

1-3-2020

Close Formation Flight Missions Using Vision-Based Position Detection System

Ashok Sarath Chandra Reddy Irigireddy

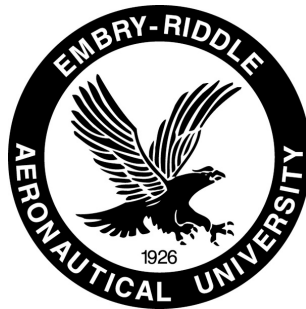
Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Aerospace Engineering Commons](#), and the [Electrical and Computer Engineering Commons](#)

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

Close Formation Flight Missions Using Vision-Based Position Detection System



Ashok Sarath Chandra Reddy Irigireddy
Department of Electrical, Computer, Software and Systems
Engineering (ECSSE)
Embry-Riddle Aeronautical University

Supervisor

Dr. Hever Moncayo

In partial fulfillment of the requirements for the degree of
*Master of Science in Unmanned and Autonomous Systems
Engineering*

January 03, 2020


Close Formation Flight Missions Using Vision Based Position Detection System

by

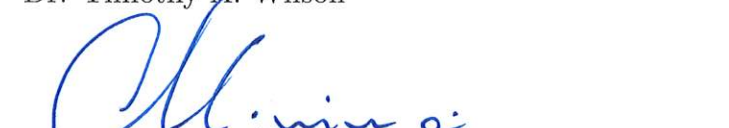
Ashok Sarath Chandra Reddy Irigireddy

This thesis prepared under the direction of the candidate's Thesis Committee Chair, Dr. Hever Monchayo, Department of Aerospace Engineering, and has been approved by the members of the thesis committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in partial fulfillment of the requirements for the Degree of Master of Science in Unmanned and Autonomous Systems Engineering.


THESIS COMMITTEE


Chair, Dr. Hever Monchayo
Member, Dr. Richard Stansbury
Member, Dr. Richard Prazenica
Chair, ECSSE,
Dr. Timothy A. Wilson

2020-01-13
Date


Dean of College of Engineering, Dr. Maj Mirmirani

01/13/2020
Date


Senior Vice President for Academic Affairs
and Provost, Dr. Lon Moeller

01/14/2020
Date

ACKNOWLEDGMENT

I would first like to express my sincere appreciation to my supervisor and committee chair, Dr. Hever Moncayo, who has the substance of a genius: he convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. His office door was always open whenever I ran into a trouble spot or had a question about my research. Without his persistent help, the goal of this project would not have been realized.

I would like to thank my committee members, Dr. Richard Stansbury for always guiding me in the right path as my mentor and providing a valuable input in the research. Also, Dr. Richard Prazenica for giving me the inspiration to perceive towards vision systems and for timely advice and suggestions, when required. The physical and technical contribution of Embry Riddle Aeronautical University is truly appreciated. Without their support and funding, this project could not have reached its goal.

I must express my very profound gratitude to my mother, Padmavathi and my brother, Bharath Chandra for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Finally, I would like to thank my friends: Andrei, Yomary, Juan, Nolan, Diana, Karina for helping me and making a long lasting memories in the lab and also Suyash, Mansi, Omkar and Ravi for being there as a family far from home and supporting me. This accomplishment would not have been possible without them. Thank you.

CLOSE FORMATION FLIGHT MISSIONS USING VISION-BASED POSITION DETECTION SYSTEM

Abstract

by Ashok Sarath Chandra Reddy Irigireddy, Master of Science in Unmanned and
Autonomous Systems Engineering
Embry-Riddle Aeronautical University
January 2020

In this thesis, a formation flight architecture is described along with the implementation and evaluation of a state-of-the-art vision-based algorithm for solving the problem of estimating and tracking a leader vehicle within a close-formation configuration. A vision-based algorithm that uses Darknet architecture and a formation flight control law to track and follow a leader with desired clearance in forward, lateral directions are developed and implemented. The architecture is run on a flight computer that handles the process in real-time while integrating navigation sensors and a stereo camera. Numerical simulations along with indoor and outdoor actual flight tests demonstrate the capabilities of detection and tracking by providing a low cost, compact size and low weight solution for the problem of estimating the location of other cooperative or non-cooperative flying vehicles within a formation architecture.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENT	iii
ABSTRACT	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
NOMENCLATURE	xi
CHAPTER	
1 Introduction	1
1.1 Background	1
1.2 Objectives	2
1.3 Overview of the Thesis	3
CHAPTER	
2 Literature Review	4
2.1 Formation Flight	6
2.2 Computer Vision and Object Tracking	11
CHAPTER	
3 Vision System	16
3.1 You Only Look Once (YOLO)	16
3.1.1 Darknet	20
3.2 Camera Geometry	22
3.3 Formation Flight Network (FF-Net)	25
3.3.1 FF-Net Training	25

3.3.2	FF-Net Depth inclusion	26
-------	----------------------------------	----

CHAPTER

4	Formation Flight Control System Design	28
4.1	Formation Flight in Unmanned Vehicles	28
4.2	Simulation in MATLAB/Simulink	34
4.2.1	Simulated Quad-rotor model	34
4.2.2	Simulation Environment	35

