ASHESI UNIVERSITY COLLEGE

STANDALONE IMU POSITIONING DETERMINING SYSTEM FOR UAVs USING ARTIFICIAL INTELLIGENCE

CAPSTONE PROJECT

B.Sc. Electrical/Electronic Engineering

William Kwesi Akuffo

2021

ASHESI UNIVERSITY COLLEGE

STANDALONE POSITIONING DETERMINING SYSTEM FOR UAVs USING ARTIFICIAL INTELLIGENCE

CAPSTONE PROJECT

Capstone Project submitted to the Department of Engineering, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Electrical/Electronic Engineering.

William Akuffo

2021

DECLARATION

I hereby declare that this capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: 

……

Candidate's Name: Williams Akuffo

……

Date: 11/04/2021

I hereby declare that preparation and presentation of this capstone were supervised in accordance with the guidelines on supervision of capstone laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date: ……

Acknowledgements

To God who first made all things that all other things are possible as a result. And then to my supervisor, Dr Nathan whose encouragement and academic advice helped me undertake this project.

Abstract

Inertial Measurement Units (IMUs) are Micro-Electromechanical Systems (MEMS) that are able to provide acceleration angular orientation rates information via inertial sensing. Unlike other positioning devices like the Global positioning System (GPS), they do not require any form of communication with an external device or technology in order to obtain this information. This makes them the ideal positioning devices to serve as standalone systems. However, with certain drawbacks associated with the IMU they are unable to effectively serve in this role. Existing schemes employ the use of Kalman filters as a complementary approach to solve this issue but this also presents complexity and drawbacks resulting in the failure of the Kalman estimator especially when there is no GPS signal available. This paper proposes a technique by employing the use of an Artificial neural Network (ANN) to model certain state variables in order to estimate the position of an Unmanned Aerial Vehicle (UAV) quadrotor with the IMU serivng as a standalone positionng determing device.

Table of Contents

DECLARATION i
Acknowledgements ii
Abstract iii
Chapter 1: Introduction 1
1. Background 1
1.2 Problem Summary and Objective 2
Chapter 2: Related Literature Review 4
2.1 Inertial Navigation Systems (INS) and Inertial Measurement Units (IMUs) 4
2.2 GPS/INS 4
2.2.1 Coupling and Estimation with the Kalman Filter 6
2.2.2 Estimation with Intelligent methods 7
2.3. Machine Learning (ML) and Artificial Neural Networks (ANNs) 8
2.4 Artificial Neural Networks in Embedded Systems 10
2.4.1 Artificial Neural Networks High Level Overview 10
2.5. Unmanned Aerial Vehicles (UAVs) / Quadcopters 12
Chapter 3: Design Specifications and Requirements 14
3.1 System Requirements 14
3.2 Data Collection Environment Requirements 15
3.3 System Specifications 15
3.4 Environment specifications 16
3.4 High Level System Design Options 16
3.5 Design Selection 18
3.6 Selected Design 19
Chapter 4: Implementation, Testing and Results 21
4.1 Device Build 21
4.1.1 Sensor module 22
4.1.1.1 Data Sensing and Feature Production 22
4.1.1.2 Feature Engineering Architecture 23
4.1.2 Communication module 26
4.1.3 Intelligent module 26
4.1.4 Power Module 27
4.1.5 Device Casing 29
4.1.6 Device Assembly 30

4.2 Data Collection 31
4.3 Model Training 33
4.3.1 Data download 33
4.3.2 Reading data 34
4.3.3 Data cleaning 34
4.3.4 Exploratory Data Analysis 34
4.4.4 Data pre-processing – Feature Normalization and Scaling 38
4.4.5 Train test split 39
4.4.6 Model build 40
4.4.7 Model fitting 41
4.4.8 Model Validation 43
4.5 Deployment and Real-World Testing and Model Iteration 46
Chapter 5: Discussion and Conclusion 49
5.1 Limitations 49
5.2 Future Works 50
References 51
APPENDICES 53

Chapter 1: Introduction

1. Background

Our lives have become increasingly reliant on positioning determining systems. We depend on them to tell us where we are in real time, or where an object is within a certain reference frame. Some modern vehicles, such as ships and boats, automobiles, planes and helicopters, and spacecrafts, are equipped with positioning determining systems that enable them to monitor their positions more easily and conveniently within the space (reference frame) in which they are operating. Inertial Measurement Units (IMUs) and the Global Positioning System (GPS) are two examples of positioning systems (GPS). Inertial Measurement Units can be electronic devices that provide acceleration and orientation information. This data can be used to estimate the position or orientation of the vehicle or an object in space by applying some special algorithms to them. Autonomous vehicles (land or aerial) require that their position in a local or global reference frame is known in real-time. This helps it navigate properly within their reference frames. These positions in real-time need to be frequently updated to match their true position in the reference frame.

Most Unmanned Aerial Vehicles (UAVs) are equipped with GPS for position reporting. Others like beginner drones may not be. However, what is common to both is the IMU. Data from the IMU is required by the UAVs control algorithms to achieve the desired flight maneuvers. The GPS onboard may not always be able to report the UAVs global position especially in situation where there is the absence of a signal (GPS outage), no-clear-line-of-sight, or in urban areas or around somewhat closed spaced such as tunnel. The GPS may suffer signal blockages or multipath errors. Inherently, the GPS suffers all drawbacks associated with signal propagation. Like most

signal propagation dependent position estimation methods, the GPS always requires some form of external technology with it can communicate before position can be estimated. The IMU on the other uses inertial sensing which allows it so provide information such as angular orientation rates and acceleration from its gyroscopes and accelerometers, respectively. The IMU however has certain drawbacks which prevents it from being used as a standalone position estimating device. Signal from the accelerometer can be noisy whereas the gyroscope may be subject to drift. Current techniques employed to solve these problems utilize Kalman filters to fuse the IMU with the GPS. This can be quite complex in development and presents its own drawbacks. The Kalman filters fail when there is GPS outage because the GPS is required to compensate the IMU error, Other schemes propose the use of Artificial Neural Networks (ANNs) to complement the Kalman filter when this happens. All in all, ANNs have been proven to have a robust ability to model complex non-linear functions and relationship. Other schemes like [4] employ ANNs as sole estimators to reduce the complexity the Kalman Filter presents while being able to stand fast in all circumstances of uncertainty.

1.2 Problem Summary and Objective

Using IMUs for position estimation is still an interesting area of research. However, their inability to provide sub-meter level accuracy over prolonged periods makes this incredibly difficult. IMUs as standalone position estimators quickly drift, producing unbounded estimates which are unreliable. Therefore, they are unable to exclusively serve as standalone positioning systems. Existing schemes like [1] fuse IMUs with GPS with Kalman Filtering approaches. However, this ramps up the approach complexity and cost by requiring additional hardware (GPS). Such methods are dependent on the availability of a GPS locking signal and fails in its absence.

More intelligent methods combine IMUs with GPS using some fuzzy logic or neural networks. In an attempt to reduce complexity, [4], [14], [15] propose techniques that prove neural network as an estimator due to its robust ability to learn very complex non-linear. Current methods are complex, not reliable, and not self-reliant. For example, [1], utilizes a Kalman Filter in the presence of a GPS signal and then switch to an ANN when there is GPS outage. However, the accuracy of the ANN only lasts for about 60 seconds after which it begins to deteriorate This makes the system somewhat dependent on the availability of a GPS signal. The system therefore is not effectively self-reliant without the GPS signal and complex with the utilization of the Kalman Filter. Therefore, taking advantage of the ability of ANNs to model complex relationships (linear and non-linear) between variables, an ANN will be trained on data collected from an IMU and sonar sensors (forming the state variables of the UAV) as a form of black box modelling approach, learning the relationships between these variables in order to estimate the position of a UAV. This technique will be also used to investigate the feasibility of an IMU as a standalone positioning determining system. A portable devices capable of being carried by the UAV will interface with two distinct sensors which are an IMU and a sonar sensor. Data will be collected from these two sensors, on which an ANN will be trained in order to model the relationship between the IMU data and the sonar data. Therefore, using the IMU alone, the position) of the UAV (which was initially given by the sonar data)could be estimated by the trained ANN making the IMU a standalone (self- reliant) positioning determining device.

Chapter 2: Related Literature Review

In this chapter, related literature will be reviewed. Existing techniques or methods and technologies related to the topic will also be discussed in detail.

2.1 Inertial Navigation Systems (INS) and Inertial Measurement Units (IMUs)

An Inertial Navigation System is made up of IMU sensors. These could be gyroscopes, accelerometers, and magnetometers. MEMS IMU sensors are Micro-Electromechanical Systems that can sense inertial information such as inertial acceleration and inertial rotation. INS (Inertial Navigation System) systems (comprising of IMU) may utilize a computer to calculate position using a method known as dead reckoning without the need for an external reference frame [4]. However, using this method with and IMU or INS has its drawbacks. The INS may only be accurate within a very short time. Over a long period, it accumulates errors which degrades the estimation [4]. INS may be coupled with GPS in some applications for positioning determination and navigation [1], [4].

An intuitive approach for position estimation using the IMU data is to perform a double time integration on the IMU acceleration (from the accelerometers) to produce an estimate of the vehicle's position and a single time integral (from the gyroscopes) to estimate the vehicle's orientation. However, this can be problematic because, the noise in the IMU's signal produces an error with a non-zero mean which gets compounded with the true estimate within a short period [1], [2]. This can lead to very large deviations from the true position of the vehicle or object.

2.2 GPS/INS

Combining the GPS with the INS provides a robust way of dealing with the problems of one sensor alone. For example, the INS is subject to accumulate errors within a fair amount of time

which causes a degradation in the position estimates it produces. However, the GPS is immune to this kind of drawback. The GPS on the other hand suffers all drawbacks associated with signal propagation whereas the INS Therefore, coupling or fusing these 2 sensors together allows for error compensation from one of the 2 sensors. The GPS is usually used to compensate the INS error [1] during position estimation. Depending on the application and functional requirements, various schemes propose various approaches for GPS/INS fusion or coupling.

. The working of GPS combines a receiver and a transmitter. The receiver must communicate with a minimum of 4 GPS satellites for effective trilateration [3]. Trilateration is the method used to determine the position of a GPS receiver on the surface of the earth [3]. GPS signals can suffer signal blockages and multipath in urban areas with building presenting themselves as obstacles, obstructing the clear line of sight needed by the GPS for position estimation. Inherently [4], GPS suffers all the drawbacks associated with signal propagation such as multipath and other forms of distortion.

GPS may be coupled with an INS (Inertial Navigation system) which is comprised of an IMU to improve the position estimation. Numerous approaches utilize the Kalman Filter as a real-time fusion algorithm of the GPS sensor data and the INS sensor data [4], [6]. The goal of the fusion is to allow one sensor to compensate the other where one might fail. In many cases, this is more robust than having one sensor as a standalone position estimator. The Kalman Filter acts as an error estimator for the INS as long as the GPS signal is available [1]. Typical Kalman Filters tend to fail along with the GPS/INS system when there is GPS outage [1]. This is because the INS without the GPS deteriorates rapidly within a short time. Without GPS signal being available, the Kalman Filter becomes useless making the whole system being dependent on the availability of a GPS signal.

When coupling GPS with INS in the case of UAVs, the extra sensor (GPS), means extra cost and extra weight. This extra weight also means more energy or power to work the GPS and fly the UAV. Hence there is a reduction in flight time [5]. This incurs extra cost on the side of the GPS hardware. The integration of these two systems as one unit is usually done using a Kalman Filter as the sensor fusion algorithm. Kalman Filters are simply state estimators that predict the system's state by making past observations in conjunction with current noisy observations. The Kalman Filters are then used to estimate the INS error for compensation [4]. However, this only works if GPS signal is available. Since the GPS is required by the fusion algorithm to estimate to INS error for compensation, the entire systems become somewhat dependent on the availability of a GPS signal which is not very helpful in the event of a GPS outage, The Kalman Filter therefore fails to do its INS error estimating job. In multipath prone environments as well, such as urban areas and semi enclosed areas, the performance of the GPS is also affected leading to the gradual deterioration of the GPS/INS system.

2.2.1 Coupling and Estimation with the Kalman Filter

Coupling the GPS with INS can take on several forms. Some of these are loosely or tightly coupled integration, closed or open loop integration. In all such approaches, a typical Kalman Filter is required [7]. As mentioned earlier, these Kalman Filters are used as state estimators. Modified versions of these Kalman Filters are the Extended Kalman Filter (EKF), which is applicable to non-linear systems, but has a degradation and divergence problem due to its linearization process [1]. The Unscented Kalman Filter (UKF), which has been investigated to have good performance than the EKF with about the same complexity [1]. The UKF makes use of an Unscented transform which is used to propagate means and covariance through a non-linear

transformation [1]. In [1],[7], a set of variables are selected to represent the state . Some of these state variables are position errors, velocity errors, accelerometer biases, as well INS and GPS position differences. This is used as a measurement vector input to the Kalman Filter to estimate the INS error for compensation. Therefore, the GPS and INS are fused together by a Kalman Filter which uses both sensors for state estimation and position correction. However, there can be situations (No clear line of sight) that lead to GPS outage or GPS signal multipath. The former can cause the Kalman Filter to fail [1] because, there would be no GPS to compensate the INS. As a result of this, some schemes like [1], propose new ways of using intelligent methods in combination with Kalman Filter to address this.

