

ADAPTIVE NEURAL NETWORK ATTITUDE CONTROL FOR A FIXED WING UAV

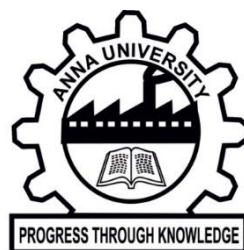
PHASE I REPORT

Submitted by

ARVINDH R (2016608004)

in partial fulfilment for the award of the degree of

**MASTER OF ENGINEERING IN
MECHATRONICS**



**DEPARTMENT OF PRODUCTION TECHNOLOGY
MADRAS INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY, CHENNAI**

NOVEMBER 2017

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Report titled “**ADAPTIVE NEURAL NETWORK ATTITUDE CONTROL FOR A FIXED WING UAV**” is the bonafide work of **ARVINDH R (Reg No. 2016608004)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. A. RAJADURAI

Professor and Head

Department of Production technology

Madras Institute of Technology

Anna University, Chennai – 600 044

Dr. P. GANESH

Assistant Professor

Department of Production technology

Madras Institute of Technology

Anna University, Chennai – 600 044

திட்டப்பணிச்சுருக்கம்

பலவிதமான பயன்பாடுகளால், ஆளில்லா வான்வழி வாகனம் உச்ச ஆராய்ச்சி ஆர்வமாக உள்ளன. சுற்றுச்சூழலில் மாற்றங்களை எதிர்கொள்ளவும், மீண்டும் திட்டமிடல், பார்வை மூலம் வழிநடத்துதல், மோதல் தவிர்த்தல் மற்றும் தொடர்பு இணைப்பு வரம்புகளுக்கு வெளியே செயல்பட முடியும் என ஆளில்லா வான்வழி வாகனத்திற்கு ஒரு அணுகுமுறை கட்டுப்பாட்டை வளர்த்துக் கொள்ள இந்த முன்மொழிவு நோக்கமாக உள்ளது. நரம்பு பின்னலமைப்பு போன்ற செயற்கை நுண்ணறிவு நுட்பத்தை பயன்படுத்தி, ஆளில்லா வான்வழி வாகனம் ஆனது அதன் தன்னியக்க நடவடிக்கைகளுக்கு தேவைகளை பூர்த்தி செய்ய திட்டமிடப்பட்டுள்ளது. நரம்பு பின்னலமைப்பு போன்ற இயந்திர கற்றல் நுட்பத்தை பயன்படுத்தி மாறும் சமன்பாட்டில் நிச்சயமற்ற காலத்தை கண்டறிவதன் மூலம், ஆளில்லா வான்வழி வாகனம் ஆனது தேவையான நடவடிக்கைகளுக்கு தன்னியக்கமாக கட்டுப்படுத்தப்படலாம் மற்றும் அதன் செயல்திறனை சரிசெய்ய மனிதத் தலையீடு தேவையில்லை. இந்த திட்டத்தின் முதல் கட்டம், ஆளில்லா வான்வழி வாகனம் இயக்கவியல் கையேடு கட்டுப்பாடு உருவகப்படுத்துதல் மூலம் வடிவமைப்பு, தரவு கையகப்படுத்தல் ஆகியவற்றை உள்ளடக்கியது மற்றும் தரவரிசைகளை பல அடுக்கு நரம்பு பின்னலமைப்பு பயிற்சிக்கு அனுப்புகிறது. உகந்த வெளியீட்டைப் பெற்றபின், நிகழ் நேர தரவு நரம்பு பின்னலமைப்பு மூலம் முன் பதிவு செய்யப்பட்ட தரவோடு சேர்த்து வழங்கப்படுகிறது. தரவு இரண்டும் ஒப்பிடப்படும் மற்றும் தரவு வேறுபாடு ஒரு பரவல் மூலம் நரம்பு பின்னலமைப்பு கருத்து இருக்கும் எடை சரிசெய்தல். வன்பொருள் சுழற்சியில் உருவகப்படுத்துதல், ஒரு முன்மாதிரி மாதிரி மற்றும் தேர்வுமுறை நிகழ்நேர சோதனை இந்த திட்டத்தின் இரண்டாம் கட்டத்தில் நடத்தப்படும்.

ABSTRACT

Unmanned Aerial Vehicles (UAV) are in peak research interest because of its wide variety of applications. This proposal aims at developing an attitude control for an UAV to react to changes in environment and be capable of re-planning, navigation through vision, collision avoidance and can be able to operate outside the communication link limits. Using artificial intelligence (AI) technique such as Neural Network (NN), the UAV is programmed for its autonomous operations to satisfy the requirements. By finding the uncertainty term in the dynamic equation using machine learning technique such as Neural Network (NN), the UAV can be controlled autonomously for the required operations and need not be required human intervention for tuning its performance. The first phase of this project covers the design, data acquisition of UAV dynamics through manual controlled simulation, and inputting the data into a multi layered Neural Network for training. After getting the optimized output, the real-time data is fed into the NN layers along with pre-recorded data. Both the data will be compared and the difference in the data will be feedback to NN through a backpropagation Weight adjustment. The hardware-in-loop simulation, real-time testing of a prototype model and optimization will be carried out in the second phase of this project.

ACKNOWLEDGMENT

I express my great heartfelt gratitude to our Dean **Dr. A. RAJADURAI**, Professor and Head of the Department, Department of Production Technology, MIT campus, Anna University, Chennai for his constant encouragement and support.

I also express my deep sense of gratitude and indebtedness to my project guide **Dr. P. GANESH**, Assistant Professor, Department of Production Technology, Madras institute of Technology, Anna University, Chennai for constant support and suggestions for carrying out this project.

I acknowledge my sincere thanks to the project committee members **Dr. R. SIVARAMAKRISHNAN**, Associate Professor, **Dr. A. SIDDARTHAN**, Associate Professor, **Dr. C. NANDAKUMAR**, Assistant Professor, **Dr. P. KARTHIKEYAN**, Assistant Professor and **Ms. S. SRIDEVI**, Teaching Fellow for their valuable suggestions during the review of project.

I also express my deep sense of gratitude and indebtedness to **Dr. S. SRINIVASAN**, Associate Professor, Department of Instrumentation Engineering, Madras institute of Technology, Anna University, Chennai for his valuable help during the project.

I am very grateful to all the staff members, research scholars of the Department of Production Technology and my friends for their valuable suggestions and help provided during this project.

ARVINDH R

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT (TAMIL)	iii
	ABSTRACT	iv
	LIST OF FIGURES	ix
	LIST OF SYMBOLS	xi
	LIST OF ABBREVIATIONS	xii
1	INTRODUCTION	1
	1.1 INTRODUCTION	1
	1.2 NEURAL NETWORKS	2
	1.3 OBJECTIVE OF THIS PROJECT	3
	1.4 NEED FOR THIS STUDY	3
	1.5 SCOPE OF THIS PROJECT	3
2	LITERATURE REVIEW	4
3	METHODOLOGY	8
4	UAV DESIGN CONSIDERATIONS	10
	4.1 AIRFOIL SELECTION	10
	4.2 DESIGN SPECIFICATION OF UAV	12
	4.3 CENTRE OF GRAVITY	13
	4.4 AERODYNAMIC CENTRE	14
	4.5 CAD MODEL DESIGN	14

5	NEURAL NETWORK MODEL	16
	5.1 INTRODUCTION	16
	5.1.1 Advantages of neural network	16
	5.2 CONTROL SYSTEM DESIGN AND	
	MODEL DEFINITION	17
	5.2.1 Inputs and outputs definition	18
	5.2.2 Neural Network Model	18
	5.2.3 Neural network parameters	20
	5.2.4 Forward propagation model	21
	5.2.5 Back propagation model	22
 6	 IMPLEMENTATION OF NEURAL	
	NETWORK MODEL	23
	6.1 OVERALL MODEL	23
	6.2 MODEL ARCHITECTURE	23
 7	 SOFTWARE IN-THE-LOOP	
	SIMULATION	25
	7.1 SITL ENVIRONMENT	25
	7.2 SOFTWARE ARCHITECTURE	
	AND CONNECTION	25
	7.3 ARDUPILOT SITL SOFTWARE	26
 8	 IMU SENSOR FUSION AND	
	CALIBRATION	29
	8.1 ABOUT IMU SENSOR MPU-9250	29
	8.2 MPU-9250 PIN OUT DIAGRAM	30

	8.3 CONNECTING IMU SENSOR WITH RASPBERRY PI	30
9	RESULTS AND DISCUSSIONS	33
	9.1 SENSOR DATA AQUISITION	35
	9.1.1 Acquiring data from MPU- 9250 IMU sensor	35
	9.1.2 Calibrating the raw data	35
	9.1.3 Calibrated sensor data after acquisition	36
10	CONCLUSION	37
	10.1 CONCLUSION	37
	10.2 WORK SCHEDULE FOR PHASE II	37
	APPENDIX I	38
	APPENDIX II	43
	APPENDIX III	46
	APPENDIX IV	47
	APPENDIX V	48
	REFERENCES	49

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.1	Methodology of the project	8
4.1	Forces on airfoil.	11
4.2	Airfoil profile of MH-60 airfoil.	12
4.3	UAV design specifications.	12
4.4	Aerodynamic centre.	14
4.5	CAD model designed using solidworks.	15
4.6	Rendered CAD model of fixed wing UAV.	15
5.1	Control system design.	17
5.2	Neural Network model with single hidden layer.	19
5.3	Neural Network model with three hidden layers.	20
6.1	Overall representation of system proposed.	24
6.2	Neural Network Architecture.	24
7.1	Software In-The-Loop Simulation Model.	26
7.2	Map of the waypoints loaded for UAV navigation in SITL environment.	27
7.3	Output of various sensor data and speed of UAV from simulation environment.	27
7.4	Various sensor and throttle data of UAV from MAVproxy in SITL environment.	28
8.1	Pinout diagram of MPU-9250 IMU sensor.	30

8.2	Connecting MPU-9250 IMU sensor through I ² C communication.	31
8.3	Connection pins of the MPU-9250 and GPIO pins of Raspberry pi.	32
9.1	three-axis Gyro sensor data obtained from simulation environment.	34
9.2	Roll and Pitch data obtained from simulation of the UAV model in SITL simulation.	34
9.3	Desired roll and pitch data compared with acquired roll and pitch data.	35
9.4	Calibration and data acquisition of MPU-9250 IMU sensor.	36

LIST OF SYMBOLS

C_d	Coefficient of drag
C_l	Coefficient of lift
C_m	Coefficient of moment
C_{root}	Root chord of the wing
C_{tip}	Tip chord of the wing
D	Drag on the surface
G	Acceleration due to gravity
L	Lift force
S	Span of the wing
S_{ref}	Control surface planform area
V	Volume of the entire vehicle
W	Overall weight of the vehicle
ρ	Density
φ	Required angle of pitch to stabilize the UAV
θ	Required angle of roll to stabilize the UAV
α	Angle of Attack

LIST OF ABBREVIATIONS

ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
AoA	Angle of Attack
API	Application Programming Interface
DNN	Deep Neural Network
FDM	Flight Dynamics Model
FFNN	Feed-Forward Neural Network
GCS	Ground Control Station
GPIO	General Purpose Input Output
I²C	Inter Integrated Circuit
IMU	Inertial Measurement Unit
MAV	Micro Aerial Vehicle
MCM	Multi-Chip Module
PID	Proportional Integral Derivative
RC	Remote Control
RMSE	Root Mean Square Error
SITL	Software In-The-Loop
UAV	Unmanned Aerial Vehicle

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Airplanes are an unrestrained rigid body in space having six degrees of freedom, three translating movements along longitudinal, lateral and vertical axis, and three rotational movements around the same axis. The rotational movements are commonly referred to as roll, pitch and yaw, and the controlled manipulation of these movements, using vehicle actuators, establishes the desired attitude. The traditional actuators on a plane are: aileron, elevator and rudder that respectively provide the roll, pitch and yaw. We also have the throttle, that controls the thrust produced by the motor. In this work, we use a fixed wing UAV which have all the traditional control surfaces mentioned. UAV is any aircraft that does not have a pilot on board, and they can be autonomous, semi-autonomous or remotely controlled. There are a huge variety of sizes, forms and functionality, being classified mainly by functionality, sustainability and dimensions.