CHAPTER

5	Communication Architecture	38
5.1	Robot Operating System (ROS)	38
5.1.1	Stereo Camera	41
5.1.2	MAVROS and PX4	42
5.1.3	Darknet_ROS	43
5.1.4	Vicon system	43

CHAPTER

6	Simulation Environment For Vision System In AirSim	44
6.1	UAS Simulator	44
6.1.1	Comparison of Various UAS Simulator	46
6.2	AirSim	47

CHAPTER

7	UAS Research Testbed and Facilities	51
7.1	Research Vehicles	51
7.2	On-board Flight Hardware	53
7.2.1	Pixhawk(PX4) Autopilot Flight Controller	53
7.2.2	GPS Receiver Module with Digital Compass	56
7.2.3	915 mhz Telemetry	57
7.2.4	Batteries	58
7.2.5	Spectrum DX8 Transmitter and DSMX Remote Receiver	60
7.3	Jetson TX2 On-board Computer	60
7.4	TaraXL Stereo Camera	61
7.5	Ground Control Station	65

7.5.1	Mission Planner (MP):	66
7.5.2	QgroundControl (QGC)	66
7.6	Flight Testing Facilities	67
7.6.1	Vicon Indoor Testing Facility	67
7.6.2	Outdoor Testing Facility at ERAU Intramural Softball Field . . .	70
 CHAPTER		
8	Experimental Results and Flight Test Program	71
8.1	Formation Flight Numerical Simulations	71
8.2	FF-Net Detection	76
8.3	FF-Net Flight Test Indoor and Outdoor	77
8.3.1	Indoor Flight Results at Vicon Indoor Testing Facility	77
8.3.2	Outdoor Flight Results at ERAU Intramural Soft-ball Field	78
 CHAPTER		
9	Conclusions And Future Work	80
9.1	Conclusion	80
9.2	Future Work	81
REFERENCES		85
 APPENDIX		

LIST OF TABLES

2.1	Some of Military and Civil applications	7
4.1	Controller gains	33
6.1	Comparison between different simulators	47
6.2	System Configuration	50
7.1	Quadrotors Dimensions and Mass Properties	52
7.2	Pixhawk Specifications sUAS, 2013	55
7.3	UBLOX GPS Specifications	57
7.4	Telemetry Specifications Ardupilot, 2019b	58
7.5	NVIDIA Jetson Operating Modes	61
7.6	NVIDIA TX2 Jetson Specifications	62
7.7	NVIDIA TX2 Jetson Specifications	64
7.8	Specifications of Mission Planner and QGC	65

LIST OF FIGURES

2.1	Perspective of NASA’s Global Hawk unmanned aircraft from one of the wings [NASA, 2017]	4
2.2	Migration of birds flying in V formation [Isaaq, 2019]	8
2.3	F/A-18 Autonomous Formation Flight (AFF) [Thomas, 2001]	10
2.4	Formation flying with nano satellites (TU Delft Space Institute) [Institute, 2018]	11
2.5	Computer Vision vs Human Vision [Elgendy, 2019]	12
2.6	R-CNN Architecture [Girshick et al., 2013]	13
2.7	Fast R-CNN Architecture [Girshick, 2015]	14
2.8	Faster R-CNN Architecture [Ren et al., 2015]	14
2.9	Mask R-CNN Architecture [He et al., 2017]	15
2.10	YOLO v3 Architecture	15
3.1	YOLO Network Architecture Redmon, 2016	17
3.2	YOLO CNN layers	18
3.3	YOLO Architecture Taru, 2019	18
3.4	Accuracy and Speed tradeoff on VOC 2007 with different Networks	19
3.5	Accuracy and Speed trade-off on VOC 2007 with different Networks	21

3.6	YOLO v3 performance comparison between Networks	21
3.7	Pinhole Geometry	23
3.8	Pinhole Geometry	23
3.9	FF-Net Training Stats	26
3.10	Camera geometry of Stereo camera	27
4.1	Formation flight geometry	30
4.2	Simulink Skyjib x8 model	35
4.3	Formation Flight model with both the vehicles	36
4.4	Control Allocation	36
4.5	Formation flight block in Outer loop controller	37
5.1	ROS Framework on-board Jetson TX2	41
6.1	Basic architecture of simulator	46
6.2	MAVLink analysis of a quad-copter	48
6.3	User display showing sensor output	49
6.4	Two quads flying in AirSim	49
6.5	FF-Net detection in AirSim	50
7.1	Skyjib and 3DR with all Hardware Components	52
7.2	Skyjib x8 with Jetson TX2 on-board	53
7.3	Pixhawk	54
7.4	Pixhawk Motor configuration for both Vehicles	56
7.5	GPS Receiver Module with Digital Compass	57
7.6	915mhz Telemetry	58