2.2.2 Estimation with Intelligent methods

There are many proposals that seek to improve GPS accuracy, but these complexities increase with the increasing need for more accuracy [4]. The common way is to combine GPS and INS with some sort of fuzzy logic or neural networks [4]. In [4], a unique technique based solely on Neural Networks is proposed. The sensor fusion algorithm or estimator which would have been the Kalman Filter in most GPS/INS sensor fusion schemes as seen in [1] is replaced entirely by a neural network. Neural Network proves robust where typical GPS/INS may suffer deterioration in urban areas because of signal blockage and multipath, the Neural Network would prove robust. Another technique to use the ANN together with the Kalman Filter. In [1], a Back propagation Neural Network is used to simulate an Unscented Kalman Filter (UKF) during GPS outages for INS error compensation. Other methods may propose other Kalman Filters depending on requirements. However, regardless of the type of Kalman Filter used, if there is no GPS signal available, the GPS/INS systems fails because the INS is left alone to do the estimation without the

GPS for compensation. Usually, the Kalman Filter algorithm also fails here. This is because it is no longer able to do the error estimating job to compensate the INS without the GPS. This makes the reliability of the whole system somewhat dependent on the availability of a GPS signal.

To account for signal multipath distortion and the likely event of a GPS outage around areas with no clear line of sight, ANNs (Artificial Neural Networks) are used in some schemes [1],[4] to either fully or partially replace the GPS/INS estimator (Kalman Filter or fusion algorithm) in the event of GPS outage. When GPS signal is available, an ANN may be trained in real time on the GPS errors, INS errors or the estimator's errors.

Some schemes train the ANN to completely replace the estimator[4]. Therefore, the Artificial Neural Network can perform the job of the Kalman filter or estimator under multipath condition where the GPS might be deteriorating or during GPS outages where the Kalman Filter itself might fail. Many others go on to train an ANN which go on to train the ANN on the INS error are able to use only data from the INS and the ANN to accurately predict navigation information while causing the INS to act as a standalone system. Due to the robust ability of Neural networks to generalize to uncertainties and non-linearities, they are explored and used in a wide variety of modelling applications.

2.3. Machine Learning (ML) and Artificial Neural Networks (ANNs)

Machine learning addresses the question of how to build computers or algorithms that self-improve over experience [8]. Though not human experience, the experience is learned in the form of fine-tuning certain weights that map input-output relationships through lots of data with the assumption that there is a theoretical relationship between the input and output variables. The situation whereby an algorithm gets better at a particular task overtime by not explicitly being

programmed. Machine learning has been widely adopted in various industries such as engineering, research, even up to customer service, to help solve extremely complex problems which cannot be easily modelled by humans. For machines, these problems become very easy once they have learned the right weights or parameters through experience or training. Generally, there are about three main approaches used in machine learning. These are, supervised learning, unsupervised learning, and Reinforcement learning. Under supervised learning, the algorithm is given labeled data from which it learns or trains on. Examples of supervised learning problems are regression and classification. Unsupervised learning on the other hand does not require labeled data. The algorithm is designed to learn these labels. Typically used where the engineers themselves have no clue what the labels are. An Example of an unsupervised learning problem is clustering. Reinforcement learning however requires no data at all. The algorithm presents itself as an agent in an environment from which it learns the best policy mapping actions to states within the environment usually driven by a reward scheme. Good actions merit good rewards whereas bad action results in bad rewards or punishments. Reinforcement learning can be used in self-driving cars or robots.

Artificial Neural Networks, a special kind of machine learning algorithm employs several techniques to learn weights that map inputs and output relationships. These relationships are not explicitly known. The first technique is a feed-forward algorithm that passes the data through the network to make predictions. After this phase, a back propagation algorithm is used to update the weights within the network. This would be the first epoch of many possible iterations. This cycle repeated several times until the optimum sets of weights is found or learned by the network, hence an optimum solution to the problem is found. This is called convergence, Most ANNs may employ ‘stochastic gradient descent’ as the learning algorithm whereas others may use the ‘Adam’ learning

algorithm. A cost or error function usually 'mean-squared-error' or 'root-mean-squared-error' is used to monitor the loss of the network. All these helps it to learn and converge. ANNs have proven themselves to be very robust for complex linear and non-linear modelling while being able to generalize well [4], [9]. Neural Networks with just a single hidden layer can be used to approximate very complex non-linear functions or relationships[4], [10].

2.4 Artificial Neural Networks in Embedded Systems

The execution of most deep learning tasks is highly restricted to platforms with high compute power making use of GPUs or external hardware accelerators [13]. Most embedded microprocessors and controllers simply do not have the compute power to perform these computationally strenuous tasks. Even if they are able to run inferences on data using deep learning models, they end being too slow. Such a performance is usually unacceptable in most machine learning or deep learning applications. In order to improve performance in embedded applications, precision scaling may be used to reduce memory occupation of such models or networks. Weights are rounded using fixed-point representations reducing them from higher bit data types to 8-bit data types. This reduces computational load and improves performance [13]. However, there could be some trade-offs in accuracy.

2.4.1 Artificial Neural Networks High Level Overview

Artificial Neural Networks are essentially algorithms that are able to extract patterns from data and learn those patterns through weights. A set of input parameters are weighted to assign levels of importance to the individual input parameters or features. These features after being weighted (multiplied with the weights), are linearly combined in a summation. A bias term may

be added. The result is then transformed by a non-linear activation function in most cases from which an output estimation may be obtained. This is visualized in **Figure 1** below.

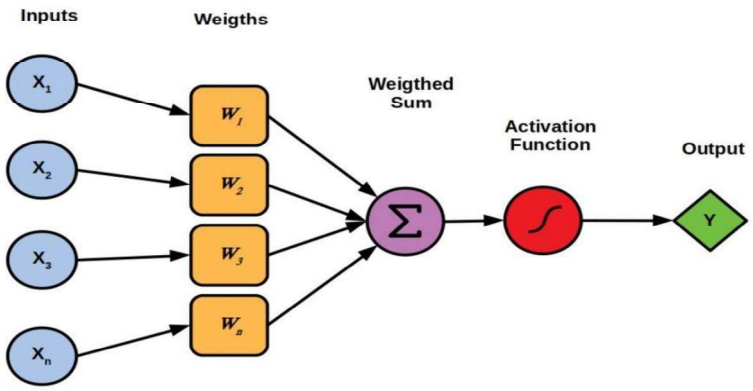


Figure 1: Artificial Neural Networks at the core.

Artificial Neural Networks primarily involve 2 processes.

- *Feed Forward:* This is the process of computation whereby predictions or estimations are done using learned or initialized weights.

Output — Linear combination of inputs

$$y = g(\sum_{i=0}^{n} Xi\, Wi)$$

Non-linear activation

- *Back Propagation:* This is the process of computations where results in updating former weights with newly computed ones. The new weight is calculated as:

$$\boldsymbol{Wnew = W - lr(\frac{\partial J(W)}{\partial W})}, \textit{ where}$$

$\frac{\partial J(W)}{\partial W} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial W}$, where z is the output of an intermediate neuron,

$\boldsymbol{Wnew}$ is the newly computed weight,

$\boldsymbol{W}$ is the former weight at the previous iteration,

$\boldsymbol{lr}$ is the learning rate of the algorithm and $\boldsymbol{J}$ is the loss function.

2.5. Unmanned Aerial Vehicles (UAVs) / Quadcopters

Unmanned Aerial Vehicles (UAVs) popularly referred to as drones are aircrafts that operate without onboard human pilots. UAVs are a part of an Unmanned Aerial System (UAS) which usually includes a ground-based controller and a system of communication between them [11]. Advanced UAVs may have an auto-pilot feature that allows it to automatically return to the ground station in the event of a signal loss or critical battery power [12]. UAVs may be designed to have a number of propellers usually with equiangular spacing between them. The number of propellers presents different flight and control dynamics to the UAV. Quadrotors or quadcopters are designed to have 4 propellers. Thrust on each propeller is varied in order to achieve a desired control or flight maneuver. In modern time, UAVs are applied in so many industries such as aerial photography, deliveries, the military, and RC-sports. UAVs may also be used for surveillance missions and surveying land areas.

The most common technology used to determine the position of UAVs is the GPS. Onboard GPS receivers communicate with GPS satellites which constantly track and report the position of the UAV in real-time using the propagation time of the signal (time of flight). GPS as a positioning system for UAVs can provide very accurate positioning information as long as there is a clear and unobstructed line of sight [1], [4]. However, in places where there is not, the GPS may suffer signal blockage or multipath signal distortion which can cause failure or errors in the GPS positioning [1], [4]. Further, GPS consumes more energy than IMUs and have a low position update rate making it unsuitable for real-time demand of UAVs [5]. This can affect the

flight time of such UAVs. In high dynamic maneuvering environments, higher update rates of the UAVs position may be required to provide a smooth position report in real-time. The drawbacks of GPS such as the increased probability of signal blockages around buildings, and multipath distortion, which yields multiple copies or reflections of the GPS signal, prevents it from being an effective positioning system indoors or somewhat enclosed spaces, or around buildings and in urban areas, where these conditions are present. Accuracy and precision of position may be greatly compromised.

Chapter 3: Design Specifications and Requirements

In this chapter, the fundamental design requirements, and specifications of a lightweight system capable of being carried by a UAV will be presented. This device will have two distinct sensors as shown in **Figure 2**. The MPU6050 which is an Inertial Measurement unit consisting of a 3-axis accelerometer and a 3-axis gyroscope. The device will be carried by a UAV within a chosen flight environment to collect data from these sensors from which an Artificial Neural Network will be modelled. The selected UAV type is a quadrotor. A quadrotor is selected for this application because;

- It is easily maneuverable.
- It can fly within a small and confined space.
- Its cost and availability are within project constraints - time, cost, and scope.

HC-SR04 sonar sensor

MPU6050 – IMU

Figure 2 : The two distinct sensors

3.1 System Requirements

The device should:

- be inexpensive.
- easily interface with the UAV.
- not be heavy for the UAV.
- have a reasonable power consumption.

- not be bulky UAV.
- Should be computationally efficient.
- Should be able to handle and collect data and store easily.

3.2 Data Collection Environment Requirements

- *Nature of environment:* The kind of environment required to collect the required data for this application has to have walls forming a 3d corner with representable 3d planes (x, y and z). A 3d corner looks like a 3-axis Cartesian plane with the meeting of 3 planes at 90-degree angles to each other. This can be represented as 2 walls meeting in a corner at 90-degrees and standing vertical at 90 degrees to the ground.
- *Environment space:* Space should be of good size to be in range of proximity sensors.

3.3 System Specifications

Criteria	Requirement	Specification
Cost	Should be inexpensive	Cost < GHC 500
Compute	• Should have reasonable inference time. • Inference time should not affect performance	Inference time < 1 second
Weight	• Should be light weight. • Weight should not compromise flight time of UAV	Weight < 100g
Power Consumption	• Power consumption should be reasonable. • Power consumption should not affect flight time of UAV	Power < 10W
Size	Should be portable	Size <150mm by 50mm by 30mm

Data Handling	• Should have good storage size for temporal data storage and handling (e.g., push to cloud a base) to be retrieved later for training.	• memory > 1Gb • Push to GitHub

3.4 Environment specifications

Environment for Data Collection	• Data collection and training environments should be in a 3d corner with representable 3d planes (x, y and z) • Space should be of good size to be in range of proximity sensors.	Size <= 400cm (l) by 400cm (w) by400cm (h) by

3.4 High Level System Design Options

Option A : As visualized in **Figure 3** below, a Micro-Controller Unit (MCU) interfaces with the sensors. The (MPU6050 IMU) 3-axis gyroscope and accelerometer module and 3 sonar sensors (HC-SR04). It also hosts the Artificial Neural Network estimator for predicting the UAV's 3-axis position. The sensors will collect data used for training the ANN and data for making real-time predictions after the ANN is trained. most MCUs do not have enough compute power and memory for Deep Learning and Machine Learning applications in general therefore inference times from hosted models might be too slow and not meet the application requirements. For this application, real time estimations need to be done by the system. However, the lack of compute power for most microcontrollers may cause this design specification to not be met.

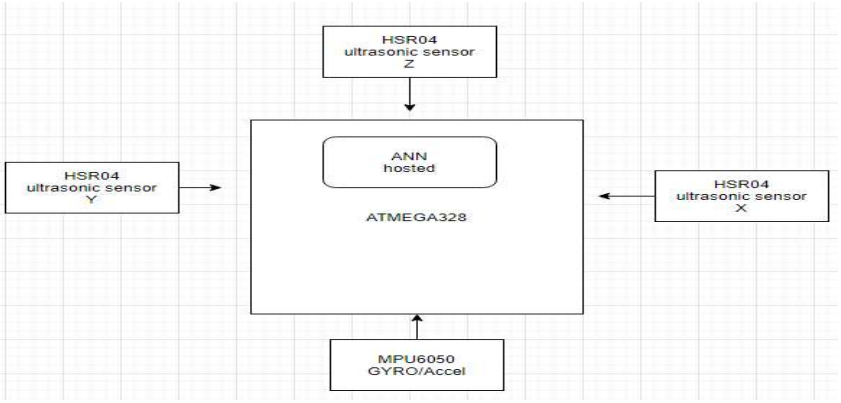


Figure 3: High level design architecture of design option A.

Option B : With this option, there are 4 submodules.