The demand for UAV has been grown for recent decades for diverse factors in both military and civilian applications like surveillance, search and rescue, missile launching etc. The flight dynamic of an aircraft is a complex nonlinear system. Classic control theories use a simplified mathematical model to optimize the response to make the control system acceptable. As dynamic systems models present a complex behaviour, solutions to these systems should be based on new concepts and advanced controlling technologies. Nowadays, a known classical controller, proportional, integral and derivative (PID) is widely used. Due to relatively simple implementation and good performance, the PID has been largely used in all kinds of systems, even flight control ones.

The flight dynamics of an aircraft is a complex nonlinear system. Thus, it is not possible to model a high order differential equation for this control system. Classic control theories use a simplified mathematical model to optimise the response to make the control system acceptable. As a dynamic systems models present a complex behaviour, solutions to these systems should be based on new concepts and advanced controlling technologies.

However, the individuality of each flight may demand constant recalibration of the parameters of the control system when using conventional system controls, due to changes in the flight characteristics as: payload, centre of gravity, flight conditions and environment. Moreover, several methods using AI techniques have been proposed to substitute the classical controller, like artificial neural networks. But some of these solutions require a high computational power, which is not always available. As can be seen in, bringing the concept of high performance of classical controllers, with the support of an optimization architecture using ANN to get better response.

1.2 NEURAL NETWORKS

The human brain is a source of intelligence and a truly remarkable parallel computer. The brain processes incomplete information obtained by the perception at an incredibly rapid rate. Nerve cells function about 10^6 slower than an electronic circuit gates, but human brains process visual and auditory information much faster than modern computers.

Inspired by biological nervous systems, many researchers have been exploring artificial neural networks. They model the brain as a continuous-time nonlinear dynamic system in connectionist architectures that are expected to mimic brain mechanisms to simulate intelligent behaviour.

An adaptive network, as the name indicates, is a network structure consisting of several nodes connected through directional links. Each node represents a process unit, and the links between nodes connected through

directional links. Each node represents a process unit, and the links between nodes specify the casual relationship between the connected nodes.

1.1 OBJECTIVE OF THIS PROJECT

This project aims in developing a control for a fixed wing UAV using neural network algorithm for its Attitude control which is roll, pitch and yaw of the UAV, Propulsion i.e. speed control of propeller using an electronic speed control which is controlled by a neural network algorithm, Navigation and obstacle avoidance by the help of stereotyped camera connected and a payload function of the UAV which is for surveillance applications.

1.2 NEED FOR THIS STUDY

To overcome the drawbacks of conventional controllers and to provide a solution for autonomous control of a complex non-linear system like UAVs. A machine learning algorithm (Artificial Neural Network) is used in this project for providing a better solution for adaptability and for autonomous operation of a Fixed Wing UAV without human intervention beyond the control limit. This autonomous UAV will be potentially useful in surveillance, vision based navigation, military application, target detection etc. this potential use for its autonomy and decision making ability makes well suitable for military and commercial operations for various payload functions in the future.

CHAPTER 2

LITERATURE REVIEW

De Oliveira et al. (2017) used a Enhanced PID controllers used. Control using ANFIS algorithm for flight dynamics and genetic algorithm for attitude (roll, pitch and yaw) optimization. Constant recalibration and adaptation required due to varying environment. So, AI techniques can be used for adaptability.

Konam et al. (2017) designed a two layered Neural Network (NN) is used for navigation and obstacle avoidance. Vision based navigation by depth perception. Monocular camera with Kinect sensor used for depth perception. Off-board GPU is used for computation. Automatic Patch Pattern Labelling for Explanation (APPLE) algorithm is used for labelling each neuron in convolutional layers. Labelled neurons are used later for ranking the best output.

Sathyan et al. (2017) proposed a solution for Reducing the uncertainty in velocity and maneuver parameters using genetic and fuzzy algorithm. A special neuro-fuzzy technique is used. Objective is to find a conflict-free trajectory of an aircraft. Genetic algorithm is used for comparison of results from a fuzzy system.

Julian et al. (2017) proposed an optimal solutions are computed offline to guide autonomous UAVs. The problem of finding the fastest trajectory to a waypoint is framed as a Markov decision process and solved using discrete value iteration. neural network was trained from the MDP solution to accurately compress the MDP table with negligible loss. Flight results with the neural network guidance system are promising and show that neural networks have great potential as on-board guidance systems or for generating optimal trajectories.

Comparison of the trajectories generated mixed-integer linear programming under the same constraints as the MDP, negligible differences is found in the generated trajectories.

Muliadi et al. (2017) compared the advantages of ANN over conventional control method especially for the UAV attitude controller. The ANN applied the learning mechanism to inversely mimicking the output of a system dynamics with respect to its input. In this paper, they found that it has a better capability in controlling the nonlinear systems. The comparison composed from the twin ANN model usage, the conversion into PWM values, the simultaneous numerical simulation has been done. The result of this paper shows that the ANN performance was better than the PID controller in handling the UAV altitude dynamics.

Canciello et al. (2017) explains the large effect of wind due to the small scale, and the absence of aerodynamic data at low speed and Reynolds numbers are considered for designing controllers. Two nested loops, high-gain controller scheme is proposed to control attitude and both longitudinal and vertical speed. Two high-gain controllers have been considered for attitude and velocity control, and both their independent and joint use has been considered. An exponentially decaying term in the controllers and an integral action mitigate the drawbacks of classic high-gain control.

Atmeh et al. (2016) paper proposed a neurodynamic system which is used to adaptively solve the problem of trajectory generation, planning path connecting a locus of points as a function of time. It was shown that by designing an NN with specific system properties and by training two FFNN a neurodynamic system with feedback can be used to adaptively generate complex trajectories. Two examples of trajectory generation were given, one of a circle and the other of a figure eight trajectories. It was noticed that the frequency of the intermediate trajectory generated by the RNN needed to be selected based on the frequency of the desired trajectory.

Schmidhuber et al (2015) paper proposes a Deep Learning in Neural Networks which is relevant for Supervised Learning, Unsupervised Learning, and Reinforcement Learning. A deep Credit Assignment Paths is used. Dynamic Programming is important for both deep Supervised Learning and traditional Reinforcement Learning with deep NNs.

Maravall et al., (2015) proposed a Dual Feedforward / Feedback architecture has been used as the UAV's controller and the K-NN classifier using the grey level image histogram as discriminant variables has been applied for landmarks recognition. A solution to the problem of visual landmark recognition by means of a dual control architecture is proposed. So, that the convergence of the UAV's controller to a state of zero error has been achieved. As the controller's error signal, which is based on the difference between the current image captured by the UAV's on board camera and the target image.

Lee et al (2015) in this paper an Off-board computing cloud GPU is used for computation. NN can be used for real time object detection. Training based on the manually recorded data sets. Input from the real-time sensors and pre-input data sets are compared and output is generated.

Selma B. et al (2013) paper provides a solution in guiding an autonomous vehicle to a desired destination along a desired path in an environment characterized by a terrain and a set of distinct objects, such as obstacles traffic lights and cars circulating in the trajectory. Where, ANFIS architecture has demonstrated a good performance in modelling the trajectory of an Unmanned Vehicle. The best feature of ANFIS is that it pre-processes all the data into several membership functions before mapping the data into an adaptive neuro structure. This pre-processing feature allows ANFIS to converge faster and better. To be able to have a basis for comparison, an ANN type controller is also designed.

Verbickas (2013) used A fixed wing UAV model with NN based vision navigation is used. A multi-layered Neural Network (NN) used. Support vector machine is used for basic flight dynamics. Stereoscopic (binocular) cameras used

for 3D mapping of image edge feature based approach for object detection and navigation.

Rimal et al (2009) used a neural network plant model to assist in the controller training. In this paper the simulation done for the modelling capabilities of a state space neural network, to act as an observer for a non-linear process allowing a simultaneous estimation of parameters and states. This paper considered the simulation of identification of a nonlinear system dynamics using artificial neural networks approach.

Dujrene (2003) proposed a method is outlined which allows a base level application for applying an Artificial Intelligence method, Fuzzy Logic, to aspects of Control Logic for UAV flight. The approaches seen in most of the research included Expert Systems; Neural Networks, Fuzzy Logic, and Hybrid combinations of these and PID control schemes. the application was studied further to gain knowledge of the effects of aircraft speed also. This type of application will allow an Intelligent Agent approach to be implemented later. Attempts can be made to expand the use of AI techniques so a comparison can be done against more conventional flight controllers.

Chang-Su Park et al (2003) used a Model inversion control is augmented with neural network in control the airship. The input-output feedback linearization allows one to design a linear controller from a nonlinear system. Neural network is used to compensate the nonlinearities resulting from this linearization. The augmented controller is applied to control the airship's velocity, pitch and yaw angle. A single augmented controller was successful in leaning and adapting to the two very different flight conditions.

CHAPTER 3

METHODOLOGY

3.1 METHODOLOGY

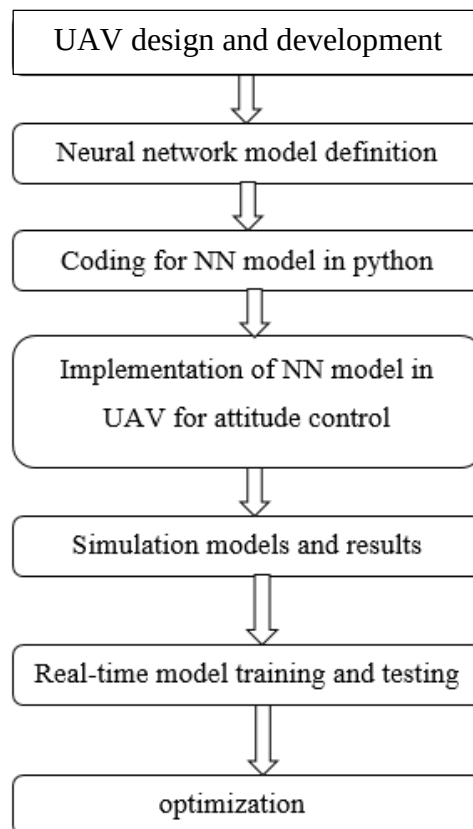


Figure 3.1 Methodology of the project

The methodology of the project is explained in the above flow chart shown in Figure 3.1, where initially a design of the UAV is done using CAD software, neural network model for multiple layers is designed and coded using python.

The NN model is trained using the gathered sensor data from simulation environment and then it is tested in simulation. After optimization of the code as the model is implemented in the real-time UAV model for training and testing. Optimization will be carried out after testing of the model for tuning its performance.

The neural network structure (size of input units, hidden units, output unit). And the below process will be implemented in UAV. The process in which the NN model will process the input data is given below.

Initialize the model's parameters. (weights, bias)

Loop:

- (i). Implement forward propagation
- (ii). Compute loss
- (iii). Implement backward propagation to get the gradients
- (iv). Update parameters (gradient descent)

CHAPTER 4

UAV DESIGN CONSIDERATIONS

4.1 AIRFOIL SELECTION

Airfoils are lifting surfaces which is used in aircrafts, it is similarly to Airfoils that are used in aircrafts. Airfoils also experiences a drag force D and the total force F is a vector quantity with components (L, D) . The forces on Airfoil is shown in Figure 4.1. The angle of attack α is the angle between the fluid velocity vector and the chord line and is used to determine values of lift and drag. As flow separates around an Airfoil, lift decreases and drag increases. Drag also increases when air flows around the tips of a wing from the high to low pressure. This air flow creates a circulatory motion that occurs downstream of the wings. This circulatory motion, also called a vortex, creates a velocity component in the downward direction at the wing, creating a downwash.

$$\text{Coefficient of Lift, } C_L = \frac{2L}{\rho v^2 S} \quad (4.1)$$

$$\text{Coefficient of Drag, } C_D = \frac{2D}{\rho v^2 S} \quad (4.2)$$

The viscous effect to be confined, the Reynolds number must be reasonably large. The body must be sufficiently thin and streamlined that separation does not occur. The additional assumption the flow leaves the trailing edge in a smooth tangential manner is fundamental to the theory of lifting surface. Wing equations for lift and drag forces on a vehicle use coefficients of lift and drag for finite wings, which are 3-dimensional wings stretching in the z -direction a distance known as the wing span. Finite wing coefficients are displayed as C_L and C_D . Equations to calculate the finite wing coefficients use infinite wing

coefficients. Infinite wings stretch to infinity in the z -direction and are two-dimensional, allowing flow only in the x - and y -directions. Infinite wing coefficients are displayed as C_l and C_d .