7.7	MAXAMP 6S 22.2 volts LiPo batteries used with Skyjib X8	59
7.8	HRB 5000mah 50c 11.1V lipo battery used with 3DR Aerial Vehicle	59
7.9	Jetson TX2 development Board	60
7.10	Jetson TX2 development Board with 3D Printed Case	63
7.11	Jetson and TaraXL Camera	64
7.12	Mission Planner User Interface	66
7.13	QGroundControl	67
7.14	Vicon System	68
7.15	3D Orthogonal view and Perspective view of quad-rotor in Vicon system	69
7.16	Outdoor Testing Facility at ERAU Intramural Softball Field	70
8.1	Formation Flight without Clearance	72
8.2	X, Y, Z tracking without clearance	73
8.3	roll, pitch, yaw tracking without clearance	73
8.4	Formation Flight with Clearance	74
8.5	X, Y, Z tracking with clearance	75
8.6	roll, pitch, yaw tracking with clearance	75
8.7	outdoor FF-Net detection for class Raft	76
8.8	FF-Net detection for class Raft	77
8.9	Compared results of the FF-Net with Vicon System	78
8.10	Flight test data trajectory	79
8.11	Compared results of the FF-Net with GPS data	79
1	FF-Net detection with class "Drone"	87

2	FF-Net detection with class "Drone"	88
---	---	----

Nomenclature

Symbols

\ddot{l}, \ddot{f}	second derivative of distance errors
$\dot{\delta}, \ddot{\delta}$	derivatives of difference between yaw angles of aerial vehicle
\dot{l}, \dot{f}	first derivative of distance errors
$\tau_{\theta}, \tau_{\phi}, \tau_{\psi}, \tau_z$	inner loop controller parameters
δ	difference between yaw angles of aerial vehicle
ω	control allocation parameter
θ, ϕ, ψ	roll, pitch and yaw angles of aerial vehicle
a_x, a_y	acceleration in x, y direction
a_{xL}, a_{yL}	acceleration in x, y direction of leader
c_x, c_y	Image center coordinates
f	focal length
f_c	forward distance
h	vertical distance

K	controller gain
l_c	lateral distance
m	mass of aerial vehicle
p_x, p_y	Anchors dimensions for the box
Q	3D world point of object
q	coordinate of object in camera's image plane
u	thrust
u, v	pixel coordinates in camera's image plane
V_x, V_y	velocity in x, y direction
V_{xL}, V_{yL}	velocity in x, y direction of leader

Abbreviations

$3DR$	3D Robotics
ADC	Analog to Digital Converter
$ADCL$	Advanced Dynamics and Control Lab
$AirSim$	Aerial Informatics and Robotics Simulation
API	Application Programming Interface
$CMOS$	Complementary metal–oxide–semiconductor
CNN	Convolutional Neural Network
$COCO$	Common Objects in Context

<i>CPU</i>	Central Processing Unit
<i>DSM</i>	Digital Spectrum Modulation
<i>ERAU</i>	Embry-Riddle Aeronautical University
<i>FF – Net</i>	Formation Flight Network algorithm
<i>FOV</i>	Field of View
<i>FPN</i>	Feature Pyramid Network
<i>GCS</i>	Ground Control Station
<i>GPS</i>	Global Positioning System
<i>GPU</i>	Graphics Processing Unit
<i>GUI</i>	Graphical User Interface
<i>HIL</i>	Hardware In the Loop
<i>IMU</i>	Inertial Measurement Unit
<i>LAN</i>	Local Area Network
<i>LiDAR</i>	Light Detection and Ranging
<i>MATLAB</i>	Matrix Laboratory
<i>MAVLink</i>	Micro Air Vehicle Link
<i>MAVROS</i>	Micro Air Vehicle Robot Operating System
<i>MOCAP</i>	Motion Capture
<i>PCL</i>	proportional–Integral–Derivative

<i>PID</i>	proportional–Integral–Derivative
<i>R – CNN</i>	Regions with Convolutional Neural Networks
<i>ROS</i>	Robot Operating System
<i>SDK</i>	Software Development Kit
<i>SIL</i>	Software In the Loop
<i>SPI</i>	Serial Peripheral Interface
<i>SSD</i>	Single Shot MultiBox Detector
<i>UART</i>	Unmanned Asynchronous Receiver/Transmitter
<i>UAS</i>	Unmanned Aerial System
<i>UE4</i>	Unreal Engine 4
<i>VRPN</i>	Virtual-Reality Peripheral Network
<i>WVGA</i>	Wide Video Graphics Array
<i>YOLO</i>	You Only Look Once

Subscripts

b_h	height of recognized bounding box by YOLO
b_w	width of recognized bounding box by YOLO
b_x	x component of recognized bounding box by YOLO
b_y	y component of recognized bounding box by YOLO
f_x	focal length in x direction

f_y	focal length in y direction
s_x	x parameter to convert meters to pixel distances
s_y	y parameter to convert meters to pixel distances
t_h	height of network output from CNN
t_w	width of network output from CNN
t_x	x component of network output from CNN
t_y	y component of network output from CNN

Chapter One

Introduction

1.1 Background

In recent years, Unmanned Aircraft Systems (UAS) persist as a high priority asset to military and a go-to tool for many civilian applications and have seen a rapid growth in the both the areas. The major advancement in this field is due to heavy use of UAS by Armed Forces. According to a forecast global spending on UAS will increase from \$6.6 billion dollars in 2013 to \$11.4 billion in 2022 Harrison, January 30, 2013 . A part of this growth is from civilian applications such as terrain mapping, crop dusting, commercial transport and by first responders for disaster relief, search and rescue, and medical deliveries. Due to this improved interest in this field, dramatic advancements have been made driven by parallel advancements in subsystem technologies. UAS is adopted by many applications that demand greater range and endurance with its increased reliability and autonomy. Yet, small UAS applications will always be limited by size, area of coverage and reduced payload capability. Many of the applications considered for small unmanned systems compensate these limitations by using multiple systems, capitalizing on cost reduction and increasing the redundancy of the system using multiple UAS in required formation. These vast applications have lead to significant study in the area of formation flying in unmanned systems.