- *The Intelligent module*: This is made up of the ANN hosted on a raspberry pi. The raspberry pi offers about 1.4GHz of processing power compared to the 16MHz clock of the ATMEGA328P which is the choice of MCU used. This would provide enough computational resources for the ANN to meet the inference time of the design specification which is supposed to be less than 1 second for every inference by the model.
- *Sensor module*: The sensor module is made up of all the systems sensors and the micro-controller unit. Data sensing and feature production (engineering) of the UAV's state variables is done here.
- *Communication module*: This is the link between the Intelligent module and the Sensor module.
- *Power Module*: The power module will consist of a voltage regulator which will supply the right amount of power to the device. The main power source will be the UAV's battery.

As visualized in **Figure 4,** this option has the with sensors (MPU6050 3-axis accelerometers and gyroscope, 3 HSR04 Ultrasonic sensors) connected to the ATMEGA328 MCU. The data is collected by the MCU and transferred to the raspberry pi which rather hosts and runs ANN.

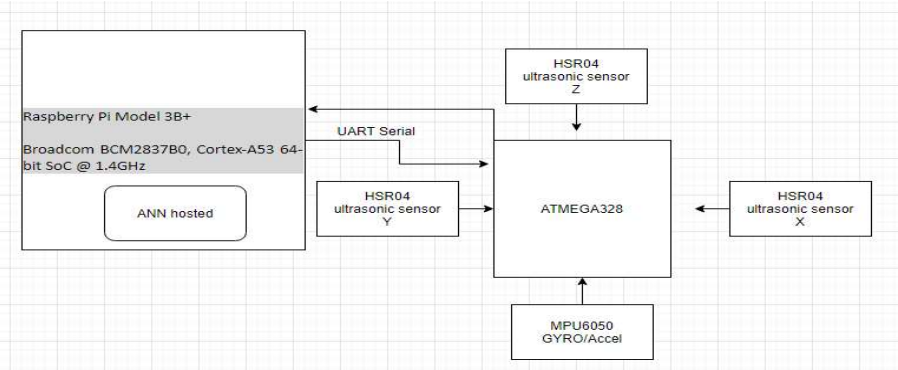


Figure 4: High level architecture of design option B.

3.5 Design Selection

Using the Pugh matrix, the design selection criteria are presented and evaluated on the proposed design options. The highest scoring design is selected and implemented. This can be seen in the Pugh matric below of **Figure 5**. Alternative 2 (option B) was selected.

Criteria	**Weight**	**Alternative1**	**Alternative 2**	**Baseline**
Weight	2	+2	+1	0
Power Consumption	2	+3	-1	0
Computational Power/Efficiency	3	-3	+1	0
Data Handling	2	+1	+2	
Size	1	+2	0	0
Cost	1	+2	+1	0
Score		**0**	**8**	

Figure 5: Pugh Matrix for design selection

3.6 Selected Design

The selected design presents two architectures. The first is during data collection and training and the second is during the real-time position estimation of the UAV. This can be seen in **Figure 6**. During data collection and training, the data is collected by the sensors which interface with the MCU and transferred to the raspberry pi for management and storage. When that is done, the data is pushed to the project's GitHub repository and then pulled unto a local machine where the model is trained. During real-time position estimation, only, data from the IMU (MPU6050) is required from the sensor module. When the data is transferred to the raspberry pi to be used for inference, the results of the predicted UAV positions are displayed over a Local Area Network IP address which the raspberry pi is connected to during an SSH connection.

A. Estimating positions B. Data collection and training

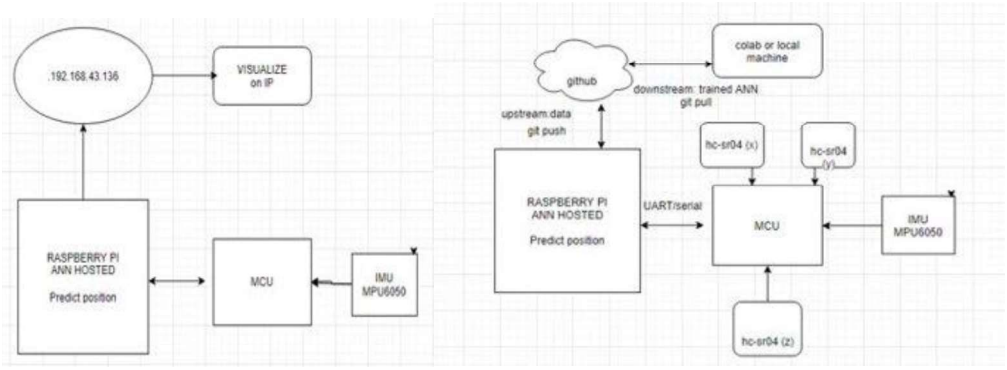


Figure 6: Selected design architecture

A complete circuit schematic of the system consisting of all submodules is shown in **Figure 7** below along with the designed Printed Circuit Board (PCB). The sensor module has digital pins that connect 3 sonar sensors (HC-SR04) and one IMU (MPU6050) to the microcontroller. These sensors are seen in **Figure 2.** The power module powers the system using power from the UAVs battery. Data collected from the sensors is then transferred to the raspberry

pi (intelligent module) over the UART-Serial interface from the TXD pin (pin 3 on the ATMEGA328 for transmit) on the microcontroller to the RXD (GPIO 15 on the raspberry pi for receive) pin on the raspberry pi.

Schematic PCB

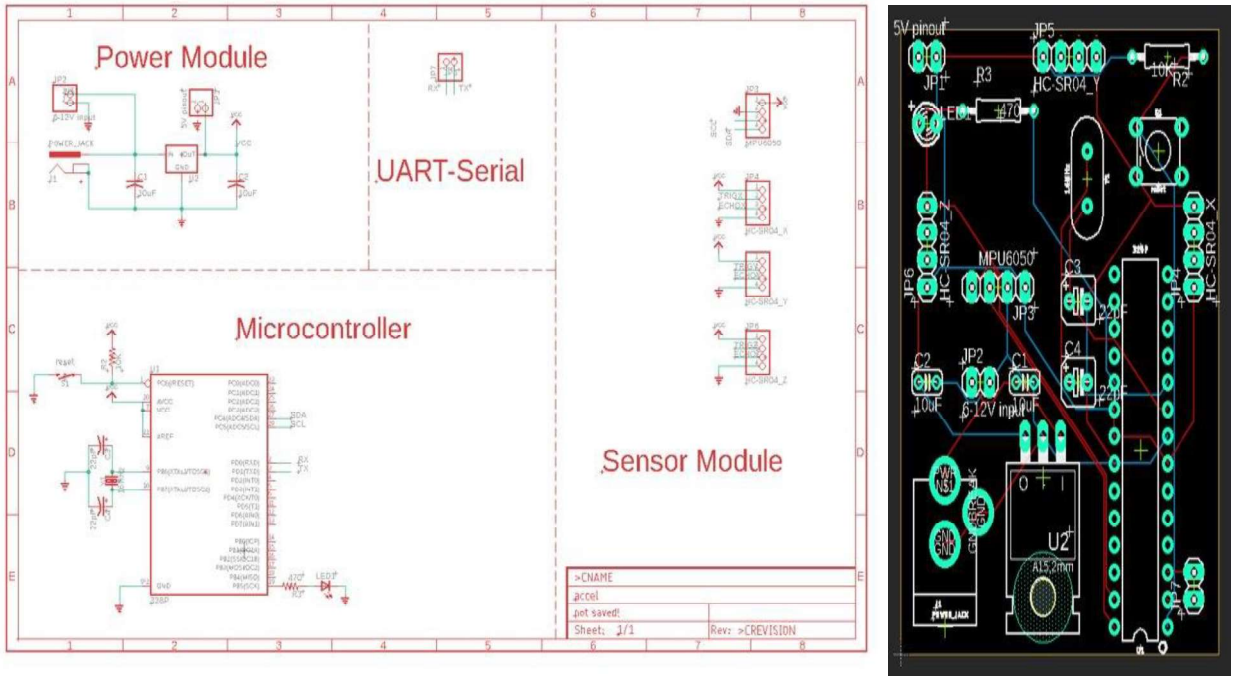


Figure 7: Electronic Design Schematic of submodules and Printed Circuit Board.

Chapter 4: Implementation, Testing and Results

In this chapter, the procedure undertaken to implement the project will be presented and discussed. The project implementation occurred under four primary parts which follow as;

- Device build
- Data collection
- Model training
- Model deployment and real-world testing.

4.1 Device Build

This phase of the project involved building the device itself by sourcing the required electronic components and implementing all the circuitry required. All hardware and electronic components were put together. This was accomplished by developing the device's 5 subsystems shown in **Figure 8.**

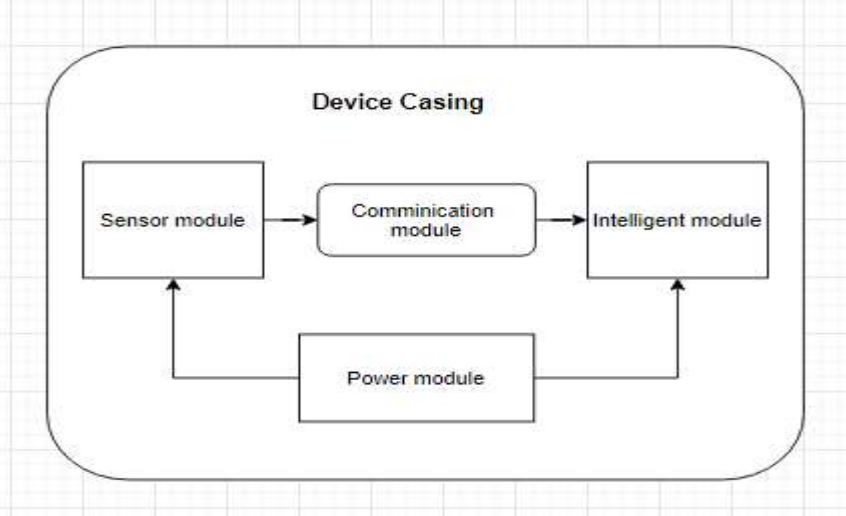


Figure 8: All submodules housed within device casing.

4.1.1 Sensor module

The sensor module consists of 4 sensors. These are three (3) HC-SR04 sonar sensors and one (1) 3-axis MPU6050 IMU. The sonar sensors provided the ground truth 3-axis (x, y, z) positions of the UAV, while the IMU provided inertial data such as 3-axis accelerations the accelerometers and 3-axis angular rotation rates from the gyroscopes. It also consisted of the microcontroller unit (ATMEGA328) which interfaced with these sensors for the required information. Data from the sensor module is collected by the intelligent module for storage and management to later serve as training data for the Artificial Neural Network. After the model has been trained and deployed to be hosted on the intelligent module, this data is used for real time prediction of the UAVs position by the Artificial Neural Network.

4.1.1.1 Data Sensing and Feature Production

Feature production consists of data sensing by the sensors and feature engineering. Feature engineering is done to produce extra features or state variables using the sensor data. Some examples of the sensor data are the 3-axis positions from the ultrasonic sensors, and the 3-axis accelerations from the IMU (Inertial Measurement Unit). All these present themselves as features or parameters to be used by the ANN (Artificial Neural Network) in making predictions. Feature engineering therefore provides more information to the ANN in the form of extra variables or parameters. These features form the state variables of the UAV (Unmanned Aerial Vehicle) in flight. Thus, more features mean more state variables to define the system, thereby supplying more information for the ANN to make better predictions. Two examples of such engineered features include velocity and position integrated and double integrated from the IMU accelerations, respectively. Acceleration here is directly sensed by the IMU. Consequently, the position and velocity are engineered from this measurement by finding their time integrals. The sensors are

grouped into two according to which features or parameters they produce (input or output features or state variables). The IMU (MPU6050) is the input sensor. It measures all data used as input features or parameters to the ANN. The ANN uses this input to predict the output parameters or state variables. Some of these input features or parameters include the 3-axis accelerations (ax, ay, az) and the 3-axis angular rotation rates from the accelerometers and the gyroscopes, respectively. Another is the angular orientation in degrees measured by the IMU. The output parameters are measured in corresponding time with the input features. The time corresponding output parameters or variables are measured by the 3 ultrasonic range sensors (HC-SR04) oriented in the x, y, and z axis to capture the 3-dimensional position of the UAV quadrotor in point space of its environment. The ultrasonic sensors serve as the output sensors, producing those parameters (ground truth) the ANN is trained to predict using the input features. Some of these output features include the 3-axis position, velocity and acceleration of the UAV measured by the 3 ultrasonic sensors.

4.1.1.2 Feature Engineering Architecture

There are 2 kinds of parameters or features. These are grouped according to which sensors they come from. Parameters from the output sensor (ultrasonic HC-SR04) forms the output features. These are the variables the ANN is trained to predict using the input features. Parameters from the input sensor or IMU forms the input parameters or features. These are used to predict the output.