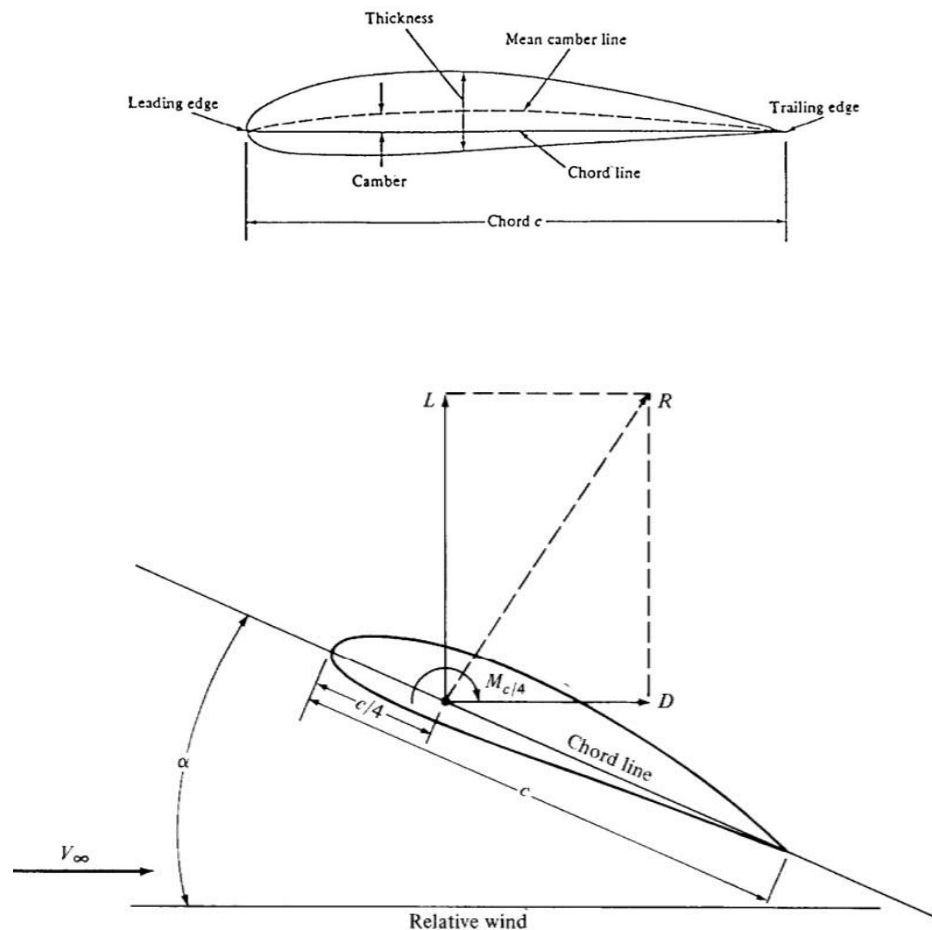


Figure 4.1 Forces on Airfoil.

Here, in this case the UAV model is a flying wing type. Which does not have a tail, which requires a special airfoil for compensating the moment created by the wing. A normal airplane with tail has a moment created by the wing in its aerodynamic centre. For compensating the moment produced by the wing, a tail is introduced in the airplane. In this case, the UAV design doesn't have a tail for compensating the moment produced by the wing. For compensating the moment produced by the wing a special airfoil which has a trailing edge cusp is used. The

airfoil chose is a MH-60 airfoil shown in Figure 4.2 which has a special trailing edge cusp for compensating the moment produced by the wing in a flying wing models. The co-ordinates of this airfoil are given in Appendix III.

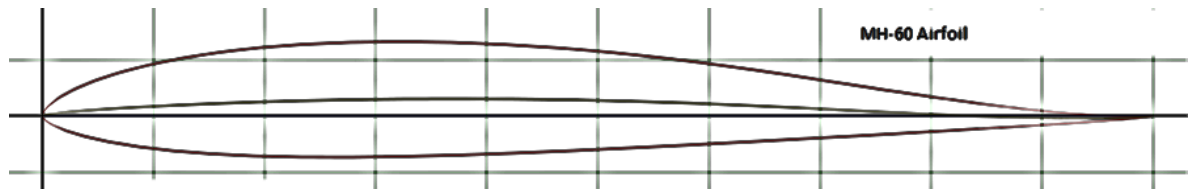


Figure 4.2 Airfoil profile of MH-60 airfoil.

4.2 DESIGN SPECIFICATION OF UAV

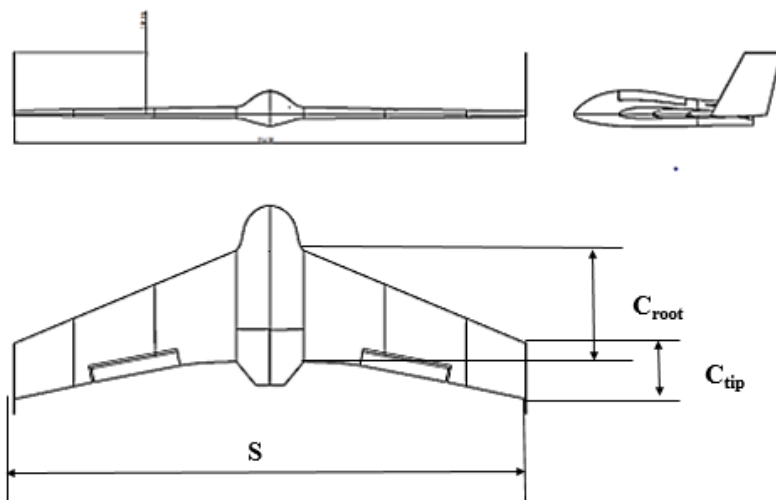


Figure 4.3 UAV design specifications.

Where,

S	920 mm
C_{root}	198 mm
C_{tip}	99 mm

The wingspan (**S**) an airplane is the distance from one wingtip to the other wingtip. here wing span is 920mm. Some wings have different chord lengths near the fuselage and at the tip. The chord length near the fuselage is the **root chord (C_{root})**, and the chord length at the end of the wing the **tip chord (C_{tip})** as shown in Figure 4.3. The maximum width of the wing at the root airfoil is 198mm. Tip chord the minimum width of the wing at tip airfoil is tip chord and here the tip chord is 99mm.

A swept wing is a wing that angles either backward or, occasionally, forward, from its root rather than in a straight sideways direction. Wing sweep has the effect of delaying the shock waves and accompanying aerodynamic drag rise caused by fluid compressibility near the speed of sound, improving performance. Here Sweep distance from the tip of the aircraft is 200mm.

4.3 CENTRE OF GRAVITY

An airplane in flight can be maneuvered by the pilot using the aerodynamic control surfaces; the elevator, rudder, or ailerons. As the control surfaces change the amount of force that each surface generates, the aircraft rotates about a point called the center of gravity. The center of gravity is the average location of the weight of the aircraft. The weight is distributed throughout the airplane, and for some problems it is important to know the distribution. But for total aircraft maneuvering, we need to be concerned with only the total weight and the location of the center of gravity.

To determine the center of gravity cg , we choose a reference location, or reference line. The cg is determined relative to this reference location. The total weight of the aircraft is simply the sum of all the individual weights of the components. Since the center of gravity is an average location of the weight, we can say that the weight of the entire aircraft W times the location cg of the center of gravity is equal to the sum of the weight w of each component times the distance d of that component from the reference location:

$$W * cg = [w * d] (\text{fuselage}) + [w * d] (\text{wing}) + [w * d] (\text{engines}) \quad (3.3)$$

Hence by the above equation the center of gravity (CG) is found to be in the location of 106.24mm from the nose tip of the UAV.

4.4 AERODYNAMIC CENTRE

The aerodynamic centre is the location on the airfoil where Moment constant for all changes of AOA. Simple airfoil theory shows the Aerodynamic center is at $c/4$ (“quarter chord”) for low subsonic symmetric airfoils. At approximately the $c/4$ for low subsonic non-symmetric airfoils. At $c/2$ (“mid-chord”) for supersonic airfoils shown in Figure 4.4.

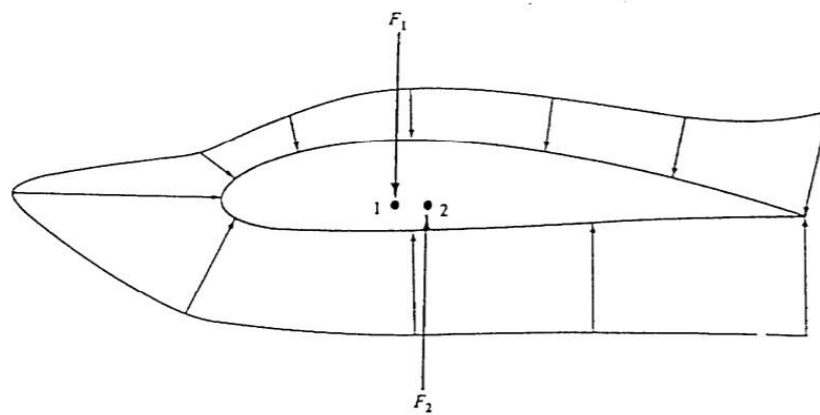


Figure 4.4 Aerodynamic centre.

Here the root chord of the wing is 198 mm, therefore the aerodynamic centre is 49.5mm from the tip of the airplane.

4.5 CAD MODEL DESIGN

The CAD model has been designed using solid works for the above specified design parameters. Each part has been designed separately and assembled together into an assembly using various mating options in Solidworks. This model has been designed and rendered in Solidworks. The three two dimensional views (Top, Front and Side view) and isometric view of UAV model is shown in the Figure 4.5. The rendered view of the UAV model is shown in the Figure 4.6 as follows. The dimensions are in mm and the designed model is a flying wing type UAV model which has no tail.

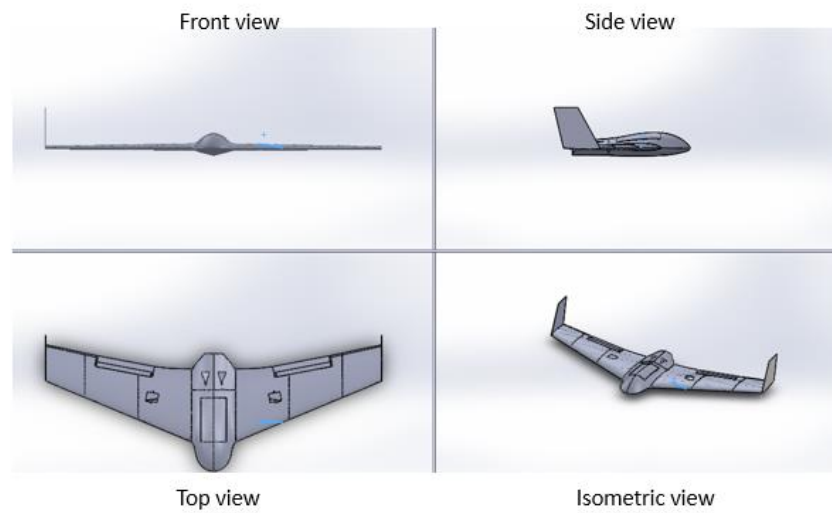


Figure 4.5 CAD model designed using Solid Works.