As the technological development of close proximity operations progresses and the cost

and size of such systems decrease, there is a perceived interest on the implementation of flight formation to accomplish several types of missions. Formation operations have enabled a wide variety of applications ranging from reduced fuel consumption using wake vortex profiles, aerial refueling services, cooperative space exploration missions, surveillance, among others. A formation flight guidance law enables each UAS to maintain its relative position in the formation, which allows the UAS to be efficiently and safely controlled while they perform their mission satisfactorily. It is this UAS Formation Flight control strategy development and evaluation process that serves as the topic area of this thesis.

1.2 Objectives

Successful sustained formation flight within small three dimensional spaces requires high accuracy, real-time and low processing requirements for position estimation. By using Global Positioning System (GPS) sensors, the estimation of the relative position of an agent with respect to others within a formation would be limited to high separations as the resolution of position provided by this sensor is in the order of meters. Even with highly accurate GPS, the problem translates to the latency or delay generated between agents. In addition, close proximity formation flight missions within GPS-denied environments such space exploration, urban scenarios or indoor applications make this sensor unsuitable for this type of application. Common methods such as radars or radio telemetry have been previously used for measuring the relative displacement between leader and follower agents. These approaches, however, limit the applications to large distances. Alternatively, vision systems can provide solutions for short distances that can be directly applied to closed-formation missions. With the use of advanced camera systems along with state-of-the-art vision-based algorithms the same results can be obtained with precision and accuracy. This can be easily achieved as we can afford the computational power on board by reducing the use of more sensors and more hardware to complete the same operations.

The research work presented in this thesis resulted in a conference paper accepted for publishing in American Institute of Aeronautics and Astronautics (AIAA) SciTech 2020 conference: Sarath & Moncayo, 2020 (Vision Based Relative Navigation for Close-Formation Flight Missions).

1.3 Overview of the Thesis

A literature review that includes explanation and background of unmanned vehicles, formation flight, formation flight in unmanned vehicles, vision systems, object tracking, and some simulation parameters are presented in Chapter 2. Chapter 3 gives an overview and a detailed explanation of vision system, object detection algorithm and its training process involved in this research. Chapter 4 presents the mathematical model and derives the equations of formation flight of unmanned systems with and without clearance distance. This chapter also includes the mathematical implementation of inner and outer loop controllers along with the required controller gains. This chapter also explains the setup of a simulation environment in MATLAB/Simulink. The simulation model, Formation flight model and control laws are presented with two models of simulated quad-copters.

Chapter 5 explains the communication protocols, packages and communication between protocols which are controlled by Robot Operating System (ROS). Chapter 6 gives a brief explanation of a graphical simulation environment used to validate the vision system. This is followed by an overview of hardware and software components used in the research test-bed including an introduction to flight test facilities both vicon indoor testing facility in Advanced Dynamics and Control Lab at ERAU and outdoor testing facility at ERAU's intramural softball field in Chapter 7. All the experimental and flight test results of Formation Flight are presented in Chapter 8, followed by conclusions of the experiments in Chapter 9 and ending this thesis document with future work in Chapter 10.

Chapter Two

Literature Review

UAS is an aircraft without human operators on-board. They are also commonly referred to as Drones due to their resemblance to the male bee. They are initially designed to carry lethal and non-lethal military payload for missions such as reconnaissance, command and control, and deception. One of the military drones used for intelligence, surveillance, and reconnaissance (ISR) is shown in Fig.2.1.



Figure 2.1 Perspective of NASA's Global Hawk unmanned aircraft from one of the wings [NASA, 2017]

A UAS is defined as a "powered, aerial vehicle that does not carry a human operator,

uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" [Tice, 1991]. UAS are available in different levels of autonomy like Remotely Piloted Vehicles (RPV's), which are capable of being controlled from a distant location through a communication link, or Unmanned Autonomous Systems (UAS), which can carryout complete autonomous or predefined missions using the on-board sensors, controllers and flight computer. They come in designs ranging from full scaled aircraft design to ball-shaped helicopter blades. Sizes vary from a vehicle larger than a commercial aircraft to small vehicles which can fit in a pocket. Ideally they are designed to be recoverable and reused except if mission is intentionally expendable.

UAS is categorized depending on size, shape, weight and application some of which are detailed below:

1. Multi-Rotors: This UAS can hover, vertically take-off and land (VTOL) and have high maneuverability. This includes helicopters, duck-type, tilting and multi-rotor rotorcraft.
2. Blimps: Unlike fix-rotor UAS, the blimps is a balloon type UAS which has long endurance but cannot cruise at high speeds.
3. Fixed-wings: This category includes UAS which are model airplanes particularly used for high cruise speed and long endurance.
4. Flapping wing: This type of UAS is inspired from birds and bees. They utilized wing that flap instead of being propelled via motors or engines.