Input – IMU (MPU6050)	**Output – ultrasonic sensor (HC-SR04) - ground truth**
3-axis accelerations (non-linear **with** gravity component) 3-axis accelerometers **(a_r)**	3-axis true positions (x, y z) of the UAV **(r)**
3-axis angular rotation rates 3-axis gyroscopes **(g_r)**	

Figure 9: Features Directly Measured by Sensors

From the features directly measured by the sensors whose symbols are $\mathbf{a_r}$ (raw 3-axis accelerations from the IMU's accelerometers) and $\mathbf{g_r}$ (raw 3-axis gyroscope rates from the IMU's gyroscopes) as in seen in **Figure 9**, other state variables are engineered to provide more information on the state of the UAV in flight. This is illustrated in **Figure 10.**

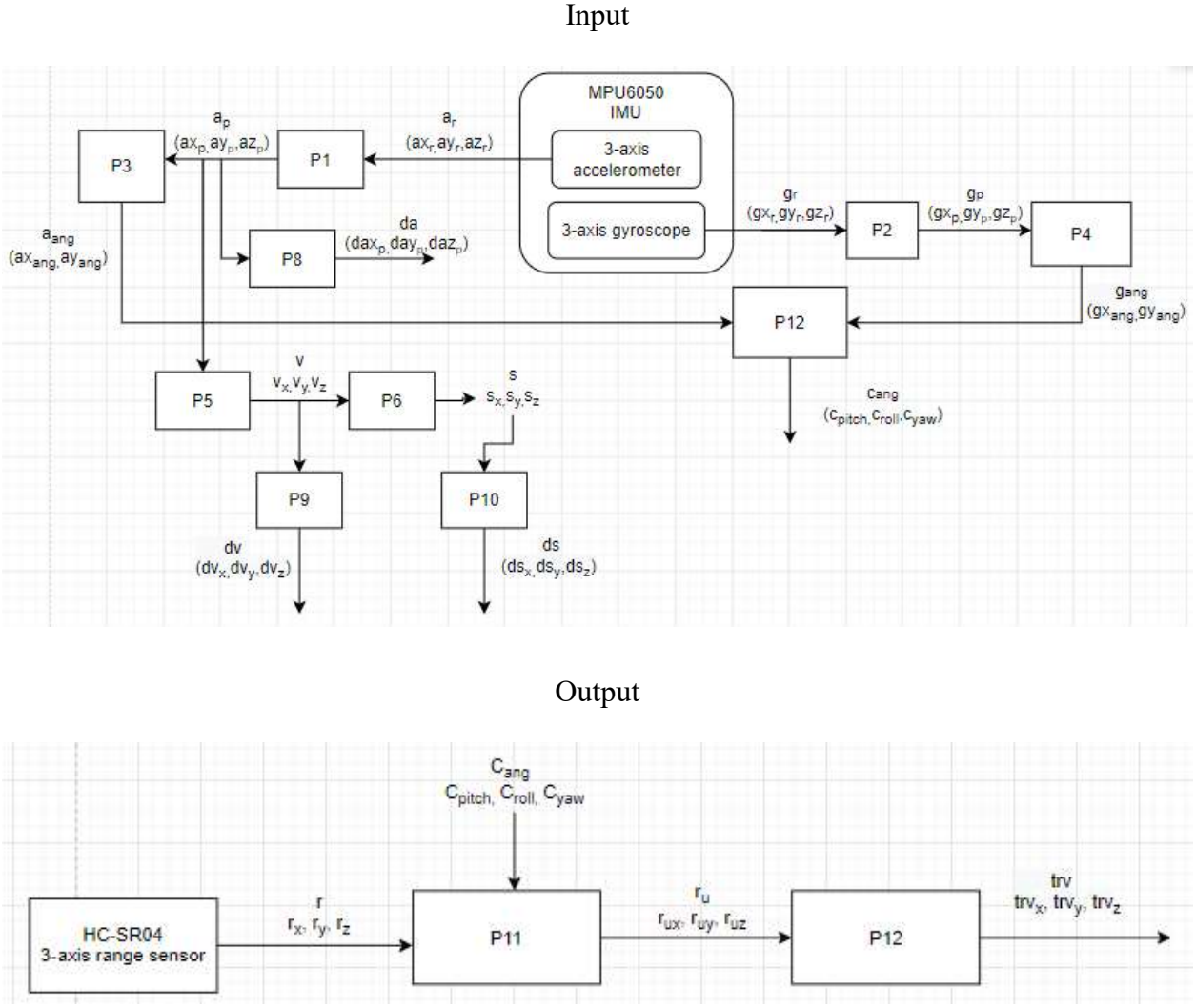


Figure 10: Feature Engineering architecture

P	Transform	Description
1	$ap = \frac{ar}{sensitivity}$	**ap** is 3-axis acceleration (**ax_p, ay_p, az_p**) in meters per second
2	$gp = \frac{gr}{sensitivity}$	3-axis angular rates in degrees per second
3	axang = $\tan^{-1}(\frac{ayp}{\sqrt{axp^2 + azp^2}})$ ayang = $\tan^{-1}(\frac{axp}{\sqrt{ayp^2 + azp^2}})$	Pitch (**axang**) and roll (**ayang**) Euler angles calculated from accelerometers only
4	gxang = $(gxp - offset)$*dt gyang = $(gyp - offset)$*dt gzang = $(gzp - offset)$*dt	Single time integrated angles pitch (**gxang**), roll (**gyang**) and yaw (**gzang**) calculated from gyroscopes only.
5	Cpitch = μ(gxang) + $(1 - \mu)$axang Croll = μ(gyang) + $(1 - \mu)$ayang Cyaw = (gxang)	**Cpitch, Croll and Cyaw** are the complementary filter angles calculated from both the gyroscope and accelerometer angles. $\boldsymbol{\mu}$ *is trust factor between* 0 *and* 1
6	$Vt = (Vt - 1) + ap * dt$, for z, ap = ap-gravity	**Vt** is the single time integrated velocity from the accelerometers
7	$St = (St - 1) + (Vt - 1)dt + 0.5(ap * dt^2)$, for z, ap = ap-gravity	**St** is the double time integrated distance or displacement from the accelerometers
8	da = ap_t - ap_{t-1}	**da** is the magnitude change in **ap**
9	dv = Vt - Vt-1	**dv** is the magnitude change in **Vt**
10	ds = St - St-1	**ds** is the magnitude change in **St**
11	ru = rcos(Cpitch)cos (Croll)cos (Cyaw)	**ru** is the un-angled distance measured from the x, y, z planes. Corrects for UAV pitch, roll and yaw
12	trv = ru - (origin)	**trv** is the distance (**trv_x**, **trv_y**, **trv_z**)travelled in each axis from each plane (x, y and z)

Figure 11: Table of feature transform

The feature transform blocks labeled P1 to P12 whose transforms can be found in **Figure 11** above are used to produce all the feature engineered parameters used to monitor the UAVs state at each time step of feature sampling during data collection or position estimation. The output of these transforms represent the UAV's state variables.

4.1.2 Communication module

Communication is required between the sensor module and the intelligent module. This is seen in **Figure 12.** After data has been sensed from the real world, it is passed from the sensor module to the intelligent module using the Universal Asynchronous Receiver Transmitter (UART) hardware. On the raspberry pi, the UART pins are GPIO 14 (TXD/Transmit) and GPIO 15 (RXD/Receive). The ATMEGA328 has its UART pins on digital pin 2 (RXD/Receive) and digital pin 3 (TXD/Transmit). Using two jumper wires, the TX pin on the raspberry pi is connected to the RX pin on the ATMEGA328 and the RX of the raspberry pi to the TX of the ATMEGA328. Data is sent asynchronously without the need for a clock synchronizing the 2 modules. A timing parameter known as the baud rate is agreed upon by the two modules indicating the number of symbols transmitted per second. The baud rate used was 115200.

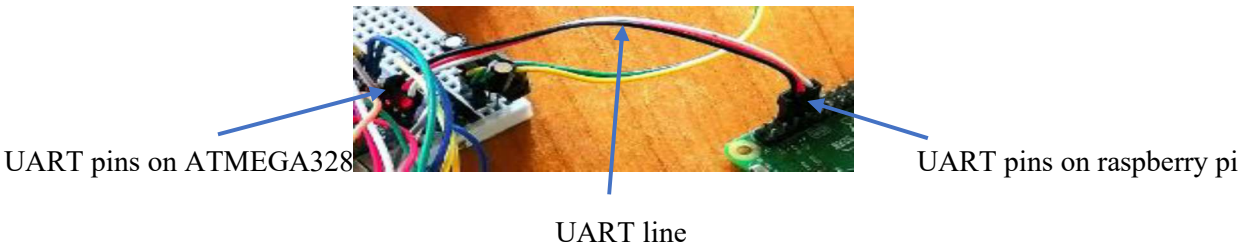


Figure 12: The communication module

4.1.3 Intelligent module

The intelligent module performs 2 roles. These are;

Data collection role: Data sent over from the sensor module is collected and managed by the intelligent module during data collection. Data collection occurs when UAV flight data. Data collection occurs when the UAVs dynamic state variables are recorded by the sensor module under various flight maneuvers and sent over to the intelligent module using the serial protocol involving the UART. During this time, the data is recorded and managed by 2 python scripts on the

intelligent module namely *datamanager.py* and *serialcom.py.* The former is responsible for collecting and the right amount of data (samples) for every flight episode and managing all storage directories. The latter is responsible for setting all communication parameters such as the baud rate. It also handles any errors that may be raised during sampling, organizes the data's features, and establishes the serial connection.

Figure 13: The raspberry pi as the intelligent module

ANN host: As the host for the Artificial Neural Network, it is also responsible for running inferences or predictions of the UAVs position in real-time from the data sent over by the sensor module. Here, only the input features from the IMU are used for the UAV's position estimation by the Artificial Neural Network. It does this by running the *inference.py* python script in real-time which imports the TensorFlow-lite-runtime library for this purpose. After deployment, the inference time of the trained ANN was determined to have an inference time of 0.05 seconds using the raspberry pi's 1.4GHz processor. Therefore, the mount of compute required to satisfy the specification of 1 second was calculated to be approximately 84MHz.

4.1.4 Power Module

The power module consists of the UAV's battery seen in and voltage regulator integrated circuit device both seen **Figure 14** . The battery is a 2S 7.4V Lithium-ion battery with 2000mAh capacity. The voltage regulator used was the L7805 which produces a 5V dc output. 10uF input

and output capacitors were used to keep the power stable. This is also seen in **Figure 15.** The total amount of power drawn by the entire system was approximately 5W. The 2000mAh UAV battery is able of supply 2A at 7.4V continuously for 1 hour.

Figure 14: 2S ,7.4V ,2000mAh Li-ion UAV battery on the left, L7805 voltage regulator on the right

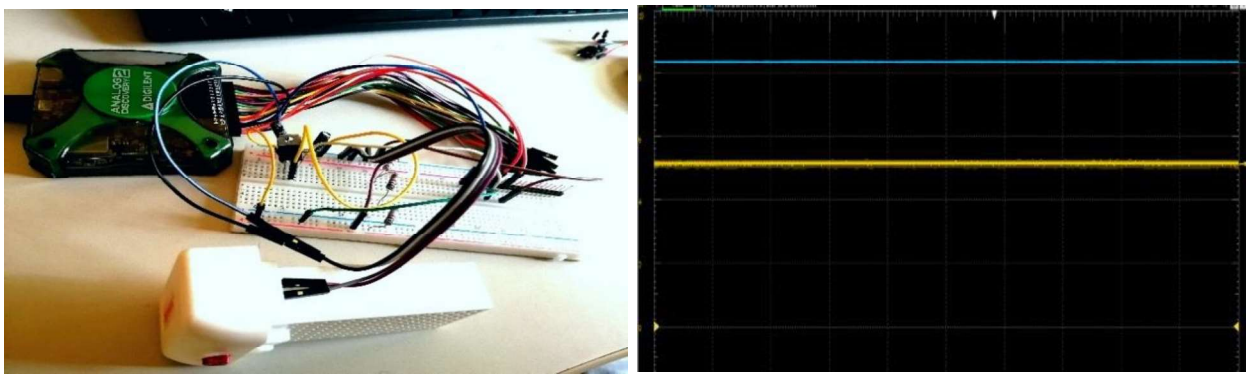


Figure 15: During the power module implementation, the fully charged UAV battery at 8.4V (blue line on scope) is regulated by the L7805 giving a 5V output (yellow line on scope)

4.1.5 Device Casing

The device casing is made up of 2 parts. The top and the bottom casings. These two come together to form an enclosure around all the electronics and circuitry to give support to the components. The top casing has extrusions designed to fit and attach itself to the UAV's removable camera mount shown in **Figure 16.**

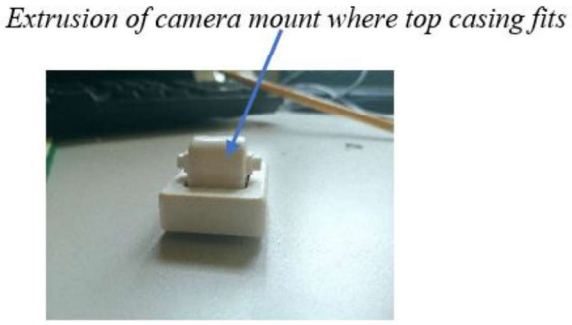


Figure 16: UAV's removable camera mount

This attaches to the UAV and carries the device. These were 3d modelled and designed using the SolidWorks software after which the parts were 3d printed and assembled with all the electronics. The material used was PLA. This is shown in **Figure 17.**

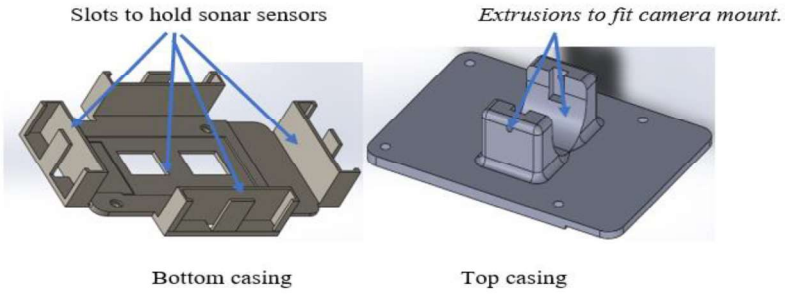


Figure 17: Top and bottom casing of device

4.1.6 Device Assembly

The device assembly took 2 forms. The circuitry is seen in **Figure 18**, and the physical assembly of the device attached to the UAV is shown in **Figure 19** below.