Figure 4.6 Rendered CAD model of fixed wing UAV.

CHAPTER 5

NEURAL NETWORK MODEL

5.1 INTRODUCTION

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of many highly-interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons.

5.1.1 ADVANTAGES OF NEURAL NETWORK

- (i). Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
- (ii). Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
- (iii). Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- (iv). Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

5.2 CONTROL SYSTEM DESIGN AND MODEL DEFINITION

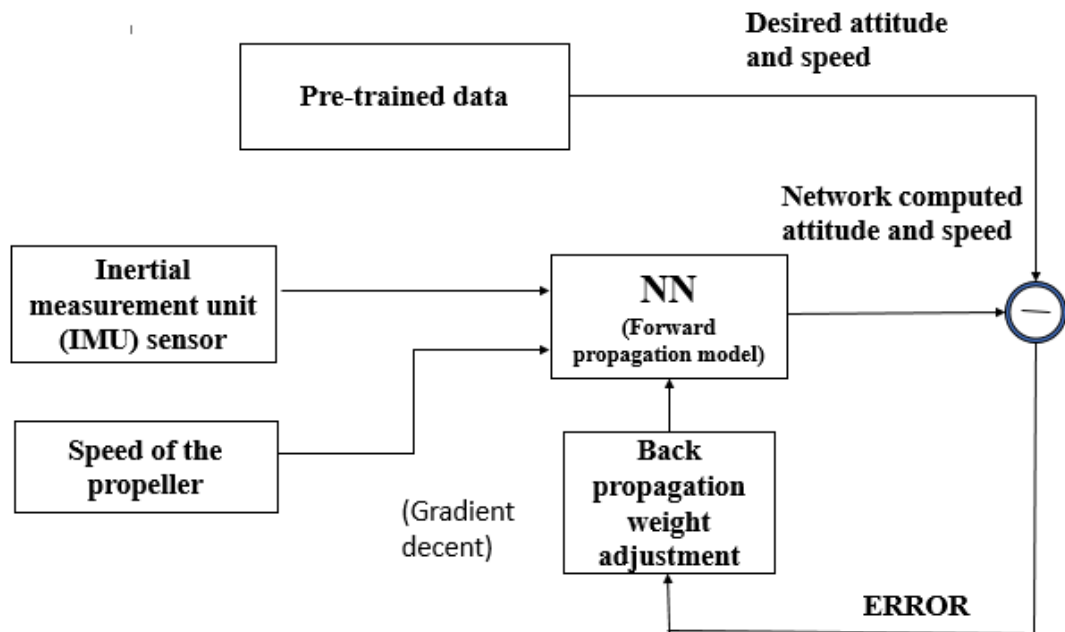


Figure 5.1 Control system design.

The control system design of the UAV is designed with a neural network control shown in Figure 5.1. Where the input for the neural network model is roll, pitch and yaw rate from inertial measurement unit (IMU) sensor and speed of the propeller from the Electronic speed controller. The neural network will be trained using the manual controlled flight, where IMU sensor data and propeller speed data will be logged. The sensor log will be given to the neural network model as a training data and trained until it gets an optimized result. After getting the trained set, the model will undergo real-time testing where the real-time data of IMU sensor and propeller speed will be given as input to the Neural Network model. the multi layered model will compare the input sensor data with the pre-trained and the difference in the values will be taken and the weight will be adjusted per the change through Back-propagation weight adjustment. And finally, the optimized output of a signal for servo will be given to maintain the attitude of the UAV.

5.2.1 Inputs and Outputs definition

The inputs of the neural network are fed from the IMU sensors and a feedback of thrust, roll, pitch and yaw from the back propagation will be given back to the Neural Network model. The input and output of the neural network model is identified using pitch and roll identifier, which defines the inputs and output of the model.

Pitch identifier:

$$\varphi(i + 1) = \varphi_p \{u(i), v(i), \varphi(i), \theta(i), p(i), q(i), r(i), \delta a(i), \delta \theta(i), \delta r(i)\} \quad (5.1)$$

Roll identifier:

$$\theta(i + 1) = \theta_p \{u(i), w(i), \varphi(i), \theta(i), p(i), q(i), r(i), \delta a(i), \delta e(i), \delta \theta(i)\} \quad (5.2)$$

- (i). φ - required angle of pitch to stabilize the UAV.
- (ii). θ – required angle of roll to stabilize the UAV.
- (iii). $p(i), q(i), r(i)$ - angular rates for roll, pitch and yaw respectively,
- (iv). $u(i), v(i)$ and $w(i)$ are the velocity components along longitudinal, lateral and vertical axis.
- (v). $\delta a, \delta e, \delta \theta$ and δr corresponds to servo outputs for aileron, elevator, throttle and rudder.

5.2.2 Neural Network model

For the attitude control of the UAV a neural network model with single hidden layer and multiple hidden layer will be tested for optimization shown in Figure 5.2. The process involves two phases, which has a training of the model and testing of the model. The training will include a training of this model using the manual data of sensors collected from the initial manual flight of UAV.

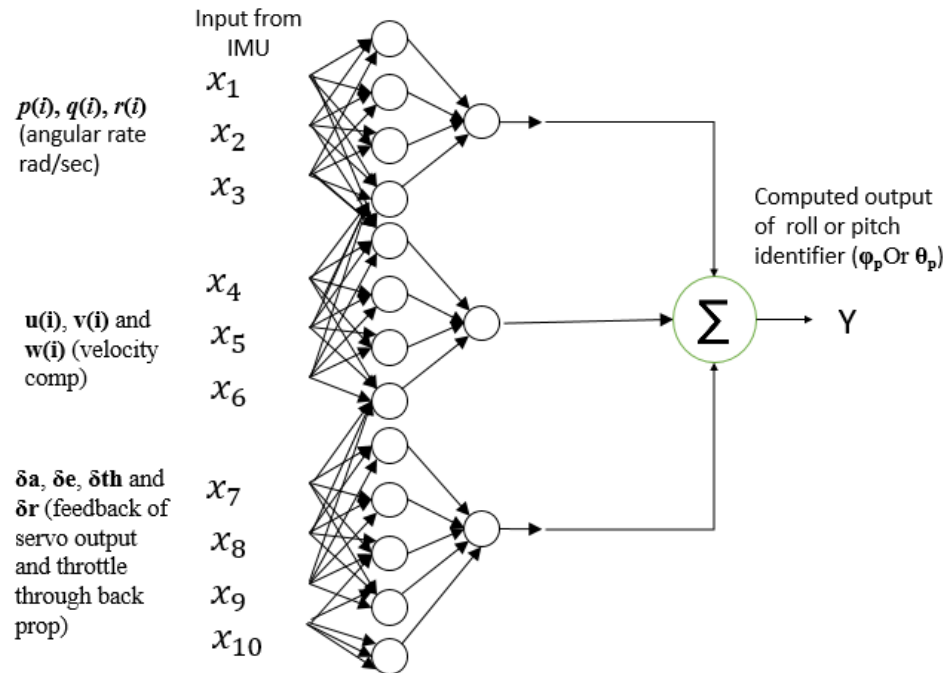


Figure 5.2 Neural Network model with single hidden layer.

The neural network model will have a forward propagation model in each neuron with an activation function for getting the output and a back-propagation model will get the derivative of output from each neuron. Then the change in weights and bias will be calculated by differencing the old weights with change in weights to get new weights. Each of this weights and bias will be updated for each iteration. By calculating the loss using cost function J will identify the change in accuracy of the output of the model which can be used for optimization during testing.

The neural network model will have a single hidden layer shown in Figure 5.2 and three hidden layers shown in Figure 5.3, which will be tested for its accuracy. The optimized model will be further implemented in the real time testing of the model.

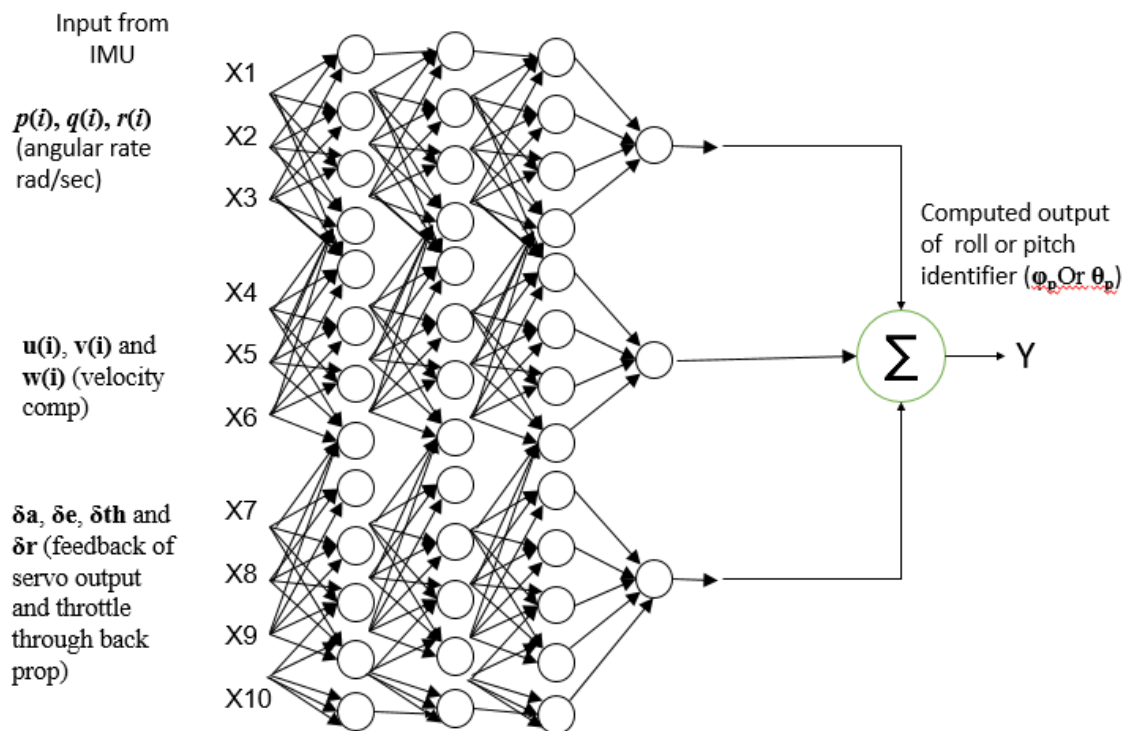


Figure 5.3 Neural Network model with three hidden layers.

5.2.3 Neural Network model parameters

The neural model has a feed-forward propagation for computation of output through an activation function. And a back-propagation neural network will compute the change and the data can be used for updating the weights and bias for each neuron in every iteration through a back-propagation weight adjustment. Defining the neural network structure (size of input units, hidden units, output unit) for a basic neural network. the following steps will be carried for every iteration and the weights and bias will be updated. Then by using the computed output neural network model and the desired output from the pre-trained data. The difference in output will be calculated using a back-propagation model and fed back to the neural network.

- (i). Initialize the model's parameters. (weights, bias)
- (ii). Loop: (for each iterations)
 - a. Implement forward propagation

- b. Compute loss
- c. Implement backward propagation to get the gradients
- d. Update parameters (gradient descent)

5.2.4 Forward Propagation model

The forward propagation model for the single and multiple hidden layer neural network and the cost function for computing the loss is defined as follows.

$$z^{[1]} = W^{[1]}X + b^{[1]} \quad (5.3)$$

$$a^{[1]} = \sigma(z^{[1]}) \quad (5.4)$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \quad (5.5)$$

$$a^{[2]} = \sigma(z^{[2]}) \quad (5.6)$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = - \frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \quad (5.7)$$

Where,

X -- input dataset of shape (input size, number of examples)

y -- labels of shape (output size, number of examples)

nx -- size of the input layer

nh -- size of the hidden layer

ny -- size of the output layer

$W^{[1]}$ -- weight matrix of shape (nh, nx)

$b^{[1]}$ -- bias vector of shape (nh, 1)

$W^{[2]}$ -- weight matrix of shape (ny, nh)

$b^{[2]}$ -- bias vector of shape (ny, 1)

$Z^{[1]}, dZ^{[1]}$ -- computed output Y matrix and its derivative of shape(nh,1)

$Z^{[2]}$, $dZ^{[2]}$ -computed output Y matrix and its derivative of shape $(n \times 1)$

The equation (5.3) defines the forward propagation for first layer where the activation function used is ReLU (Rectified Linear Unit) (equation 5.4) which outputs the maximum of 'z' between 0 and 1. The equation (5.5) and (5.6) defines the forward propagation and activation function for next layers, this will continue for number of hidden layers used in this model.