Depending on weight and size they can be categorized as:

1. Micro/Mini: UAS which has very small takeoff weight falls under this category. The takeoff weight is less than 30kg and can reach a maximum altitude of less than 300m with a lower endurance of less than 2 hours.

2. Tactical: UAS which has maximum takeoff weight between 150kg to 1,500kg is categorised as Tactical UAS. The Tactical UASs can reach a maximum flight altitude of 8km with an endurance of up to 48 hours.
3. Strategic: UAS in this category has takeoff weight between 2,500kg to 12,500kg and can reach an altitude of 15km to 20km.
4. Special Task: UAS which is specifically customised for a specific task comes under this category.

UAS have become an integral part of military for complex tasks including surveillance, reconnaissance, precision strike and aerial refueling missions in the presence of disturbances, failures, and complicated battlefield subjected to uncertainties and variations [Duan et al., 2013].

Now, UASs are being used in civil applications for remote sensing, goods delivery, agriculture, wireless coverage, security and coverage and real-time monitoring of roads, pipeline and civil infrastructure due to their small size, noiseless operation, hovering capability, low cost, high-mobility, ease of deployment and low maintenance. Some of the applications in military and non-military mission are presented in Table.2.1. For these reasons a lot of research is being done in this field and on the various control problems associated with it. One of such research is done in formation flight phenomenon and a brief introduction is given below.

2.1 Formation Flight

Humans have been gaining inspiration for solving engineering and design challenges by observing the natural phenomena, which is known as bio-mimicry (for example aeroplanes are originally designed by studying birds). While engineers looked at birds to fly, some looked at birds to learn how to fly more efficiently [Ning, Flanzer, and Kroo, 2010]. Among

Military and Civil applications	
Military	Civil
<ul style="list-style-type: none"> - Battle Damage Assessment (BDA) - Intelligence, surveillance, and reconnaissance (ISR) - Reconnaissance Surveillance and Target Acquisition (RSTA) - Surveillance for peacetime and combat Synthetic Aperture Radar (SAR) - Radar (SAR) - Deception operations - Maritime operations (Naval fire support, over the horizon targeting, anti-ship missile defence, ship classification) - Electronic Warfare (EW) and SIGINT (Signals Intelligence) - Special and psyops - Meteorology missions - Route and landing reconnaissance support - Adjustment of indirect fire and Close Air Support (CAS) - Radio and data relay - Nuclear cloud surveillance - Military roles according to arm and forces 	<ul style="list-style-type: none"> - Agricultural operations - Pipeline survey - Fire fighting - Agriculture and forestry - Environmental monitoring - Power line survey - Disaster and crisis management search and rescue - Aerial mapping and meteorology - Communications relay - Law enforcement - Aerial photography - Border patrol - Policing duties - Traffic spotting - Research by university laboratories

Table 2.1 Some of Military and Civil applications



Figure 2.2 Migration of birds flying in V formation [Isaaq, 2019]

the many insights gleaned from nature was the observation that migratory birds often flew together in formations as shown in Figure.2.2 which was a fascinating concept in natural occurrence for ages.

In AD 79, Pliny the elder noted that birds flew 'like fast galleys, cleaving the air more easily than if they drove at it with a straight front', which has become a cornerstone to study in this phenomenon. From then many ideas have been proposed to explain it but Peter Lissaman and Carl Shollenberger in 1970 were the first to publish in detail the exact benefit of the flock flying in formation with aerodynamic interactions [Steven Portugal, 2016]. The authors have predicted the exact position of each individual bird within the formation for maximum utilization of the resources which have persisted as the gold standard in the study of formation flight. This gives the basic principle that an object flying in a fluid produces lift by creating downward momentum within its span. When a wing is generating lift, the air on the upper side of the wing has lower pressure relative to the bottom side, and air flows from below the wing and out around the wingtips. At the wingtips, vortices – circular patterns of rotating air around the wingtip are generated, with a wingtip vortex trailing from the tip of each wing; this results in a vortex trailing from the right-hand wing and a vortex trailing from the left-hand wing. These vortices generate upwash, creating a

favourable airflow for other birds flying abreast that they could take advantage of if they flew in the optimal position to capture the upwash. The lift provided by the upwash causes a reduction in the lift power that trailing individuals must produce, and thus can bring about an energetic saving. Between these two regions of upwash, however, there is a large region of downwash – created as a result of air being pushed down as the bird moves forward – that most birds want to avoid [Steve Portugal, 2016].

Recently biologists like Weimerskirch et al. instrumented trained pelicans with a heart-rate logger and measured their wing-beat frequency using a digital camera [Weimerskirch et al., 2001] and found that the pelicans exhibited a significantly decreased heart rate and wing-beat frequency when they are flying in a formation flight.