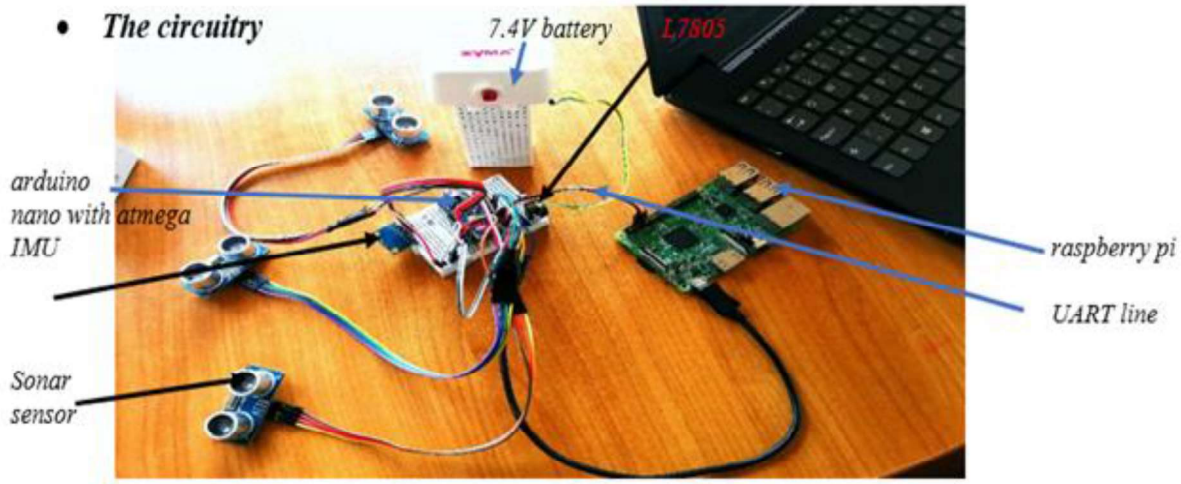


Figure 18: Circuitry with all the electronics

- ***Physical assembly***

Figure 19: Full realization of device attached to base of UAV using its camera mount.

4.2 Data Collection

In his phase, data made up of features representing the state variables of the UAV was recorded and stored. The data measured by the sensor module is sent via the UART/serial interface to the intelligent module and stored locally for a short while. Then it is pushed to the project repository on GitHub where it is later pulled at a convenient time onto another remote or local machine for analysis, preprocessing and training of the Artificial Neural Network.

In his phase, data made up of features representing the state variables of the UAV was recorded and stored. The data measured by the sensor module is sent via the UART/serial interface to the intelligent module and stored locally for a short while. This process is managed by 2 python scripts namely, *datamanager.py* and *serialcom.py.* The former manages the data in the raspberry pi's local directory and he latter gets the data from the sensor module. The data is then pushed to the project repository on GitHub where it is later pulled at a convenient time onto another remote or local machine for analysis, preprocessing and training of the Artificial Neural Network. This process is visualized in **Figure 20**.

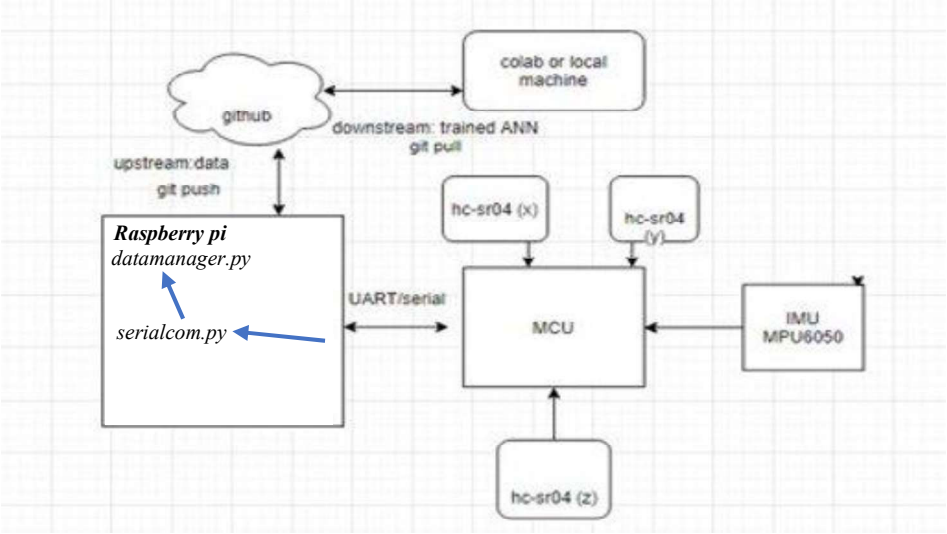


Figure 20: Data collection pipeline

The process: At the onset, the UAV with the device attached is placed in the data collection environment as shown in **Figure 21**. The environment characteristic is a 3d corner like a 3-dimensional cartesian plane.

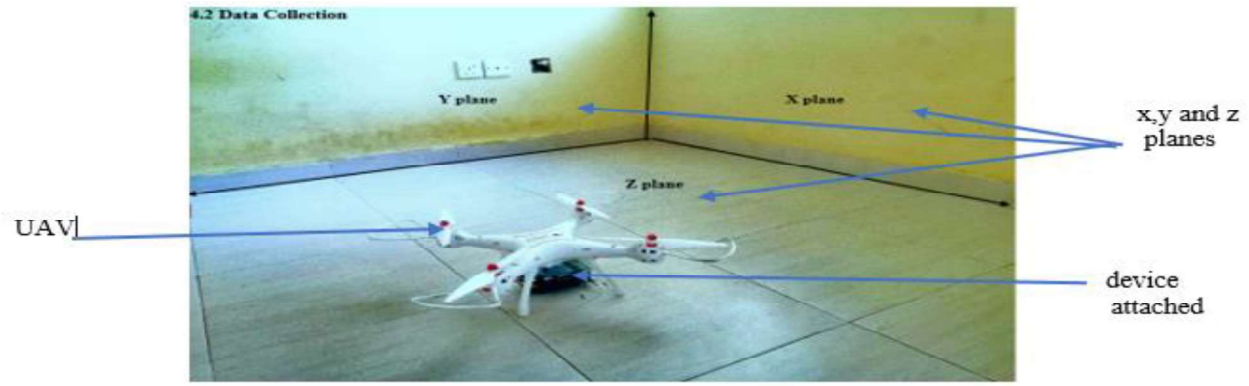


Figure 21: UAV in data collection environment -3d corner for data collection.

After powering on the device, the sensor module automatically calibrates certain essential parameters before sent to the intelligent module. These calibrations involve zeroing out the IMU and the 3-axis position from the 3 sonar sensors. After this is done, the *datamanager.py* script is run as shown in **Figure 22** to start the data collection process. There are 2 categories of flight data collected. The first is labelled *ground hold* and the second, *3d translate.* The former is implemented as a local directory that holds stores episodes of flight data when the UAV is stationary. This presents real-examples to the Artificial Neural Network of a events when the IMU is held at a fixed 3-axis position and not moving. The latter is just for normal flight maneuvers in any direction. Though there are not specific categories in flight, the data is collected this way to ensure that real word examples of when the UAV is stationary is provided to the ANN and to keep track of how much data has been collected for a certain maneuver.

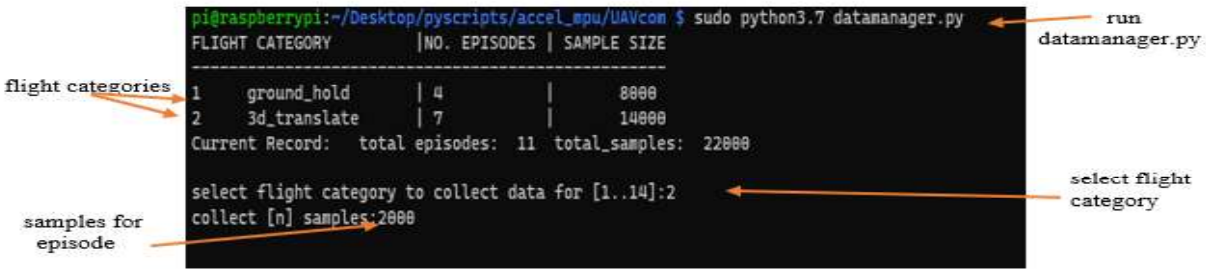


Figure 22: running datamanager.py.

4.3 Model Training

The model training phase follows after the collected data has been pushed or uploaded to the GitHub project repository. The steps taken before and after the real training of the Artificial Neural Network are the following.

4.3.1 Data download

The data required for training the Artificial Neural Network or ANN is downloaded from the project repository into the training environment. Google Colab or jupyter notebook provides python environments that can be used for such a purpose. The 'wget' command was used in the Google Colab environment to download the data from the GitHub remote project repository. This is seen in **Figure 23**.

```
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-10%2008:23:37.484231.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-10%2008:26:02.411146.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-23%2009:54:44.197761.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-23%2009:56:40.992664.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-23%2009:57:40.912381.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-23%2010:00:57.576887.pickle
!wget -x https://github.com/willakuffo/accel_mpu/raw/main/UAVcom/flight_data/3d_translate/UAV_episode2021-03-23%2010:01:53.840230.pickle
```

Figure 23: Google Colab code cell showing 'wget' method for downloading the data.

4.3.2 Reading data

At this point, the data is now in the training environment's (Google Colab/ jupyter notebook) local directory. The data is read using the *pandas* library into dataframes. This is shown in **Figure 24.**

```
#3d translate
df_3d0 = pd.DataFrame(load_data('3d_translate',0))
df_3d1 = pd.DataFrame(load_data('3d_translate',1))
df_3d2 = pd.DataFrame(load_data('3d_translate',2))
df_3d3 = pd.DataFrame(load_data('3d_translate',3))
df_3d4 = pd.DataFrame(load_data('3d_translate',4))
df_3d5 = pd.DataFrame(load_data('3d_translate',5))
df_3d6 = pd.DataFrame(load_data('3d_translate',6))

#ground_hold
df_gnd0 = pd.DataFrame(load_data('ground_hold',0))
df_gnd1 = pd.DataFrame(load_data('ground_hold',1))
df_gnd2 = pd.DataFrame(load_data('ground_hold',2))
df_gnd3 = pd.DataFrame(load_data('ground_hold',3))
```

Figure 24: Code cell reading data into pandas dataframes.

4.3.3 Data cleaning

To ensure that the data has no null values or NaNs (Not a Number), rows containing any null and NaNs were dropped from the Pandas dataframe into which the data was read into. This is shown in **Figure 25** using the *dropna* function.

```
df_3d0.dropna(axis = 0,how = 'any',inplace = True)
df_3d1.dropna(axis = 0,how = 'any',inplace = True)
df_3d2.dropna(axis = 0,how = 'any',inplace = True)
df_3d3.dropna(axis = 0,how = 'any',inplace = True)
df_3d4.dropna(axis = 0,how = 'any',inplace = True)
df_3d5.dropna(axis = 0,how = 'any',inplace = True)
df_3d6.dropna(axis = 0,how = 'any',inplace = True)
```

Figure 25: An example code cell dropping rows with NaNs or null values.

4.3.4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) involved searching the dataset prior to the model's training to learn insights that may be present in the data. **Figure 26** shows the various features and

their corresponding labels used in the training environment. Their corresponding feature symbols are also indicated. These feature symbols are the same as in **Figure 11.**

Feature/state variable label **n** = axis => x, y, z	**Description**	**Symbol**
accel_ms_**n**	IMU 3-axis acceleration in x, y, z axes in m/s^2	*ap*
accel_ang_**n**	IMU 3-axis orientation angles in x, y, z in degrees	a_{ang}
IMU_vel_**n**	IMU estimate of velocity in x, y, z in m/s	*vt*
IMU_dist_**n**	IMU 3-axis position in x, y, z in meters (m)	*St*
gyro_deg_**n**	IMU gyroscope angular orientation rates in x, y, z in degrees per second	*gp*
chng_vel**n**	Magnitude change in 3-axis IMU velocity in x, y, z	*dv*
chng_a**n**p	Magnitude change in IMU 3-axis accelerations in x, y, z	*da*
gyro_ang_**n**	IMU gyroscope angles in x, y, z	g_{ang}
com_pitch	Complementary filter angle for UAV pitch	C_{pitch}
com_yaw	Complementary filter angle for UAV yaw	*Cyaw*
com_roll	Complementary filter angle for UAV roll	*Croll*
travel_**n**	Sonar sensor 3-axis position in x, y, z. This is the true position of the UAV	*trv*

Figure 26: Table of labels and descriptions

- ***Magnitude of feature relationships:*** To have an idea of which features or state variables within the dataset contributed more to the output (3-axis positions of the UAV), the Spearman's rank correlation was used to find the degree of monotonic relationship that

existed between axis features or state variables and their corresponding axis position. For example, the distance moved by the UAV in the x direction is the x-axis position. X-axis features or state variables will be the all the state variables that occur in that axis such as acceleration in the x-axis labeled as '*accel_ms_x'*. The absolute value of the correlation was used because the sign in this application only showed the orientation or direction of the of the state variables with respect to the UAV's orientation or direction. These are all vector quantities. The direction of relationship therefore was ignored as it was dependent on these factors. Only the magnitude of monotonic relationships was focused on.