5.2.5 Back Propagation model

The back-propagation model for the single and multiple hidden layer neural network and the derivative of the cost function for computing the loss is defined as follows.

$$dW^{[2]} = dz^{[2]}a^{[1]T} \quad (5.8)$$

$$dz^{[2]} = a^{[2]} - y \quad (5.9)$$

$$db^{[2]} = dz^{[2]} \quad (5.10)$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) \quad (5.11)$$

$$dW^{[1]} = dz^{[1]}x^T \quad (5.12)$$

$$db^{[1]} = dz^{[1]} \quad (5.12)$$

The equation (5.8) defines the back-propagation for second layer and the change in weights and bias is defined (equation 5.4) which outputs the maximum of 'z' between 0 and 1. The equation (5.5) and (5.6) defines the forward propagation and activation function for next layers.

Thus, the neural network model has been defined. This algorithm will be coded using python language for single and multiple hidden layer is shown in appendix I and II. This model will be trained using a pre-trained data which has been collected using the manual control of UAV in simulation environment and in real-time model.

CHAPTER 6

IMPLEMENTATION OF NEURAL NETWORK MODEL

6.1 OVERALL MODEL

Implementation of the neural network model with a single and multiple hidden layer for the UAV model is explained in this chapter. The first component, is the data acquisition from sensors in the UAV (through Raspberry Pi) shown in Figure 6.1. The second part, is a program that use a python application programming interface (API) for neural network model of single and multiple hidden layer, the application block in this part, have a direct connection with sensors, over a low-latency link of I²C communication interface to communicate and store telemetry data in a buffer.

6.2 MODEL ARCHITECTURE

A significant factor is the input of this component, a decision block that receives telemetry data from the buffer, and compare a threshold value to select how the system behaves. The system can operate in: (i) learning mode, or in (ii) optimization mode shown in Figure 6.2. In this way, the decision block analyses the path measurement error considering the two-dimensional paths. This measure of threshold is called XTrack error.

When the system, run over a low XTrack error, the decision block activates the learning mode. The learning mode, operates using a Neural Network model, receiving data from telemetry and using a hybrid learning algorithm to estimate the attitude dynamics. The optimization block is activated, if the XTrack

error exceeds a threshold. To this block to work properly, the Neural Network system must be able to estimate the attitude with low prediction error.

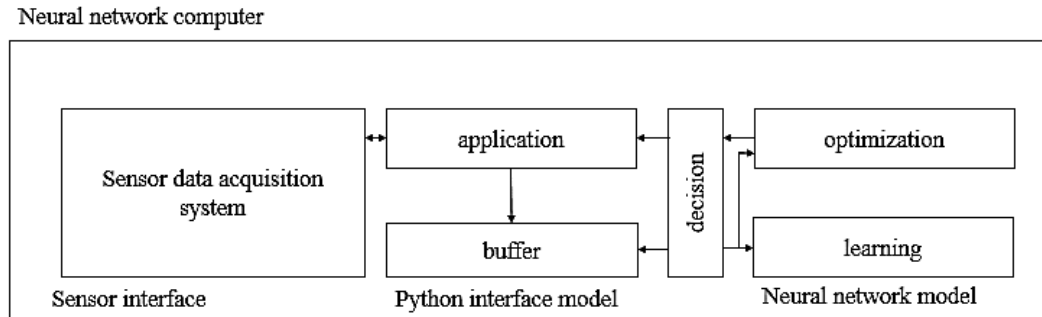


Figure 6.1 Overall representation of system proposed.

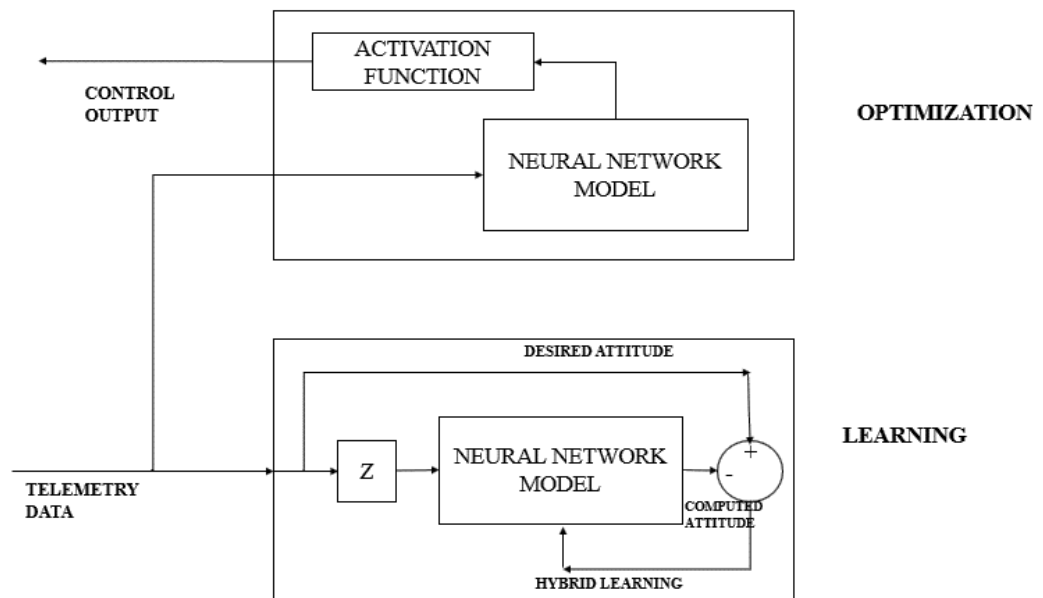


Figure 6.2 Neural Network Architecture.

CHAPTER 7

SOFTWARE IN-THE-LOOP SIMULATION

7.1 SITL ENVIRONMENT

The simulation environment can run the algorithm without any specific autopilot hardware. The software in the loop (SITL) simulation is a part of the autopilot code which gives a native executable to test the behaviour of the flight and the proposed solution. The advantages of this methodology are the early detection of bugs, design flaws, and the optimization of control software parameters. Thus, more mature designs are transferred to later design and test stages.

7.2 SOFTWARE ARCHITECTURE AND CONNECTION

The ArduPlane is the main software, the software compiled and build is the same one that goes embedded in the UAV. The MAVProxy is a portable console based app, that allows to simulate the radio control transmitter (RC Tx), supports the direct communication with the autopilot software using MAVLink, and can relay the real-time telemetry from the ArduPlane to the ground control station (GCS), like a proxy.

The physics simulation software JSBSim is an open source flight dynamics model (FDM), fully configurable. The FDM is essentially the math model that defines the movement of an aircraft under the forces and moments applied to it using the various control mechanisms and from the forces of nature. As JSBSim does not have native graphical support, here FlightGear is used as a graphical viewer, a sophisticated and open flight simulator framework for use in

research or academic environments, pilot training, and as an industry engineering tool, besides being a realistic desktop flight simulator. A Software In The Loop simulation model is defined in the Figure 7.1.

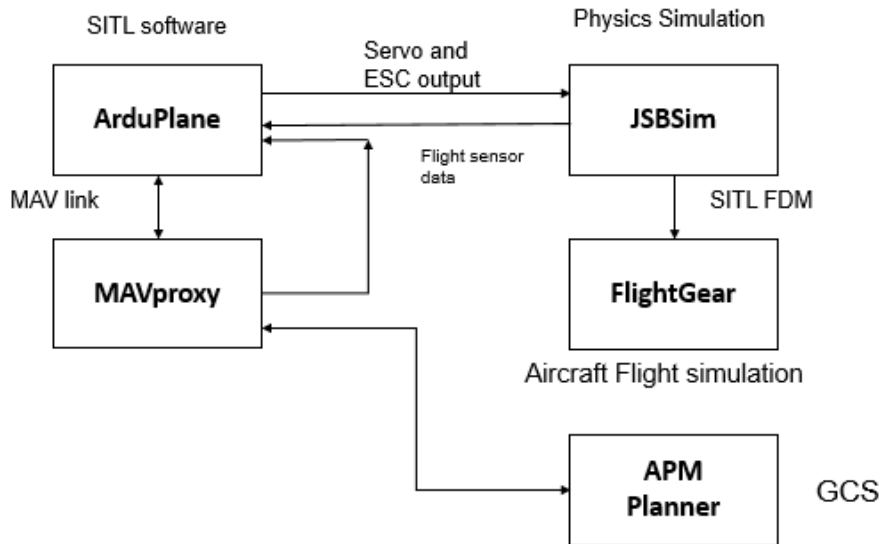


Figure 7.1 Software In-The-Loop Simulation Model.

7.3 ARDUPILOT SITL SOFTWARE

SITL can run ArduPilot on PC directly, without any special hardware. It takes advantage of the fact that ArduPilot is a portable autopilot that can run on a very wide variety of platforms. PC is just another platform that ArduPilot can be built and run on.

When running in SITL the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of vehicle simulators built in, and can interface to several external simulators. by using the arduplane option the simulation environment for Software in-the-loop simulation will be started using a command. The JSBsim physics simulation software will simulate all the required physics environment required for an UAV flight.

The MAVproxy is a ground control station software which can be used for simulation of ground control and mapping the sensor data and Waypoint data

of the UAV model. The below Figure 7.2 and Figure 7.3 shows the map and path of the UAV and sensor data respectively.



Figure 7.2 Map of the waypoints loaded for UAV navigation in SITL environment.

```
Loaded defaults from /home/arvinhd/ardupilot/Tools/autotest/default_params/plane
-jsbsim.parm
77: GEAR_CONTACT: RIGHT_MLG 0
78: GEAR_CONTACT: LEFT_MLG 0
79: GEAR_CONTACT: TAIL_LG 0

Takeoff report for LEFT_MLG (Liftoff at time: 139.038000 seconds)
Distance traveled:      22.955491 ft.      6.996834 meters
Distance traveled (over 50'):  199.578735 ft.  60.831598 meters
[Altitude (ASL): 51.137568 ft. / 15.586731 m | Temperature: 58.817635 F / 14.
898686 C]
[Velocity (KCAS): 33.005789]

Takeoff report for RIGHT_MLG (Liftoff at time: 139.042000 seconds)
Distance traveled:      16.477920 ft.      5.022470 meters
Distance traveled (over 50'):  199.682294 ft.  60.863163 meters
[Altitude (ASL): 51.178976 ft. / 15.599352 m | Temperature: 58.817488 F / 14.
898604 C]
[Velocity (KCAS): 33.008491]

Takeoff report for TAIL_LG (Liftoff at time: 139.086000 seconds)
Distance traveled:      23.039885 ft.      7.022557 meters
Distance traveled (over 50'):  200.821858 ft.  61.210502 meters
[Altitude (ASL): 51.635060 ft. / 15.738366 m | Temperature: 58.815861 F / 14.
897701 C]
[Velocity (KCAS): 33.038028]
```