This concept is later introduced in the engineering field learning from birds and has been studied extensively with numerous applications in aeronautics and space systems. One of the initial basis is formed by Multhopp and Black in 1998, as they looked at modeling airplanes with vortex lattice methods as well as horseshoe models with viscous cores and found that the two methods produced similar trends, but differed in the predicted lateral position for maximum induced drag savings Blake and Multhopp, 1998. Later in 2001, Wagner et al. included the effect of trimming in roll using aileron deflection which the optimal lateral position for maximum induced drag savings Jacques et al., 2001 which lead to many studies in this concept [Mason and Iglesias, 2002] [King and Gopalarathnam, 2005]. As this concept is proven many flight tests were conducted to evaluate formation flight. From the flight tests conducted by Hummel in 1996 with Dornier Do-28 and Wagner et al. in 2002 shows reduction in power usage by 15% and fuel flow by 8% [Wagner et al., 2002]. Autonomous Formation Flight Project in 2002, funded by NASA’s Revolutionary Concepts Program showed a maximum fuel flow reduction of 18% for the trailing aircraft using two F/A-18 aircraft [Vachon et al., 2003] [Cobleigh, 2002] [Ray et al., 2002] which is shown in Figure.2.3 More recent studies and analysis of formation flight have proved that there is at least 13% reduction in fuel burn is achievable in commercial areas [Bower, Flanzer, and



Figure 2.3 F/A-18 Autonomous Formation Flight (AFF) [Thomas, 2001]

Kroo, 2009].

The benefits of formation flight include fuel savings at certain close formation positions, cooperative task allocation, mission success in terms of redundancy and battle damage assessment and improved efficiency in air traffic control for aerial applications and an accurate control of formation flight in space vehicles one of which is shown in Figure.2.4 will be a greater asset in autonomous rendezvous and docking, large-aperture space telescopes and robotic assembly of space structures.

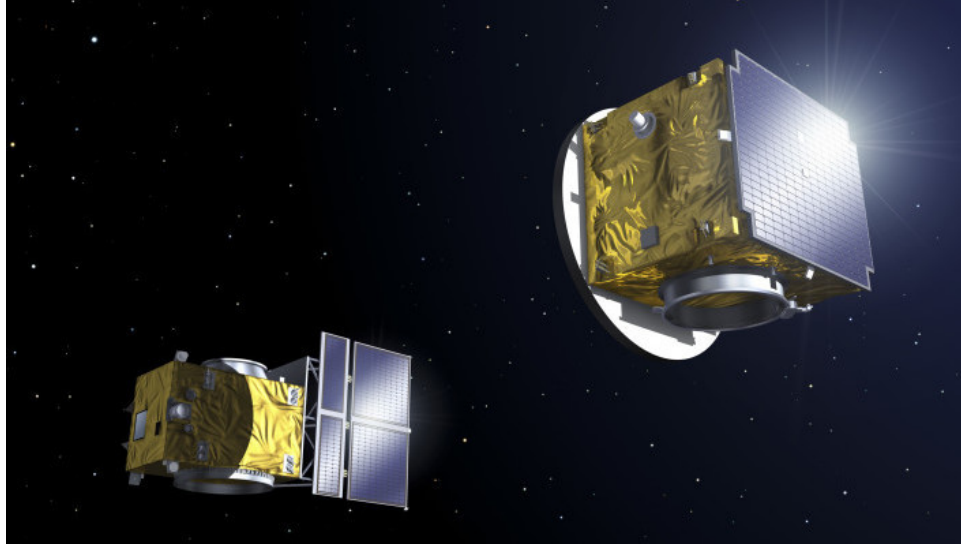


Figure 2.4 Formation flying with nano satellites (TU Delft Space Institute) [Institute, 2018]

2.2 Computer Vision and Object Tracking

Computer Vision (CV) is a field of study focused on artificial intelligence that trains computers to interpret, understand and see the visual world. It is a sub-field of artificial intelligence and machine learning, which uses general learning algorithms, specialized methods to identify and classify digital images from cameras, videos and deep learning models, machines then react to what they “see”.

Neural Networks used in 1950s to detect the edges of an object and to sort them into shapes was considered as the first use of CV. Later in 1970, it was used to recognise and read the text and characters to the blind which then flourished into facial recognition 1990. Now the rapid advances in artificial intelligence, machine learning and deep learning has lead to extend far more than human capabilities in tasks related to managing data and sorting it. A relation to computer vision and human vision system is given in Figure.2.5.

The advances like built-in cameras in mobile technology, affordable high performance computing have made systems more actuate than humans at detecting and reacting to visual inputs. In less than a decade the accuracy for object identification and classification have