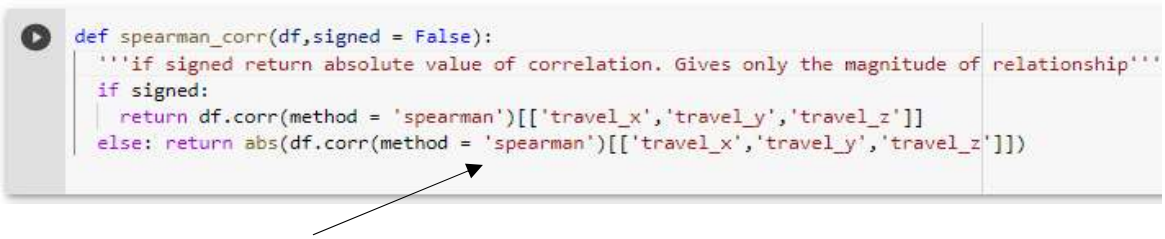


Figure 27: function that calculates using pandas correlation function.

	travel_x
chng_velx	0.097197
accel_ang_x	0.085161
IMU_vel_x	0.076298
gyro_deg_x	0.073837
chng_distx	0.071045
accel_ms_x	0.058626
com_pitch	0.011008
gyro_ang_x	0.010308
IMU_dist_x	0.006443
chng_axp	0.002421

	travel_y
com_roll	0.295265
gyro_ang_y	0.294607
chng_disty	0.219042
IMU_dist_y	0.204469
IMU_vel_y	0.196489
accel_ms_y	0.098656
chng_vely	0.091579
accel_ang_y	0.025219
gyro_deg_y	0.007059
chng_ayp	0.003788

	travel_z
IMU_dist_z	0.196427
com_yaw	0.193224
gyro_ang_z	0.193084
IMU_vel_z	0.135400
chng_distz	0.100477
gyro_deg_z	0.080778
chng_velz	0.038865
accel_ms_z	0.037778
chng_azp	0.000799

Figure 28: Absolute value of Spearman's rank correlation showing the magnitude of monotonic relationship between axis features (state variables) and axis position.

From **Figure 28,** the green color represents the all the features or state variables and their Spearman's rank correlation in descending order associated with the x-axis of the UAV. The red

color represents the y-axis and the blue, the z-axis. From the results shown in **Figure 28** per the data collected, *chng_velx* (whose feature symbol is **dv**) contributed more to the x-axis position of the UAV. This means that the more the magnitude of the velocity in the x-axis changed, the more likely the x-axis position (distance) of the UAV changed as well. For the y-axis, the greatest contributor was *com_roll* (whose feature symbol is **Croll**).This also meant that the more the Complementary filter roll angle of the UAV changed, the more likely there was going to be a change in the y-axis position of the UAV. This makes sense considering that UAV has to pitch or roll in a particular axis in order to translate through that axis. And finally, for the z-axis *IMU_dist_z* (whose feature symbol is St) contributed more to the z-axis position (altitude) of the UAV.

- ***Feature Visualizations:*** For the second part of Exploratory Data Analysis (EDA), some episodes of the UAV's flight data belonging to the 2 different flight categories were visualized. This revealed visually, how some of the state variables (which are the dataset's features or columns) behaved while the UAV was in flight. This can be seen in **Figure 29** and **Figure 20.**

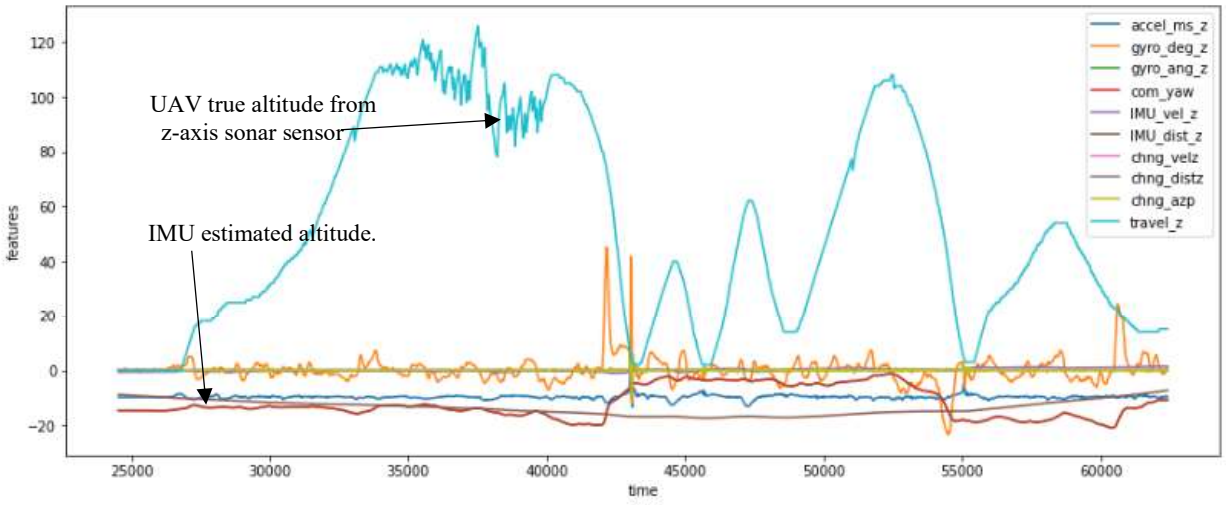


Figure 29: A visualization of the state variables or features of one **3d translate** episode of the UAV's flight data. The z-axis features (state variables) are isolated and visualized.

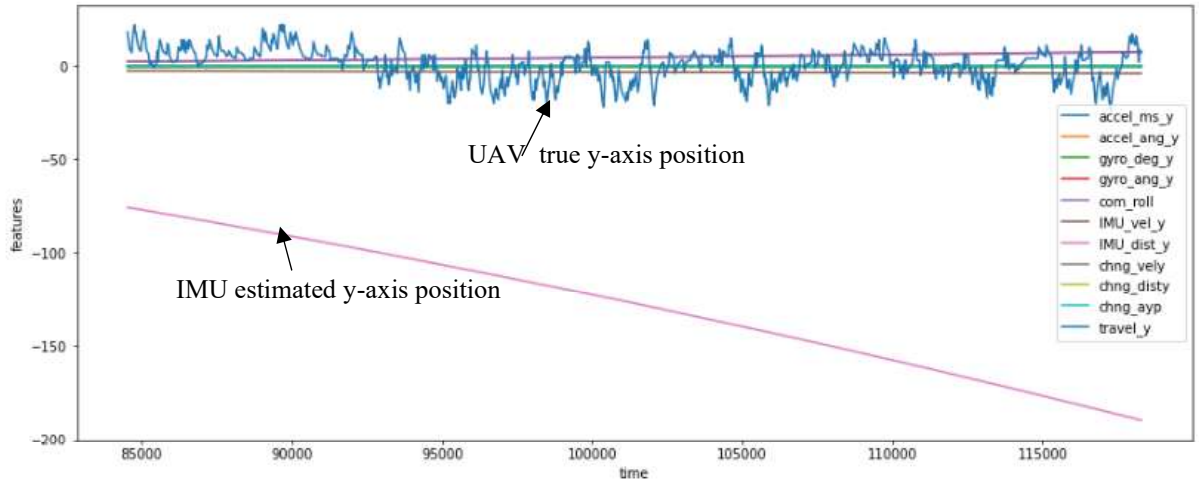


Figure 30: A visualization of the state variables or features of one **ground hold** episode of the UAV's flight data. The y-axis features (state variables) are isolated and visualized.

4.4.4 Data pre-processing – Feature Normalization and Scaling

This section involved scaling and normalizing the data. The features in the dataset were transformed to fit within a range of 0 to 1 using a minimum-maximum scaler. This is shown in **Figure 32.** This was achieved by using sklearn's preprocessing library as shown in **Figure 31.**

```
from sklearn.preprocessing import MinMaxScaler
input_scaler = MinMaxScaler()
output_scaler = MinMaxScaler()
scaled_input = input_scaler.fit_transform(data_input,y = data_output)
scaled_output = output_scaler.fit_transform(data_output)
```

Figure 31: Code cell showing scaling and normalization of data features.

Unscaled

accel_ms_x	accel_ms_y	accel_ms_z
-0.01	-0.00	-9.78
0.03	-0.04	-9.73
-0.02	-0.02	-9.78

scaled

accel_ms_x	accel_ms_y	accel_ms_z
0.412262	0.397924	0.383113
0.413672	0.394464	0.386263
0.411910	0.396194	0.383113

Figure 32: Code cell showing scaling and normalization of data features.

$$X_{scaled} = \frac{Xi - Xmin}{Xmax - Xmin} ,$$

where X_{scaled} is the scaled feature or state variable,
X_{max} is the maximum value of the feature or state variable within the entire dataset,
X_{min} is the minimum value of the feature or state variable within the entire dataset and,
X_i is each sample of the state variable or feature to be scaled.

4.4.5 Train test split

The data was split into two unequal parts. The first part made up 70 % and the second, 30%. The larger portion was used to train the Artificial neural Network and the smaller portion for testing or validating the trained model's performance. This portion of the data is unseen by the model during training therefore when used to test, represents an idea of how well the model will do on real world data which is unseen just the same. The total size of the dataset after preprocessing and dropping NaNs and null values was 12,449. After splitting, 70 % that made up the training data was 8,714 samples and 30% that made up the testing or validation data was 3,735 samples. All in all, the number of state variables or features in the dataset was 34. Three of which were the output variables. These were *travel_x, travel_y* and *travel_z* which represented the 3-axis UAV positions measured by the sonar sensor. Therefore, the input state variables were 31.

4.4.6 Model build

The ANN model was built from the Keras Sequential class with eight hidden layers with ReLu (Rectified Linear Unit) activation. A linear activation function is used at the output layer because the ANN is designed to be a regression model. **Figure 33** shows the code that implements and compiles the ANN model. The built and compiled ANN model is shown in **Figure 34.**

```
def create_model(optimizer = None,loss = None):
  model = Sequential()
  model.add(Dense(units = xtrain.shape[1],input_dim = xtrain.shape[1],activation= 'relu'))
  model.add(Dense(units = 64,activation = 'relu'))
  model.add(Dense(units = 128,activation = 'relu'))
  model.add(Dense(units = 256,activation = 'relu'))
  model.add(Dense(units = 512,activation = 'relu'))
  model.add(Dense(units = 256,activation = 'relu'))
  model.add(Dense(units = 128,activation = 'relu'))
  model.add(Dense(units = 64,activation = 'relu'))
  model.add(Dense(units = 8,activation = 'relu'))
  model.add(Dense(units = ytrain.shape[1],activation = 'linear'))
  model.compile(optimizer = optimizer,loss = loss)
  return model
```

Input layer with relu

hidden layers with relu activation

output layer with linear activation

Figure 33: Function in code cell defining the ANN model.

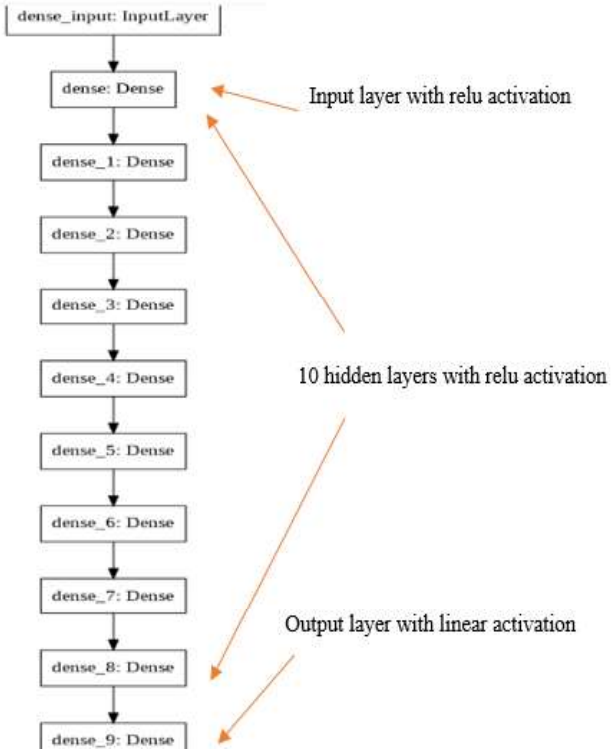


Figure 34: Model Architecture.

4.4.7 Model fitting

Under this phase, a training experiment is done to fit the Artificial Neural Network to the data (using the IMU data as input and the sonar data as output) using a number of selected loss functions and optimizers shown in **Figure 37**. For the training experiment, the goal was to train different ANN models using different optimizer-loss pair combinations, from which the best is select after performance evaluation. In total, 18 ANN models were trained all the possible optimizer-loss pair combinations in **Figure 37.**

Two callbacks are employed in the fit functions to improve performance. This is shown in the train experiment function in **Figure 36.**

- *ReduceLROnPLateau (Reduce Learning Rate on Plateau):* This callback reduces the learning rate by a set or desired factor after a number of iterations (epochs) through the dataset while training or fitting is ongoing. The factor was set to 0.1. The number of epochs to wait before reducing the learning rate known as the patient factor. This was set to 10. The validation loss (error) is used as a monitoring metric. If after 10 iterations without any improvement o reduction (this is the plateau) in this score, the learning rate is reduced by the set factor (patient factor) to allow the algorithm take smaller steps than before to increase the chances of the optimizer finding a global minimum with respect to the loss function.