Figure 7.3 Output of various sensor data and speed of UAV from simulation environment.

```

AUTO  ARM  GPS: OK6 (10)  Vcc 5.00  Radio: --  INS  MAG  AS  RNG  AHRS  EKF  LOG  FEN  RC  TERR  PWR:  Srv 0.00
Batt: 86%/12.44V 10.5A  Link 1 OK 100.0% (18660 pkts, 0 lost, 0.00s delay)
Hdg 355/354  Alt 99m  AGL 104m/103m  AirSpeed 21m/s  GPSSpeed 22m/s  Thr 21  Roll 0  Pitch 0  Wind -11/0.22
WP 5/8  Distance 295m  Bearing -5  AltError 0L  AspdError 0.3L  FlightTime 1:33  ETR 1:05
Sent waypoint 0: MISSION_ITEM {target_system: 1, target_component: 0, seq: 0, frame: 0, command: 16, current: 1, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.362938, y: 149.165085, z: 584.409973}
Sent waypoint 1: MISSION_ITEM {target_system: 1, target_component: 0, seq: 1, frame: 3, command: 22, current: 0, autocontinue: 1, param1: 15.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.361164, y: 149.163986, z: 28.110001}
Sent waypoint 2: MISSION_ITEM {target_system: 1, target_component: 0, seq: 2, frame: 3, command: 16, current: 0, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.359467, y: 149.161697, z: 99.800003}
Sent waypoint 3: MISSION_ITEM {target_system: 1, target_component: 0, seq: 3, frame: 3, command: 16, current: 0, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.366333, y: 149.162659, z: 100.730003}
Sent waypoint 4: MISSION_ITEM {target_system: 1, target_component: 0, seq: 4, frame: 3, command: 16, current: 0, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.366131, y: 149.164581, z: 100.0}
Sent waypoint 5: MISSION_ITEM {target_system: 1, target_component: 0, seq: 5, frame: 3, command: 16, current: 0, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.359272, y: 149.163757, z: 100.0}
Sent waypoint 6: MISSION_ITEM {target_system: 1, target_component: 0, seq: 6, frame: 3, command: 177, current: 0, autocontinue: 1, param1: 2.0, param2: -1.0, param3: 0.0, param4: 0.0, x:
0.0, y: 0.0, z: 0.0}
Sent waypoint 7: MISSION_ITEM {target_system: 1, target_component: 0, seq: 7, frame: 3, command: 16, current: 0, autocontinue: 1, param1: 0.0, param2: 0.0, param3: 0.0, param4: 0.0, x:
-35.359272, y: 149.163757, z: 100.0}
Sent all 8 waypoints
Got MAVLink msg: MISSION_ACK {target_system: 255, target_component: 0, type: 0}
APM: Flight plan received
height 90
height 100
height 110
APM: Reached waypoint #2 dist 57m
APM: Executing nav command ID #16
waypoint 3
600
400
200
Flight battery 90 percent
APM: Reached waypoint #3 dist 57m
APM: Executing nav command ID #16
waypoint 4
APM: Reached waypoint #4 dist 56m
APM: Executing nav command ID #16
waypoint 5
600
400
200

```

Figure 7.4 Various sensor and throttle data of UAV from MAVproxy in SITL environment.

The above Figure 7.4 shows the various sensor data and position data of the UAV model from MAVproxy GCS software in SITL environment.

CHAPTER 8

IMU SENSOR FUSION AND CALIBRATION

8.1 ABOUT IMU SENSOR MPU-9250

MPU-9250 is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8963 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Hence, the MPU-9250 is a 9-axis Motion Tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU-6515. With its dedicated I²C sensor bus, the MPU-9250 directly provides complete 9-axis MotionFusion output. The MPU-9250 Motion Tracking device, with its 9-axis integration, on-chip MotionFusion, and runtime calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices, guaranteeing optimal motion performance for consumers. MPU-9250 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I²C port.

MPU-9250 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full-scale range of ± 250 , ± 500 , ± 1000 , and

$\pm 2000^\circ/\text{sec}$ (dps), a user-programmable accelerometer full-scale range of $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$, and a magnetometer full-scale range of $\pm 4800\mu T$.

8.2 MPU-9250 PIN OUT DIAGRAM

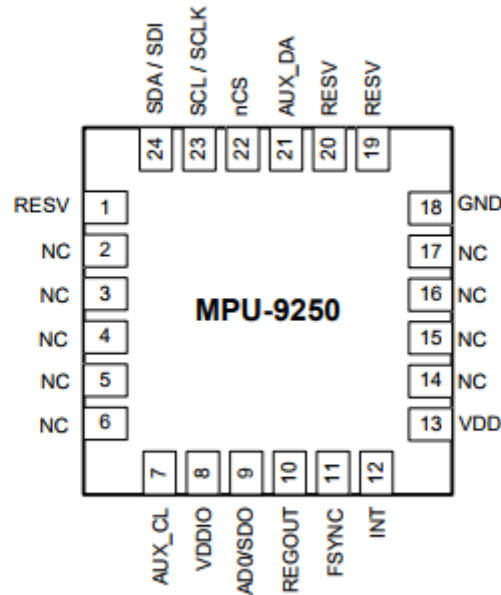


Figure 8.1 Pin Out diagram of MPU-9250 IMU sensor

The above Figure 8.1 explains the pinout diagram of MPU-9250 IMU sensor used for UAV attitude sensing.

8.3 CONNECTING IMU SENSOR WITH RASPBERRY PI

The IMU sensor relates to raspberry pi model 3 using I²C communication. The I²C communication channel in the IMU sensor can be accessed through the following procedure. The system processor is an I²C master to the MPU-9250. In addition, the MPU-9250 is an I²C master to the optional external 3rd party sensor. The MPU-9250 has limited capabilities as an I²C Master, and depends on the system processor to manage the initial configuration of any auxiliary sensors. The MPU- 9250 has an interface bypass multiplexer, which connects the system processor I²C bus (SDA and SCL) directly to the auxiliary sensor I²C bus (AUX_DA and AUX_CL). Once the auxiliary sensors have been configured by the system processor, the interface bypass multiplexer should be

disabled so that the MPU-9250 auxiliary I²C master can take control of the sensor I²C bus and gather data from the auxiliary sensors. The INT pin should be connected to a GPIO on the system processor that can wake the system from suspend mode shown in Figure 8.2.

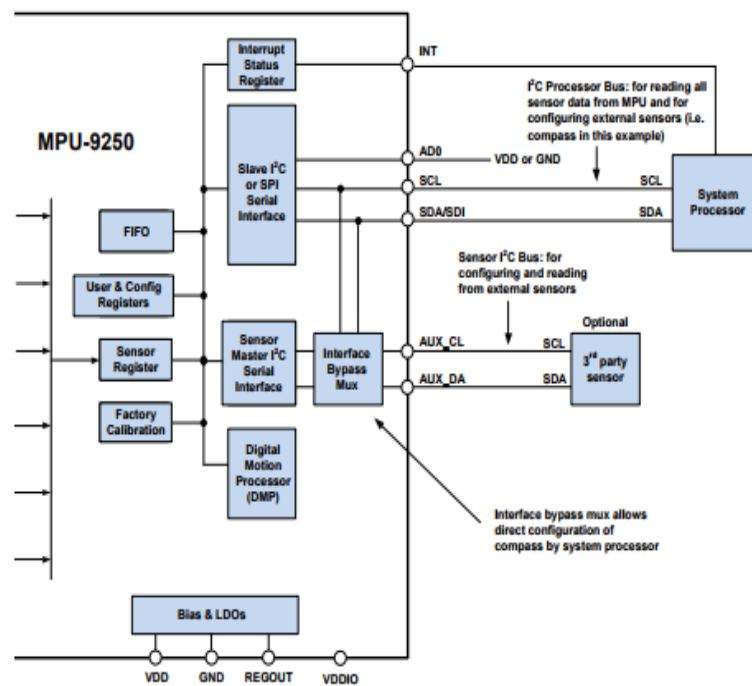


Figure 8.2 Connecting MPU-9250 IMU sensor through I²C communication.

MPU 9250	R pin
Vcc	1
SDA	3
SCL	5
GND	6
INT	11

Table 8.1 Connection pin of IMU and Raspberry pi GPIO pins.



Figure 8.3 Connection pins of the MPU-9250 and GPIO pins of Raspberry pi.

The connection pins of raspberry pi and IMU sensor is defined in the table 8.1 and shown in Figure 8.3.

CHAPTER 9

RESULTS AND DISCUSSIONS

9.1 SIMULATION RESULTS

In order to test, the simulation environment will be configured to preliminary experimental tests. First, a flight mission with 7 points will be defined, using the control ground station (Appendix V). By adjusting this point, regardless of a system failure such as low battery, the UAV will activate the “fail safe” mode and will attempt to return to this point. The “take off” point is the point that defines the route from the current position of the aircraft (Home point) to the defined point. Also, a desired altitude and a maximum pitch angle is set for an abrupt or softer take-off. The Waypoints has been loaded through MAVProxy command prompt and data has been logged in a log file for inputting these data into Neural Network model for training.

The sensor data acquired from the manual simulation of the model has been plotted as graph in Figure 9.1. the roll and pitch output of the simulation model has been plotted in the Figure 9.2 as follows. This will be fed to the Neural Network model for training the model in simulation environment. After training the computed output from the model is stored in the log file and compared with the desired output which is shown in the Figure 9.3. The graph in Figure 9.3 compares the desired roll and pitch data obtained through manual simulation of UAV model with the computed data from the autonomous model. The simulation output is validated with manual data for getting a desired performance from the proposed model.

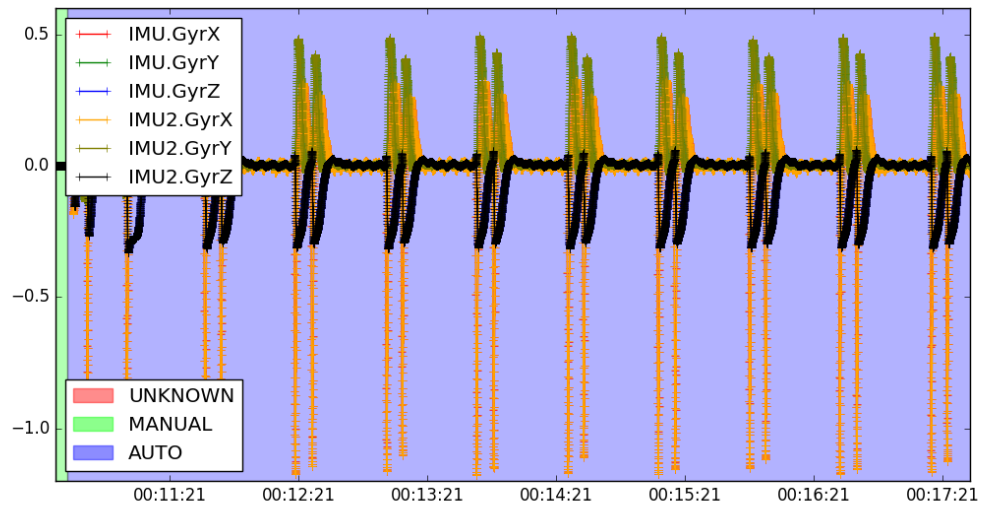


Figure 9.1 three-axis Gyro sensor data obtained from simulation environment.

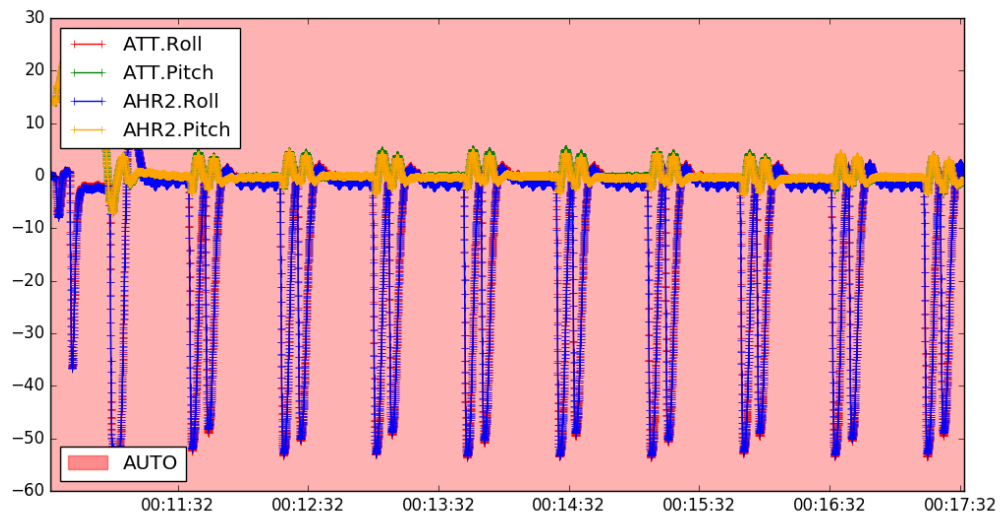


Figure 9.2 Roll and Pitch data obtained from simulation of the UAV model in SITL simulation.