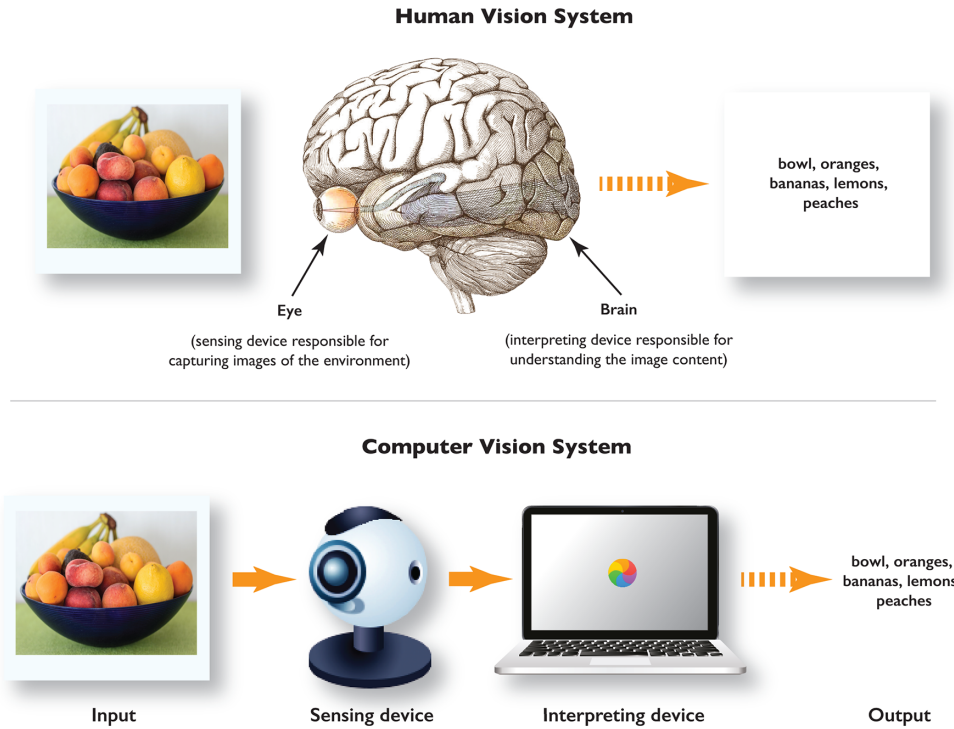


Figure 2.5 Computer Vision vs Human Vision [Elgendy, 2019]

gone from 50 percent to 99 percent. One of the important factors behind the rapid growth of computer vision is the amount of data that is grown then used to train and make computer vision better. Recent algorithms like Convolutional Neural Networks (CNN) take advantage of this hardware and software capabilities to make CV faster and more accurate.

The main goal of CV is to understand and interpret the content of digital image which involve extracting a description from the image, which may be an object, a text description, a three-dimensional model. Today's AI systems can go a step further and take actions based on an understanding of the image. Some of the methods used are object classification, object identification, object verification, object detection, object tracking, object recognition, object segmentation and object landmark detection.

One of the main concepts covered in this research is object detection. Object detection

identifies any defined object in a digital image. The models use an X,Y coordinate to create a bounding box and recognizes single or multiple objects in the box. Object detection is widely applied in autonomous systems, self-driving cars and for video surveillance. This usually takes two processes: classification of objects type and drawing a box around it. Common object detection architectures are:

1. R-CNN : This technique uses a combination of segmenting objects and using CNN to localize. It is named as Regions with CNN features (R-CNN) as it combines regional proposals with CNN which yeilds performance boost with domain-specific fine-tuning and an architecture is shown in Figure.2.6. This model extracts more than 2000 bottom-up region proposals from an image and computes the features for each proposal using a large CNN. Then using class-specific linear support vector machines (SVMs) it classifies each region acheiving 53.7 % of mean average precision (mAP) on PASCAL VOC 2010.

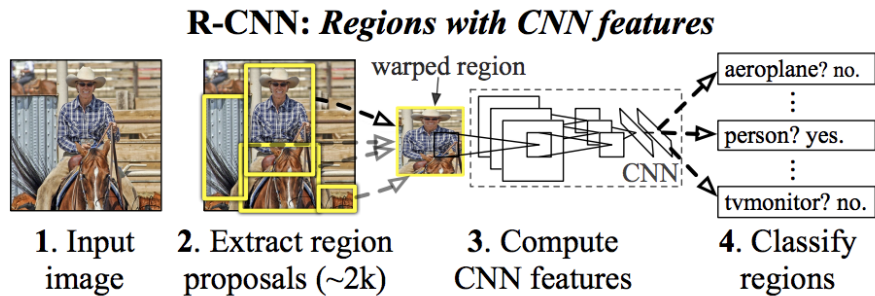


Figure 2.6 R-CNN Architecture [Girshick et al., 2013]

2. Fast R-CNN: This technique takes an image as input as well as a set of object proposals and processes the image with convolutional and max-pooling layers to produce a convolutional feature map from which a fixed-layer feature vector is extracted that are fed to fully connected layers shown in Figure.2.7. These then produce softmax probability estimates over several object classes and four real-value numbers for each of the classes which represent the position of the bounding box for each of the objects.