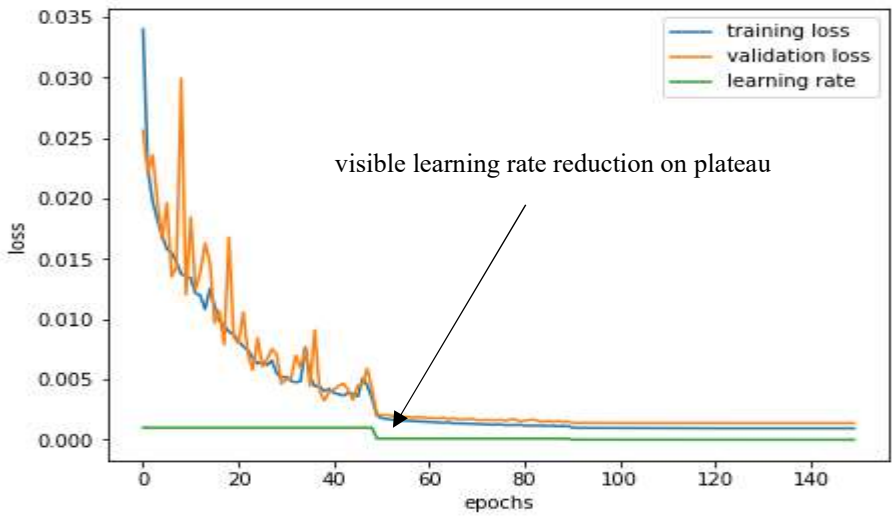


Figure 35: Model training history showing the model's loss over epochs.

In **Figure 35** the green line represents the learning rate through the entire training process or epochs. At the 45th epoch, a plateau in the validation loss occurred which triggered the *ReduceLROnPLateau* callback to reduce the learning rate by the patient factor. The corresponding effect is a visible reduction in the validation and training loss (which is an improvement) shown by the blue and orange lines.

- *ModelCheckPoint:* This call back is used to save the model at the epoch or iteration when the validation loss or error is at its minimum during training. This is able to check the problem of over fitting as the model's weights are saved only when the validation loss improves by reducing.

```
def train_on_optimizer_loss(optimizer = None,loss = None,epochs = 1):
    model = create_model(optimizer=optimizer,loss = loss)
    model_name = 'accelANN_'+loss+'_'+optimizer
    print('training',model_name)
    callbacks = [ModelCheckpoint(model_name+'.hdf5',monitor = 'val_loss',mode = 'min',save_best_only = True,verbose = 1),
                 ReduceLROnPlateau(monitor= 'val_loss',factor = 0.1,patience = 10,mode = 'min',verbose = 1)]


    hist = model.fit(x = xtrain,y = ytrain,epochs = epochs,callbacks = callbacks,validation_data= (xtest,ytest))

    pred_on_train = pd.DataFrame(output_scaler.inverse_transform(model.predict(xtrain)),columns = ['travel_x_pred','travel_y_pred','travel_z_pred'])
    train_ground_truth = pd.DataFrame(output_scaler.inverse_transform(ytrain),columns = output_features)

    test_ground_truth = pd.DataFrame(output_scaler.inverse_transform(ytest),columns = output_features)
    pred_on_test = pd.DataFrame(output_scaler.inverse_transform(model.predict(xtest)),columns = ['travel_x_pred','travel_y_pred','travel_z_pred'])


    model_r2_train = r2_score(train_ground_truth,pred_on_train)
    model_r2_test = r2_score(test_ground_truth,pred_on_test)
    model_train_loss = model.evaluate(x = xtrain,y = ytrain)
    model_test_loss = model.evaluate(x = xtest,y = ytest)

    print('modelR2 on train: ',model_r2_train,'model R2 on test: ',model_r2_test,'model train loss: ',model_train_loss,'model test loss :',model_test_loss)
    return [hist,model_r2_train,model_r2_test,model_train_loss,model_test_loss,model_name]
```

callbacks

Figure 36: Function that runs training experiment.

```
[ ]  optimizers = ['SGD','RMSprop','Adam','Adadelta','Adagrad','Nadam']
     loss_fncs = ['mean_squared_error','mean_absolute_error','mean_squared_logarithmic_error']
```

Figure 37: Optimizers and loss functions used in training experiment.

4.4.8 Model Validation

The trained ANN model was validated and tested on 30 % of the entire dataset. This made up 3,537 samples or real-world validation examples. Since this portion of the dataset was not seen by the model during training, performance evaluation on this data gives an idea of how well the trained neural network would perform on real-world data. The trained model was validated on the unseen data and ranked in two ways.

- *Overall Ranking by Coefficient of determination:* Using the coefficient of determination or R^2 metric, all 18 models were ranked according to how well their predictions on the validation data fitted to the ground truth from the sonar sensors.

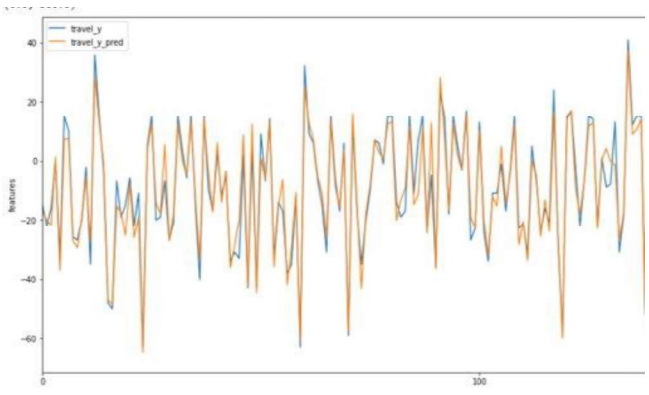
	R2_score
accelANN_mean_absolute_error_Nadam	0.959141
accelANN_mean_squared_error_Nadam	0.957425
accelANN_mean_squared_error_Adam	0.954451
accelANN_mean_squared_error_RMSprop	0.938292
accelANN_mean_squared_logarithmic_error_Nadam	0.931257
accelANN_mean_absolute_error_RMSprop	0.928455
accelANN_mean_squared_logarithmic_error_Adam	0.848681
accelANN_mean_squared_logarithmic_error_RMSprop	0.810103
accelANN_mean_absolute_error_SGD	0.470616
accelANN_mean_squared_error_SGD	0.360145
accelANN_mean_squared_logarithmic_error_SGD	0.347193
accelANN_mean_squared_error_Adagrad	0.320595
accelANN_mean_absolute_error_Adadelta	0.181550
accelANN_mean_squared_error_Adadelta	0.095828
accelANN_mean_absolute_error_Adagrad	0.025942
accelANN_mean_absolute_error_Adam	-0.064991
accelANN_mean_squared_logarithmic_error_Adagrad	-1.641793
accelANN_mean_squared_logarithmic_error_Adadelta	-6.936672

Figure 38: R^2 model ranking.

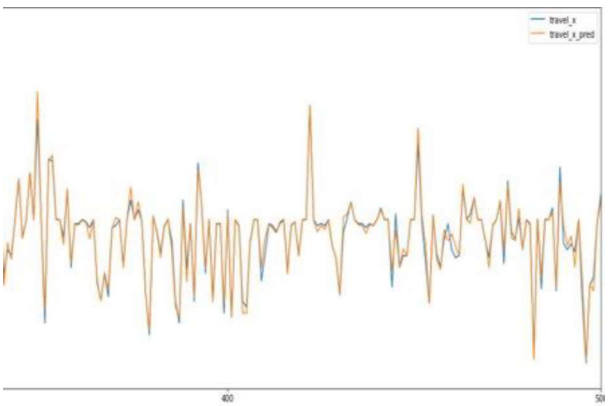
The model ranking results from **Figure 38** shows that the best model was the one trained with the mean absolute error loss and the Nadam optimizer which achieved a coefficient of determination of 0.959 on the validation data.

The figure below (**Figure 39**) shows the selected model's prediction on the validation data. The blue line represents the ground truth of the UAV's position and the orange line is the predicted output by the ANN.

x-axis position



y-axis position



z-axis position

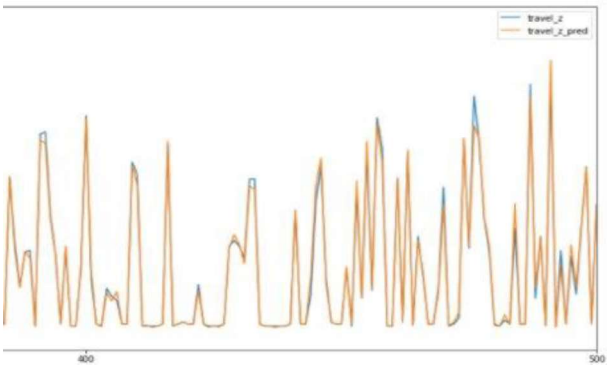


Figure 39: Selected model's predictions on the validation data of the UAV positions

- *Best optimizer per loss function :* For all the loss functions used in the training experiment, all optimizers were ranked. This revealed which optimizer performed best for a given loss function.

Optimizer	**Loss Function**					
	Mean absolute error		***Mean squared error***		***Mean squared logarithmic error***	
	R^2	*loss*	R^2	*loss*	R^2	*loss*
Nadam	0.959	0.017	0.957	0.0014	0.931	0.0009
Adam	-0.065	0.146	0.954	0.0015	0.849	0.0012
RMSprop	0.928	0.023	0.938	0.0019	0.810	0.0009
Stochastic Gradient Descent (SDG)	0.470	0.082	0.360	0.0208	0.347	0.0111
Adagrad	0.026	0.123	0.320	0.0230	-1.641	0.0292
Adadelta	0.181	0.119	0.095	0.0336	-6.937	0.1104

Figure 40: model ranking – optimizer per loss.

The best optimizer-loss pair combination was the Nadam optimizer and the mean absolute error loss achieving a validation R^2 score of 0.959 and a loss of 0.017. For a given loss function, there was an optimizer that performed the best. Generally, the Nadam optimizer was the best optimizer regardless of the loss function used. This can be seen in **Figure 38** and **Figure 40.**

4.5 Deployment and Real-World Testing and Model Iteration

After selecting the best performing ANN model (based on the R^2 score on the validation data), it was converted to a TensorFlow lite model using the TensorFlow lite converter. This is shown in **Figure 41.** During this process, the trained 64-bit floating-point weights are converted to 32-bit floating-point weights. This allows for faster computational times during inference.

Figure 41: Code cell to convert model to TensorFlow lite model

The model is then put on the raspberry pi and allowed to make real world inferences or estimations using the IMU data only. Data from the real-world was used to evaluate te model. At this point, the architecture is the one shown in **Figure 6A.**

Model Iteration: The initial training sample size of 8,253 was increased to 42,110 by collecting more data. This was done to investigate the effect of the sample size on the model's real-world performance. As seen in **Figure 39**, even though the initial selected trained model performs really well on its validation data, thus achieving an R^2 of 0.959, this may not be representative of real-world data and performance and as such, more data was collected to investigate this effect.

Using the best optimizer-loss pair a newer model was trained on new larger data having 42,110 samples. Two ANN models (the first trained on 8,253 samples and second trained on 42,110) were both used to perform predictions in the real-world. The positions estimated by the two models are shown in **Figure 42** (showing the x estimated position), **Figure 43** (showing the y estimated position) and **Figure 44** (showing the z estimated position). The R^2 validation of the estimated against the true positions as shown in **Figure 45.** In these figures, the blue line represents the UAV's true position reported by the sonar sensors, the orange line represents the estimates of model 1 (trained on 8,253 samples), and the green line represents the estimates made by model 2 (trained on 42,110 samples).