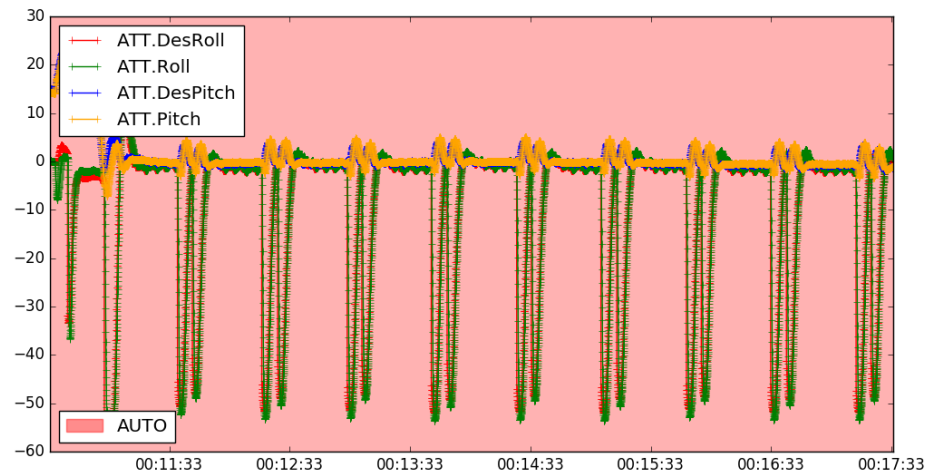


Figure 9.3 Desired roll and pitch data compared with acquired roll and pitch data.

9.2 SENSOR DATA AQUISITION

9.2.1 Acquiring data from MPU-9250 IMU sensor

The IMU sensor has been connected to raspberry pi using I²C communication. The sensor data can be acquired to the raspberry pi has been done using python RTIMULIB (a library database for calibrating and acquiring data from IMU sensor)

- RTIMULibCal will get the data from sensor for calibration.
- RTIMULibDrive will get the serial output of roll, pitch, and yaw
- RTIMULibDemo will get sensor data from gyro, magnetometer and accelerometer

9.2.2 Calibrating the raw data

The data that is received then must be calibrated per the user's environment. The calibration of the magnetometer is required to compensate for Magnetic Declination. The exact value of the correction depends on the location. There are two variables that must calibrated: yaw and magbias.

This is one of the simplest and important part of the MPU code. This function calibrates the magnetometer while you move the sensor in a figure eight. It stores the maximum and minimum readings and takes the average. The calibration code for calibrating the MPU-9250 is defined in Appendix IV.

9.2.3 Calibrated sensor data after acquisition

The calibrated sensor data can be displayed in the GUI interface as shown in Figure 9.4 and the required roll pitch and Yaw rate of the IMU sensor can be obtained from the python API using the command 'RTIMULibDrive'. This command will output the Roll, Pitch and Yaw rate of the sensor serially. Which can be given as an input to the Neural Network model for optimization and learning process.

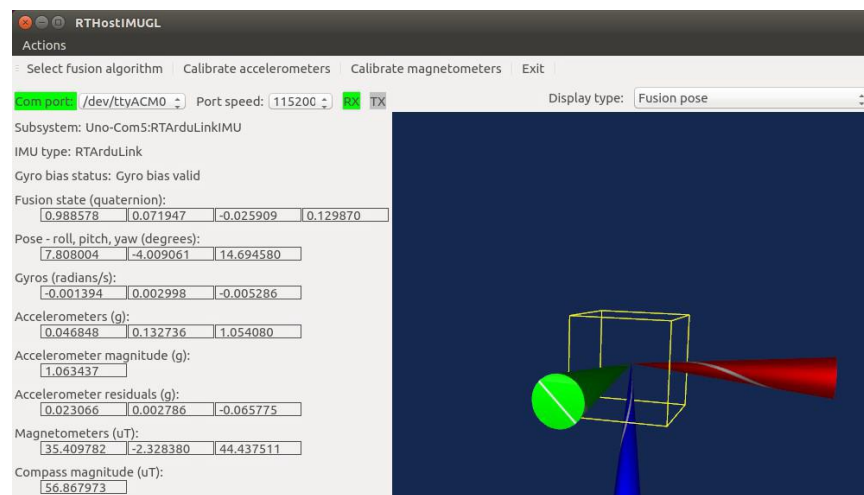


Figure 9.4 Calibration and data acquisition of MPU-9250 IMU sensor.

CHAPTER 10

CONCLUSION

10.1 CONCLUSION

This project proposes a new adaptive algorithm to ensure an optimal adjustment in attitude controller throughout the mission. The use of artificial intelligence techniques, such as Artificial Neural Network (ANN), proved to be very efficient in searching for optimized parameters, even without the supervision of a human. This automates the task of the pilot of each mission having to perform a routine of gain adjustments in a pre-flight. As future work, the aim is to use the assembled prototype to test the proposed approach, in order to test the effectiveness of the approach in a real environment.

10.2 WORK SCHEDULE FOR PHASE II

The work schedule for the project phase II is as follows,

- (i). Hardware in-the-loop simulation of the sensor with the raspberry pi model 3 hardware using python.
- (ii). Fabrication of UAV model and fixing the electronics in the model.
- (iii). Manual test flight for collecting the sensor data for training the neural network model.
- (iv). Training and optimization of Neural Network model code.
- (v). Real-time Testing of the UAV model in Autonomous mode with the help of trained neural network model.

APPENDIX I

Python coding for single hidden layer neural network is given as follows

```
import numpy as np
import matplotlib.pyplot as plt
from testCases_v2 import *
import sklearn
import sklearn.datasets
import sklearn.linear_model
from planar_utils import plot_decision_boundary, sigmoid,
load_planar_dataset, load_extra_datasets
%matplotlib inline
np.random.seed(1)

def layer_sizes(X, Y):

    n_x = X.shape[0] # size of input layer
    n_h = None        # size of hidden layer
    n_y = Y.shape[0] # size of output layer
    return (n_x, n_h, n_y)

def initialize_parameters(n_x, n_h, n_y):
    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros((n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros((n_y, 1))
    assert (W1.shape == (n_h, n_x))
    assert (b1.shape == (n_h, 1))
    assert (W2.shape == (n_y, n_h))
    assert (b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters

def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    Z1 = np.dot(W1, X) + b1
    A1 = relu(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = relu(Z2)

    assert (A2.shape == (1, X.shape[1]))

    cache = {"Z1": Z1,
             "A1": A1,
```

```

        "Z2": Z2,
        "A2": A2}

    return A2, cache

def compute_cost(A2, Y, parameters):
    m = Y.shape[1] # number of example
    logprobs = -(np.multiply(np.log(A2),Y) + np.multiply((np.log(1-
A2)), (1-Y)))/m
    cost = np.sum(logprobs)
    cost = np.squeeze(cost)
    assert(isinstance(cost, float))

    return cost

def backward_propagation(parameters, cache, X, Y):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    A1 = cache["A1"]
    A2 = cache["A2"]
    dZ2 = A2-Y
    dW2 = np.dot(dZ2,A1.T)/m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)/m
    dZ1 = np.dot(W2.T,dZ2)*(1-np.power(A1,2))
    dW1 = np.dot(dZ1,X.T)/m
    db1 = np.sum(dZ1, axis = 1, keepdims = True)/m

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads

def update_parameters(parameters, grads, learning_rate):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1-np.dot(learning_rate,dW1)
    b1 = b1-np.dot(learning_rate,db1)
    W2 = W2-np.dot(learning_rate,dW2)
    b2 = b2-np.dot(learning_rate,db2)

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters

def nn_model(X, Y, n_h, num_iterations = 10000, print_cost=False):

```

```

np.random.seed(3)
n_x = layer_sizes(X, Y)[0]
n_y = layer_sizes(X, Y)[2]

parameters = initialize_parameters(n_x, n_h, n_y)
W1 = parameters["W1"]
b1 = parameters["b1"]
W2 = parameters["W2"]
b2 = parameters["b2"]

for i in range(0, num_iterations):

    A2, cache = forward_propagation(X, parameters)

    cost = compute_cost(A2, Y, parameters)

    grads = backward_propagation(parameters, cache, X, Y)

    parameters = update_parameters(parameters, grads,
learning_rate = None)

    if print_cost and i % 1000 == 0:
        print ("Cost after iteration %i: %f" %(i, cost))

    return parameters
def predict(parameters, X):

    A2, cache = forward_propagation(X, parameters)
    predictions = A2>0.5
    return predictions

hidden_layer_sizes = [1, 2, 3, 4, 5, 20, 50]
for i, n_h in enumerate(hidden_layer_sizes):
    plt.subplot(5, 2, i+1)
    plt.title('Hidden Layer of size %d' % n_h)
    parameters = nn_model(X, Y, n_h, num_iterations = 5000)
    plot_decision_boundary(lambda x: predict(parameters, x.T), X,
Y)

    predictions = predict(parameters, X)
    accuracy = float((np.dot(Y,predictions.T) + np.dot(1-Y,1-
predictions.T))/float(Y.size)*100)
    print ("Accuracy for {} hidden units: {}".format(n_h,
accuracy))

```

python coding for activation functions used in the neural network coding:

```

import numpy as np
def sigmoid(Z):
    """
    Implements the sigmoid activation in numpy

    Arguments:
    Z -- numpy array of any shape

```

```

Returns:
A -- output of sigmoid(z), same shape as Z
cache -- returns Z as well, useful during backpropagation
"""

A = 1/(1+np.exp(-Z))
cache = Z

return A, cache

def relu(Z):
    """
    Implement the RELU function.

    Arguments:
    Z -- Output of the linear layer, of any shape

    Returns:
    A -- Post-activation parameter, of the same shape as Z
    cache -- a python dictionary containing "A" ; stored for
computing the backward pass efficiently
    """

    A = np.maximum(0, Z)

    assert(A.shape == Z.shape)

    cache = Z
    return A, cache

def relu_backward(dA, cache):
    """
    Implement the backward propagation for a single RELU unit.

    Arguments:
    dA -- post-activation gradient, of any shape
    cache -- 'Z' where we store for computing backward propagation
efficiently

    Returns:
    dZ -- Gradient of the cost with respect to Z
    """

    Z = cache
    dZ = np.array(dA, copy=True) # just converting dz to a correct
object.

    # When z <= 0, you should set dz to 0 as well.
    dZ[Z <= 0] = 0

    assert (dZ.shape == Z.shape)

    return dZ

def sigmoid_backward(dA, cache):
    """

```

Implement the backward propagation for a single SIGMOID unit.

Arguments:

dA -- post-activation gradient, of any shape
cache -- 'Z' where we store for computing backward propagation
efficiently

Returns:

dZ -- Gradient of the cost with respect to Z
"""

Z = cache

s = 1/(1+np.exp(-Z))
dZ = dA * s * (1-s)

assert (dZ.shape == Z.shape)

return dZ

APPENDIX II

python coding for multi-layered neural network, here the coding is done for general multi-layer neural network. A three-hidden layer neural network can be defined by defining the number of hidden layer in the main function.

