and S. Scherer, "Automatic real-time anomaly detection for autonomous aerial vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 5679–5685. DOI: 10.1109/ICRA.2019.8794286.

[36] Y. Chen, B. Wang, W. Liu, and D. Liu, "On-line and non-invasive anomaly detection system for unmanned aerial vehicle," in *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, 2017, pp. 1–7. DOI: 10.1109/PHM.2017.8079160.

[37] D. Pan, L. Nie, W. Kang, and Z. Song, "Uav anomaly detection using active learning and improved s3vm model," in *2020 International Conference on Sensing, Measurement Data Analytics in the era of Artificial Intelligence (ICSMD)*, 2020, pp. 253–258. DOI: 10.1109/ICSMD50554.2020.9261709.

[38] H. Lu, Y. Li, S. Mu, D. Wang, H. Kim, and S. Serikawa, "Motor anomaly detection for unmanned aerial vehicles using reinforcement learning," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2315–2322, 2018. DOI: 10.1109/JIOT.2017.2737479.

[39] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019. DOI: 10.1109/TNNLS.2018.2876865.

[40] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, no. 2, 2004. DOI: 10.1023/B:VISI.0000029664.99615.94. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[41] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[42] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. International Conference on Image Processing*, vol. 1, 2002, pp. I–I. DOI: 10.1109/ICIP.2002.1038171.

[43] J. Du, "Understanding of object detection based on cnn family and yolo," *Journal of Physics: Conference Series*, vol. 1004, p. 012 029, Apr. 2018. DOI: 10.1088/1742-6596/1004/1/012029.

[44] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, *Cnn features off-the-shelf: An astounding baseline for recognition*, 2014. DOI: 10.48550/ARXIV.1403.6382. [Online]. Available: https://arxiv.org/abs/1403.6382.

[45] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[46] J. Du, "Understanding of object detection based on CNN family and YOLO," *Journal of Physics: Conference Series*, vol. 1004, p. 012 029, 2018. DOI: 10.1088/1742-6596/1004/1/012029. [Online]. Available: https://doi.org/10.1088/1742-6596/1004/1/012029.

[47] G. Vinod, "Cooperative Perception in Multi-agent Systems," Jul. 2021. DOI: 10.25394/PGS.14854575.v1. [Online]. Available: https://hammer.purdue.edu/articles/thesis/Cooperative_Perception_in_Multi-agent_Systems/14854575.

[48] P. Kim, "Convolutional neural network," in *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*. Berkeley, CA: Apress, 2017, pp. 121–147, ISBN: 978-1-4842-2845-6. DOI: 10.1007/978-1-4842-2845-6_6. [Online]. Available: https://doi.org/10.1007/978-1-4842-2845-6_6.

[49] A. F. Agarap, *Deep learning using rectified linear units (relu)*, 2019. arXiv: 1803.08375 [cs.NE].

[50] S. Seb, *An introduction to neural network loss functions*, 2021. [Online]. Available: https://programmathically.com/an-introduction-to-neural-network-loss-functions/.

[51] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong, "Analysis and noise modeling of the intel realsense d435 for mobile robots," in *2019 16th International Conference on Ubiquitous Robots (UR)*, 2019, pp. 707–711. DOI: 10.1109/URAI.2019.8768489.

[52] I. Media, *Intel realsense product family d400 series*, Jun. 2020. [Online]. Available: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf.

[53] M. Raspberry Pi, *Raspberry pi 4 model b specifications*, 2018. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[54] I. Corporation, Oct. 2019. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/neural-compute-sticks/NCS2_Product-Brief-English.pdf.

[55] G. Jocher, *Yolov5 in pytorch; onnx; coreml; tflite*, 2020. [Online]. Available: https://github.com/ultralytics/yolov5.

[56] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767v1.

[58] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, Apr. 2020. [Online]. Available: https://arxiv.org/abs/2004.10934?sid=ddnA0P.

[59] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "Cspnet: A new backbone that can enhance learning capability of CNN," *CoRR*, vol. abs/1911.11929, 2019. arXiv: 1911.11929. [Online]. Available: http://arxiv.org/abs/1911.11929.

[60] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu, "A forest fire detection system based on ensemble learning," *Forests*, vol. 12, p. 217, Feb. 2021. DOI: 10.3390/f12020217.

[61] K. Wang, J. H. Liew, Y. Zou, D. Zhou, and J. Feng, "Panet: Few-shot image semantic segmentation with prototype alignment," *CoRR*, vol. abs/1908.06391, 2019. arXiv: 1908.06391. [Online]. Available: http://arxiv.org/abs/1908.06391.

[62] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[63] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, vol. 6, pp. 61 677–61 685, 2018. DOI: 10.1109/ACCESS.2018.2874767.

[64] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, TOsmanov, D. Kruchinin, A. Zankevich, DmitriySidnev, M. Markelov, Johannes222, M. Chenuet, a-andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming, and T. Truong, *Opencv/cvat: V1.1.0*, version v1.1.0, Aug. 2020. DOI: 10.5281/zenodo.4009388. [Online]. Available: https://doi.org/10.5281/zenodo.4009388.

[65] S. Shahinfar, P. D. Meek, and G. Falzon, "How many images do I need? understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring," *CoRR*, vol. abs/2010.08186, 2020. arXiv: 2010.08186. [Online]. Available: https://arxiv.org/abs/2010.08186.

[66] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, Jul. 2019. DOI: 10.1186/s40537-019-0197-0.

[67] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5537–5544. DOI: 10.1109/ICRA.2015.7139973.