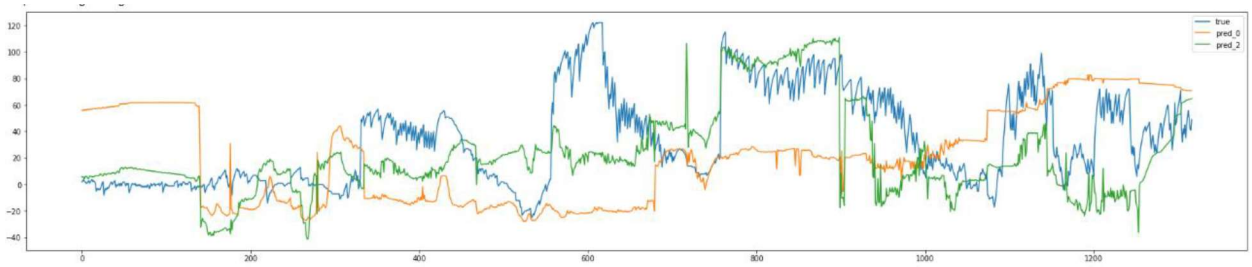


Figure 42: x position estimation of model 1 and 2

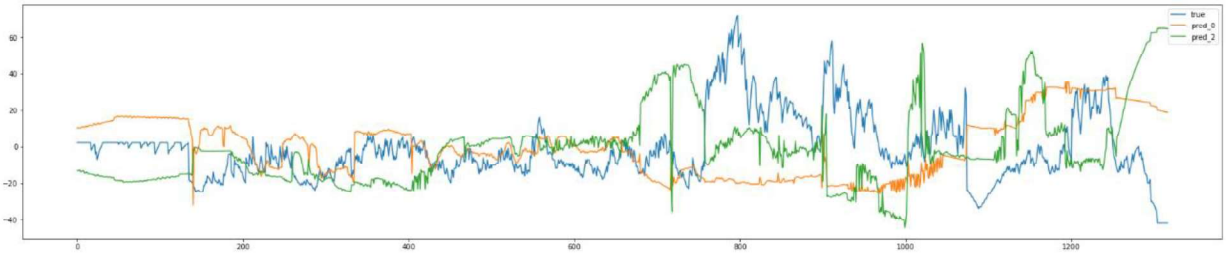


Figure 43: y position estimation of model 1 and 2

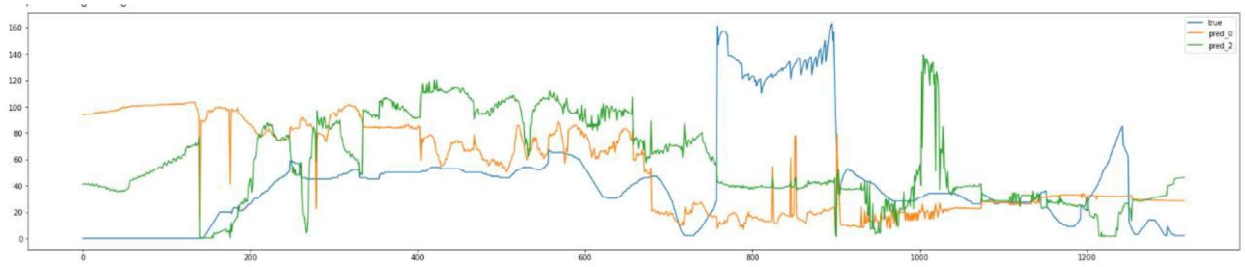


Figure 44: z position (altitude) estimation of model 1 and 2

The results showed that a drastic increase in the training sample size resulted in a rather small improvement in the real-world performance of the ANN. However, increasing the training sample size still improved the real-world performance of the ANN in predicting the UAV's 3-axis position.

Model	**Training Sample Size**	**Real World R^2**
1	8,253	-1.33
2	42,110	-0.87

Figure 45: Model training sample size and real-world R^2 performance

Chapter 5: Discussion and Conclusion

The problem of making an IMU a standalone positioning determining system is an interesting one. However, this is quite hard to achieve. The technique employed in this paper utilizes an Artificial Neural Network as an estimator, taking data inputs from feature engineered state variables or parameters from an IMU. This data is used to predict the 3-dimensional position of the UAV in the x, y and z cartesian coordinates. However, with very good performance of the ANN on the validation or unseen data obtained during data collection, it was not able to replicate a similar ability of performance on real-world data. As a result, the ANN lacked the required generalization capability required on real-world data. Even though a substantial amount of data was collected, this data also was not very representative of the real-world data. The results, however demonstrated that, the ANN was able to model the relationship between the collected input data from IMU and the true UAV positions reported by the sonar sensor. But since this data was still not representative of all the real-world data, the ANN failed to generalize to maintain a very good performance in the real-world. The results also showed that a substantial increase in the size of the training set generated a little improvement in the real-world performance and generalization capability of the ANN. This means with more data, better estimations in the real-world could be achieved.

5.1 Limitations

The following were things that served to obstruct project progress and consequently the overall performance of the Artificial neural Network. These limitations prevented the collection of enough data which could have boosted the performance of the Artificial Neural Network.

- Serial Exception Errors: These errors were randomly thrown during the data collection process while the UAV was in flight. This caused a significant reduction in the total data collected during flight episodes.
- Inadequate UAV battery life: With a very long charging time and a short battery life after charging, a lot of time was spent recharging the UAV battery rather than collecting data. The short battery life also meant very little data could be collected after each charge cycle.

5.2 Future Works

Overall, the design and technique proposed by this paper are feasible. The following are some recommendations to improve the outcome of the technique used in this paper.

- Collecting lots of data to train the ANN and improve its generalization capability in the real-world
- Expand the size of the Artificial Neural Network by adding more hidden layers to allow the learning of more complex relationships which may not have been captured by this model. This would help tackle edge cases not seen by the model during training.
- Develop a more efficient data collection algorithm to improve data collection efficiency.
- Doing more feature engineering to present more state variables for better generalization.
- Improving the weight and form factor of the device built. This will improve power consumption of the UAV during flight as it will carry less weight.

References

[1] Q. Zhang and B. Li, "A low-cost GPS/INS integration based on UKF and BP neural network," in *Fifth International Conference on Intelligent Control and Information Processing*, Aug. 2014, pp. 100–107, doi: 10.1109/ICICIP.2014.7010322.

[2] H. Koyuncu and S. H. Yang, "A Survey of Indoor Positioning and Object Locating Systems," p. 8, 2010.

[3] K. Cheung, G. Lightsey, and C. Lee, "Accuracy/computation performance of a new trilateration scheme for GPS-style localization1," in *2018 IEEE Aerospace Conference*, Mar. 2018, pp. 1–10, doi: 10.1109/AERO.2018.8396377.

[4] Sara. Benkouider, Nasreddine. Lagraa, Mohamed. B. Yagoubi, and Abderrahmane. Lakas, "Reducing complexity of GPS/INS integration scheme through neural networks," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Jul. 2013, pp. 53–58, doi: 10.1109/IWCMC.2013.6583534.

[5] W. Li and Z. Fu, "Unmanned aerial vehicle positioning based on multi-sensor information fusion," *Geo-Spat. Inf. Sci.*, vol. 21, no. 4, pp. 302–310, Oct. 2018, doi: 10.1080/10095020.2018.1465209.

[6] M. Malleswaran, S. A. Deborah, S. Manjula, and V. Vaidehi, "Integration of INS and GPS using radial basis function neural networks for vehicular navigation," in *2010 11th International Conference on Control Automation Robotics Vision*, Dec. 2010, pp. 2427–2430, doi: 10.1109/ICARCV.2010.5707295.

[7] Honghui Qi and J. B. Moore, "Direct Kalman filtering approach for GPS/INS integration," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 2, pp. 687–693, Apr. 2002, doi: 10.1109/TAES.2002.1008998.

[8] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/science.aaa8415.

[9] G. D. Buckner, K. T. Schuetze, and J. H. Beno, "Active vehicle suspension control using intelligent feedback linearization," in *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, Chicago, IL, USA, 2000, pp. 4014–4018 vol.6, doi: 10.1109/ACC.2000.876976.

[10] C. K. Chui and X. Li, "Approximation by ridge functions and neural networks with one hidden layer," *J. Approx. Theory*, vol. 70, no. 2, pp. 131–141, Aug. 1992, doi: 10.1016/0021-9045(92)90081-X.

[11] A. Sharma *et al.*, "Communication and networking technologies for UAVs: A survey," *J. Netw. Comput. Appl.*, vol. 168, p. 102739, Oct. 2020, doi: 10.1016/j.jnca.2020.102739.

[12] M. P. Vasylenko, "Testing system for unmanned aerial vehicles microelectromechanical sensors," in *2017 IEEE 4th International Conference Actual Problems of Unmanned Aerial*

Vehicles Developments (APUAVD), Kiev, Oct. 2017, pp. 180–183, doi: 10.1109/APUAVD.2017.8308804.

[13] C. Alippi, S. Disabato, and M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2018, pp. 212–223, doi: 10.1109/IPSN.2018.00049.

[14] A. Hiliuta, R. Landry, and F. Gagnon, "Fuzzy corrections in a GPS/INS hybrid navigation system," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 40, no. 2, pp. 591–600, Apr. 2004, doi: 10.1109/TAES.2004.1310007.

[15] N. El-Sheimy, K.-W. Chiang, and A. Noureldin, "The Utilization of Artificial Neural Networks for Multisensor System Integration in Navigation and Positioning Instruments," *IEEE Trans. Instrum. Meas.*, vol. 55, no. 5, pp. 1606–1615, Oct. 2006, doi: 10.1109/TIM.2006.881033.

APPENDICES

APPENDIX A: ENGINNERING DRAWING OF TOP CASING

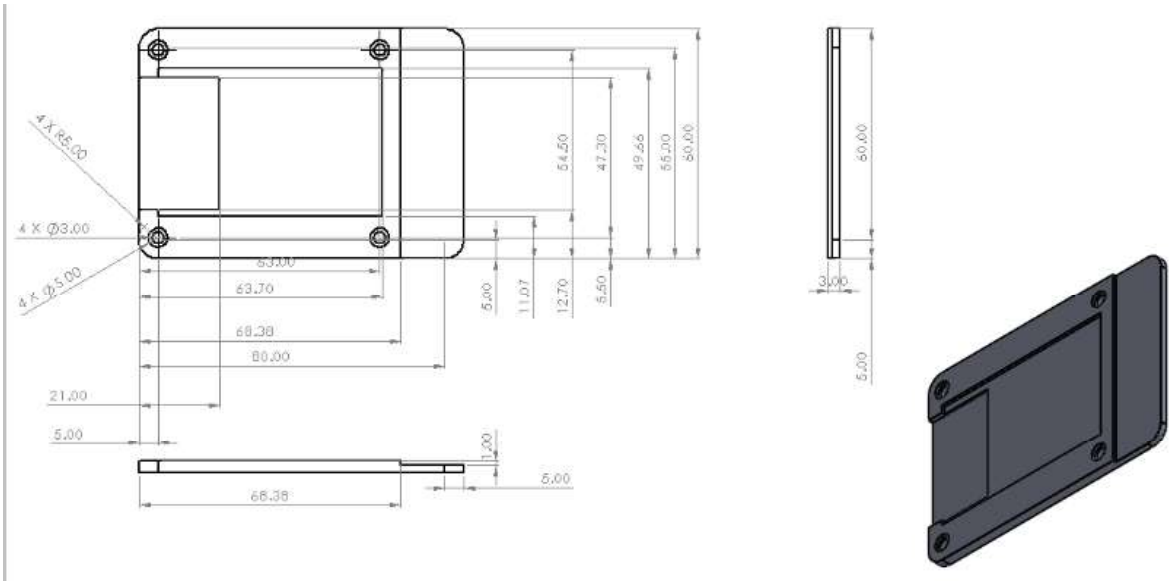


Figure 46: Engineering drawing of Top casing

APPENDIX B: ENGINNERING DRAWING OF BUTTOM CASING

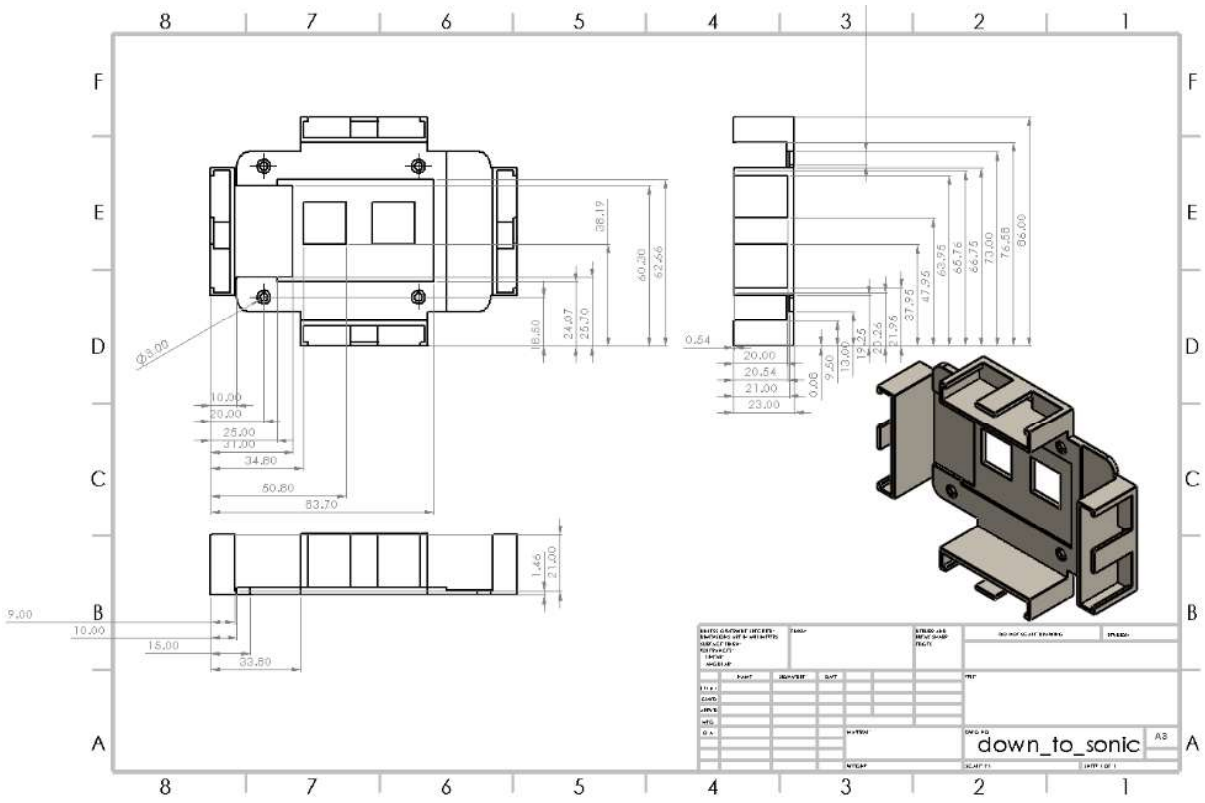


Figure 47: Engineering drawing of Bottom casing