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
from testCases_v3 import *
from dnn_utils_v2 import sigmoid, sigmoid_backward, relu,
relu_backward

%load_ext autoreload
%autoreload 2
def initialize_parameters(n_x, n_h, n_y):
    np.random.seed(1)

    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros((n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros((n_y, 1))

    assert(W1.shape == (n_h, n_x))
    assert(b1.shape == (n_h, 1))
    assert(W2.shape == (n_y, n_h))
    assert(b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
def initialize_parameters_deep(layer_dims):
    np.random.seed(3)
    parameters = {}
    L = len(layer_dims)            # number of layers in the
    network

    for l in range(1, L):
        parameters['W' + str(l)] = np.random.randn(layer_dims[l],
layer_dims[l-1]) * 0.01
        parameters['b' + str(l)] = np.zeros((layer_dims[l], 1))
```

```

        assert(parameters['W' + str(l)].shape == (layer_dims[l],
layer_dims[l-1]))
        assert(parameters['b' + str(l)].shape == (layer_dims[l],
1))

    return parameters
def linear_forward(A, W, b):
    Z = np.dot(W,A)+b
    assert(Z.shape == (W.shape[0], A.shape[1]))
    cache = (A, W, b)

    return Z, cache
def linear_activation_forward(A_prev, W, b, activation):

    if activation == "sigmoid":
        Z, linear_cache = linear_forward(A_prev,W,b)
        A, activation_cache = sigmoid(Z)

    elif activation == "relu":

        Z, linear_cache = linear_forward(A_prev,W,b)
        A, activation_cache = relu(Z)
    assert (A.shape == (W.shape[0], A_prev.shape[1]))
    cache = (linear_cache, activation_cache)

    return A, cache
def update_parameters(parameters, grads, learning_rate):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1-np.dot(learning_rate,dW1)
    b1 = b1-np.dot(learning_rate,db1)
    W2 = W2-np.dot(learning_rate,dW2)
    b2 = b2-np.dot(learning_rate,db2)

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
def nn_model(X, Y, n_h, num_iterations = 10000, print_cost=False):

    np.random.seed(3)
    n_x = layer_sizes(X, Y)[0]
    n_y = layer_sizes(X, Y)[2]

    parameters = initialize_parameters(n_x, n_h, n_y)
    W1 = parameters["W1"]

```

```

b1 = parameters["b1"]
W2 = parameters["W2"]
b2 = parameters["b2"]

for i in range(0, num_iterations):

    A2, cache = forward_propagation(X, parameters)

    cost = compute_cost(A2, Y, parameters)

    grads = backward_propagation(parameters, cache, X, Y)

    parameters = update_parameters(parameters, grads,
learning_rate = None)

    if print_cost and i % 1000 == 0:
        print ("Cost after iteration %i: %f" %(i, cost))

    return parameters
def predict(parameters, X):

    A2, cache = forward_propagation(X, parameters)
    predictions = A2>0.5
    return predictions

hidden_layer_sizes = [1, 2, 3, 4, 5, 20, 50]
for i, n_h in enumerate(hidden_layer_sizes):
    plt.subplot(5, 2, i+1)
    plt.title('Hidden Layer of size %d' % n_h)
    parameters = nn_model(X, Y, n_h, num_iterations = 5000)
    plot_decision_boundary(lambda x: predict(parameters, x.T), X,
Y)

    predictions = predict(parameters, X)
    accuracy = float((np.dot(Y,predictions.T) + np.dot(1-Y,1-
predictions.T))/float(Y.size)*100)
    print ("Accuracy for {} hidden units: {}".format(n_h,
accuracy))

```

APPENDIX III

MH-60 aerofoil profile co-ordinates used in UAV wing design:

X	Y	X	Y
1	0	0.00083234	0.00331581
0.99905259	-0.00005216	0.00000109	-0.00011239
0.9961929	-0.00019524	0.0009573	-0.00301111
0.99138013	-0.00037832	0.00386886	-0.00573968
0.9845912	-0.00050399	0.00841997	-0.00856261
0.97587596	-0.00046151	0.01447741	-0.01140124
0.96533011	-0.00019403	0.02198015	-0.01419411
0.95303206	0.00031678	0.0309044	-0.01689967
0.9390489	0.00109321	0.04121715	-0.01950424
0.92345332	0.0021602	0.05287086	-0.021982
0.90632414	0.00355111	0.06583016	-0.02429933
0.88777298	0.00530589	0.08006805	-0.02643462
0.86794139	0.00742765	0.09554275	-0.02837913
0.84695326	0.00988506	0.11219985	-0.03010299
0.82491572	0.01264874	0.13001668	-0.03156802
0.80193895	0.01569112	0.1489813	-0.0327765
0.77813483	0.01897812	0.16904297	-0.03374973
0.75361697	0.02247127	0.19013197	-0.03447592
0.72849875	0.02611723	0.2122158	-0.03493487
0.70287517	0.02985863	0.23526401	-0.03515298
0.67684165	0.03365108	0.259203	-0.03516244
0.65051092	0.03743952	0.28395141	-0.03496803
0.62397802	0.04112463	0.30944621	-0.0345729
0.59725986	0.04462541	0.3356208	-0.03400185
0.57039295	0.0479499	0.36238532	-0.03327692
0.54348574	0.05107831	0.3896525	-0.03240358
0.51660873	0.05393569	0.41734679	-0.03139295
0.48977448	0.05648657	0.44538663	-0.03027027
0.46302806	0.05876042	0.47367555	-0.0290584
0.43646905	0.06075914	0.50211759	-0.02777175
0.41016957	0.06245174	0.53061805	-0.02642642
0.38418189	0.06383272	0.55908049	-0.02503829
0.35857989	0.06491471	0.58740679	-0.02362352
0.33345452	0.0656874	0.61549674	-0.0221963
0.3088699	0.06612731	0.64324961	-0.02076756
0.28488022	0.0662355	0.67056835	-0.01934497
0.26156162	0.06602104	0.69735876	-0.01793852
0.23899764	0.0654595	0.723525	-0.01655782
0.21721852	0.06451776	0.74897274	-0.01520824
0.1962466	0.06322167	0.77361347	-0.01389353
0.17615221	0.06159861	0.79736189	-0.01261951
0.15700155	0.05963455	0.8201341	-0.01139138
0.13881347	0.05731716	0.84184947	-0.01021285
0.12160544	0.05468498	0.86243222	-0.00908741
0.10543815	0.05177138	0.88180889	-0.00802058
0.09035882	0.04857406	0.89990613	-0.0070152
0.07637871	0.04510162	0.91665655	-0.00606891

APPENDIX IV

Python coding for calibrating IMU sensor MPU-9250:

```

void magcalMPU9250(float * dest1, float * dest2) {
    uint16_t ii = 0, sample_count = 0;
    int32_t mag_bias[3] = {0, 0, 0}, mag_scale[3] = {0, 0, 0};
    int16_t mag_max[3] = {0x8000, 0x8000, 0x8000}, mag_min[3] =
{0x7FFF, 0x7FFF, 0x7FFF}, mag_temp[3] = {0, 0, 0};

    Serial.println("Mag Calibration: Wave device in a figure eight
until done!");
    sample_count = 128;

    for(ii = 0; ii < sample_count; ii++) {
        readMagData(mag_temp); // Read the mag data
        for (int jj = 0; jj < 3; jj++) {
            if(mag_temp[jj] > mag_max[jj]) mag_max[jj] =
mag_temp[jj];
            if(mag_temp[jj] < mag_min[jj]) mag_min[jj] =
mag_temp[jj];
        }
        delay(135); // at 8 Hz ODR, new mag data is available every
125 ms
    }
    mag_bias[0] = (mag_max[0] + mag_min[0])/2; // get average x
mag bias in counts
    mag_bias[1] = (mag_max[1] + mag_min[1])/2; // get average y
mag bias in counts
    mag_bias[2] = (mag_max[2] + mag_min[2])/2; // get average z
mag bias in counts

    dest1[0] = (float) mag_bias[0]*mRes*magCalibration[0]; // save
mag biases in G for main program
    dest1[1] = (float) mag_bias[1]*mRes*magCalibration[1];
    dest1[2] = (float) mag_bias[2]*mRes*magCalibration[2];
    // Get soft iron correction estimate
    mag_scale[0] = (mag_max[0] - mag_min[0])/2; // get average x
axis max chord length in counts
    mag_scale[1] = (mag_max[1] - mag_min[1])/2; // get average y
axis max chord length in counts
    mag_scale[2] = (mag_max[2] - mag_min[2])/2; // get average z
axis max chord length in counts

    float avg_rad = mag_scale[0] + mag_scale[1] + mag_scale[2];
    avg_rad /= 3.0;
    dest2[0] = avg_rad/((float)mag_scale[0]);
    dest2[1] = avg_rad/((float)mag_scale[1]);
    dest2[2] = avg_rad/((float)mag_scale[2]);
    Serial.println("Mag Calibration done!");}

```

APPENDIX V

Waypoint data for collecting the sensor data from SITL simulation for training the Neural Network Model.

waypoint no.	x	y	z	pitch	altitude	speed
0	1	0	16	0	149.165085	584.409973
1	0	3	22	15	149.163986	28.110001
2	0	3	16	0	149.161697	99.800003
3	0	3	16	0	149.162659	100.730003
4	0	3	16	0	149.164581	100
5	0	3	16	0	149.163757	100
6	0	3	177	-2	0	0
7	0	3	16	0	149.163757	100

REFERENCES

1. Atmeh G. and Subbarao K. (2017), ‘A dynamic neural network with feedback for trajectory generation,’ IFAC-PapersOnLine, vol. 49, no. 1, pp. 367–372.
2. Canciello G., Cavallo A., Amato E. D., and Mattei M. (2017), ‘Attitude and Velocity High-gain Control of a Tilt-rotor UAV,’
3. Chang-Su Park, Hyunjae Lee, Min-Jea Tahk, and Hyochoong Bang (2003), ‘Airship control using neural network augmented model inversion,’ Proc. IEEE Conf. Control Appl. CCA., no. 2, pp. 558–563.
4. De Oliveira H. A. and Rosa P. F. F (2017), ‘Adaptive genetic neuro-fuzzy attitude control for a fixed wing UAV,’ Proc. IEEE Int. Conf. Ind. Technol., pp. 726–731.
5. Dujrene W. R. (2003), ‘Application of artificial intelligence techniques in uninhabited aerial vehicle flight,’ Digit. Avion. Syst. Conf., p. 8.C.3-8.1-6 vol.2.
6. Julian K. D. and Kochenderfer M. J. (2017) ‘Neural Network Guidance for UAVs,’ AIAA Guid. Navig. Control Conf., no. January, pp. 1–14,.
7. Konam S., Rosenthal, and Scherer S. (May, 2017), ‘Vision-Based Navigation and Deep-Learning Explanation for Autonomy,’ no.
8. Lee J., Wang J., Crandall D., Sabanovic S., and Fox G. (2015), ‘Real-Time Object Detection for Unmanned Aerial Vehicles based on Cloud-based Convolutional Neural Networks,’.
9. Maravall D., de Lope J., and Pablo Fuentes J. (2015), ‘Vision-based anticipatory controller for the autonomous navigation of an UAV using artificial neural networks,’ Neurocomputing, vol. 151, no. P1, pp. 101–107.
10. Muliadi J. and Kusumoputro B. (2017), ‘Neural Network Control System of UAV Altitude Dynamics and Its Comparison with the PID Control System,’ J. Adv. Transp.

11. Rimal B. P., Shin H., and Choi E. (2009), 'Simulation of Nonlinear Identification and Control of Unmanned Aerial Vehicle: An Artificial Neural Network Approach,' Commun. Inf. Technol. 2009. Isc. 2009. 9th Int. Symp., pp. 442–447.
12. Sathyan A., Ernest N., Lavigne L., Cazaurang F., Kumar M., and Cohen K. (2017), 'A Genetic Fuzzy Logic Based Approach to Solving the Aircraft Conflict Resolution Problem,' AIAA Inf. Syst. Infotech @ Aerosp., no. January, pp. 1–8.
13. Schmidhuber J. (2015), 'Deep Learning in neural networks: An overview,' Neural Networks, vol. 61, pp. 85–117.
14. Selma B. and Chouraqui S. (2013), 'Neuro-fuzzy controller to navigate an unmanned vehicle.,' Springerplus, vol. 2, p. 188.
15. Verbickas R. (2013), 'Convolutional Neural Network-based Vision Systems for Unmanned Aerial Vehicles,' ProQuest Diss. Theses, vol. MR94243, no. December, p. 138.