[68] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, A. V, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, and M. T. Minh, *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*, version v6.1, Feb. 2022. DOI: 10.5281/zenodo.6222936. [Online]. Available: https://doi.org/10.5281/zenodo.6222936.

[69] T. Tensorflow, *Tensorflow mobilenetssdv2*, 2019. [Online]. Available: https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2.

[70] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[71] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," *CoRR*, vol. abs/2007.02500, 2020. arXiv: 2007.02500. [Online]. Available: https://arxiv.org/abs/2007.02500.

[72] B. Rinner and U. Weiss, "Online monitoring of hybrid systems using imprecise models," *IFAC Proceedings Volumes*, vol. 36, pp. 759–764, Jun. 2003. DOI: 10.1016/S1474-6670(17)36584-9.

[73] J. Stoustrup, H. Niemann, and A. la Cour-Harbo, "Optimal threshold functions for fault detection and isolation," vol. 2, Feb. 2003, pp. 1782–1787, ISBN: 0-7803-7896-2. DOI: 10.1109/ACC.2003.1239853.

[74] P. Goel, G. Dedeoglu, S. Roumeliotis, and G. Sukhatme, "Fault detection and identification in a mobile robot using multiple model estimation and neural network," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 3, 2000, 2302–2309 vol.3. DOI: 10.1109/ROBOT.2000.846370.

[75] K. Zhou and L. Liu, "Unknown fault diagnosis for nonlinear hybrid systems using strong state tracking particle filter," in *2010 International Conference on Intelligent System Design and Engineering Application*, vol. 2, 2010, pp. 850–853. DOI: 10.1109/ISDEA.2010.428.

[76] Schneider and Frank, "Fuzzy logic based threshold adaption for fault detection in robots," in *1994 Proceedings of IEEE International Conference on Control and Applications*, 1994, 1127–1132 vol.2. DOI: 10.1109/CCA.1994.381357.

[77] E. Khalastchi, M. Kalech, G. Kaminka, and R. Lin, "Online data-driven anomaly detection in autonomous robots," *Knowledge and Information Systems*, vol. 43, Jun. 2014. DOI: 10.1007/s10115-014-0754-y.

[78] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[79] C.-F. Wang, *The vanishing gradient problem*, Jan. 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[80] M. Phi, *Illustrated guide to lstm's and gru's: A step by step explanation*, Jun. 2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[81] W. Badr, *Auto-encoder: What is it? and what is it used for? (part 1)*, Jul. 2019. [Online]. Available: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

[82] H. Bandyopadhyay, *An introduction to autoencoders: Everything you need to know*, Mar. 2022. [Online]. Available: https://www.v7labs.com/blog/autoencoders-guide.

[83] T. Luo and S. G. Nagarajan, "Distributed anomaly detection using autoencoder neural networks in wsn for iot," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6. DOI: 10.1109/ICC.2018.8422402.

[84] "Time series," in *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 536–539, ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_401. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_401.

[85] M. InfluxData, *What is time series data?* Mar. 2022. [Online]. Available: https://www.influxdata.com/what-is-time-series-data/.

[86] B. Fortuner, *Activation functions*, Jan. 2022. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html.

[87] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes, "A study of deep convolutional auto-encoders for anomaly detection in videos," *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018, Machine Learning and Applications in Artificial Intelligence, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2017.07.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865517302489.

[88] P. PX4 Media, *Open source autopilot for drones*, Sep. 2021. [Online]. Available: https://px4.io/.

[89] D. T. ArduPilot, *Ardupilot*, 2021. [Online]. Available: https://ardupilot.org/.

[90] P. PX4 Media, *Pixhawk 4 mini technical specifications*, Sep. 2021. [Online]. Available: https://docs.px4.io/master/en/flight_controller/pixhawk4_mini.html.

[91] M. Linux Foundation; DroneCode Project, *Qgc - qgroundcontrol - drone control*, 2019. [Online]. Available: http://qgroundcontrol.com/.

[92] S. Musa, "Techniques for quadcopter modeling and design: A review," *Journal of unmanned system Technology*, vol. 5, no. 3, pp. 66–75, 2018. [Online]. Available: http://www.ojs.unsysdigital.com/index.php/just/article/view/10.21535%5C%252Fjust.v5i3.981.

[93] A. PX4, *Px4 autopilot flight log review*, 2022. [Online]. Available: https://review.px4.io/browse.

[94] A. PX4, *Ros with gazebo simulation px4*, 2020. [Online]. Available: https://docs.px4.io/master/en/simulation/ros_interface.html.

[95] D. T. PX4 Autopilot, *Px4 quadrotor public flight log repository pilot error crash*, 2020. [Online]. Available: https://review.px4.io/plot_app?log=00a25d12-c11a-46d2-a704-e2abe3fd38c6.

[96] M. PX4 Autopilot Dev Team, *Px4 autopilot for drones*, 2022. [Online]. Available: https://github.com/PX4.

[97] M. L. Hachemi, A. Ghomari, Y. Hadjadj-Aoul, and G. Rubino, "Mobile traffic forecasting using a combined FFT/LSTM strategy in SDN networks," in *HPSR 2021 - 22nd IEEE International Conference on High Performance Switching and Routing*, Paris, France: IEEE, Jun. 2021, pp. 1–6. [Online]. Available: https://hal.inria.fr/hal-03510094.

[98] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[99] J. Heaton, *Introduction to Neural Networks with Java*, 2nd. Heaton Research, Inc., 2008, ISBN: 9781604390087. [Online]. Available: https://books.google.com/books?id=Swlcw7M4uD8C.