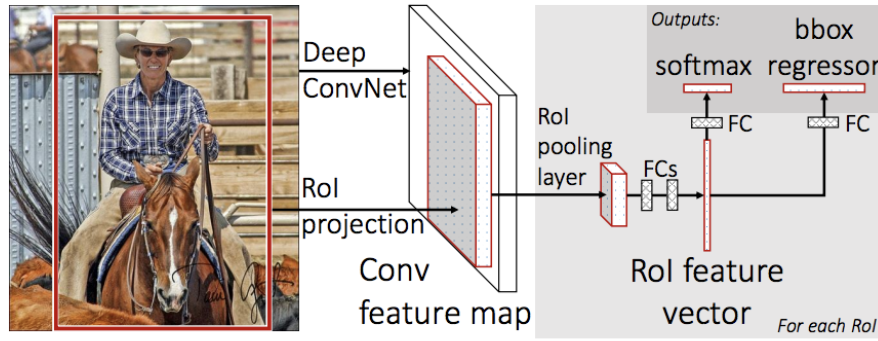


Figure 2.7 Fast R-CNN Architecture [Girshick, 2015]

Fast R-CNN achieves mAP of 66% on PASCAL VOC 2012, which is lot better than R-CNN and is implemented in python and in C++ using Caffe.

3. Faster R-CNN: This technique uses a Deep Convolutional Network (DCN) as shown in Figure.2.8 for proposing regions and a Fast R-CNN detector which uses the regions. It takes an image as input and generates rectangular object proposals with objectiveness score.

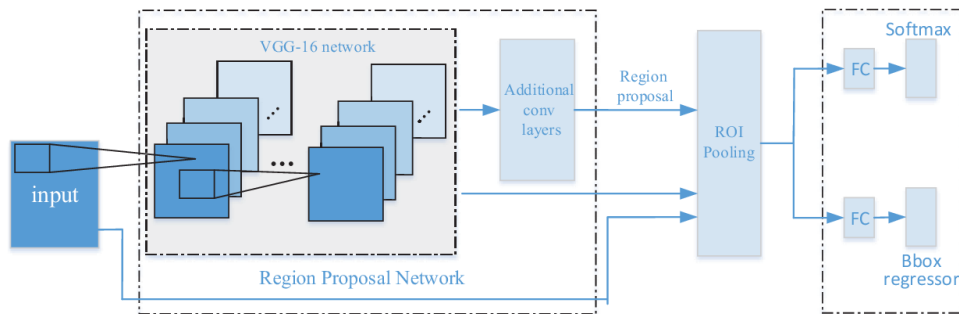


Figure 2.8 Faster R-CNN Architecture [Ren et al., 2015]

4. Mask R-CNN: This technique is an extension of Faster R-CNN in which objects are classified and localized using a bounding box and semantic segmentation to classify pixels into categories. This Mask R-CNN produces class label and bounding box. Mask R-CNN architecture is shown in Figure.2.9.
5. You Only Look Once (YOLO): In this technique each bounding box contains x , y , w ,

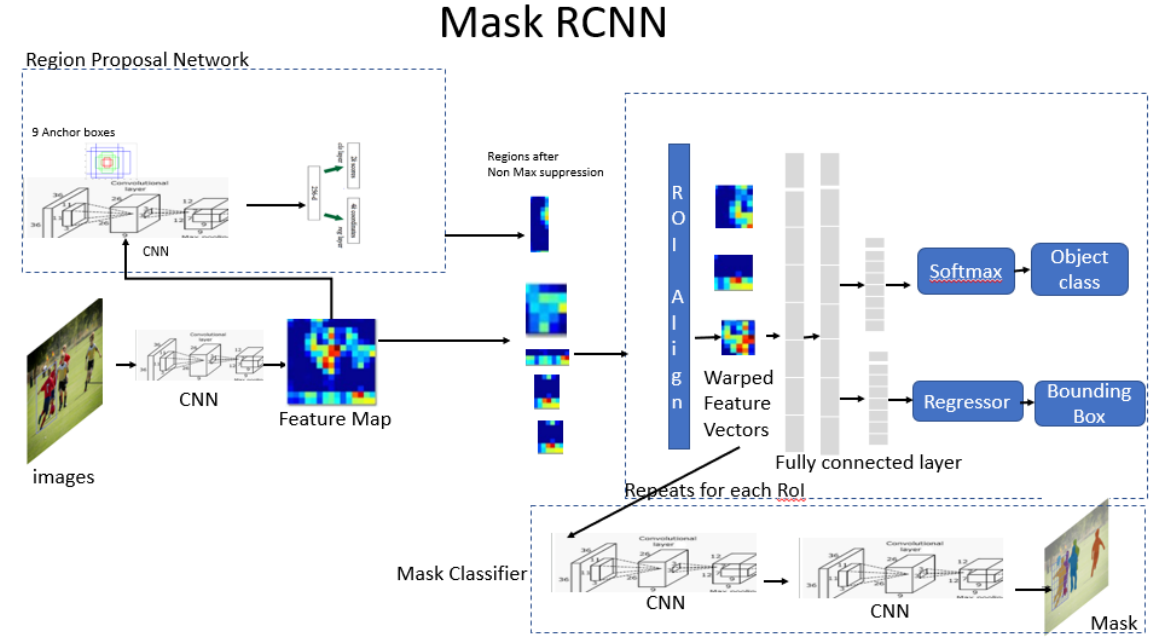


Figure 2.9 Mask R-CNN Architecture [He et al., 2017]

h , confidence and is predicted by features from the entire image. Where (x, y) is the center of bounding box, w and h are the predicted width and height. YOLO is implemented as CNN and these layers are responsible for extracting features, coordinates and output probabilities are predicted by fully connected layers and its architecture is shown in Figure.2.10.

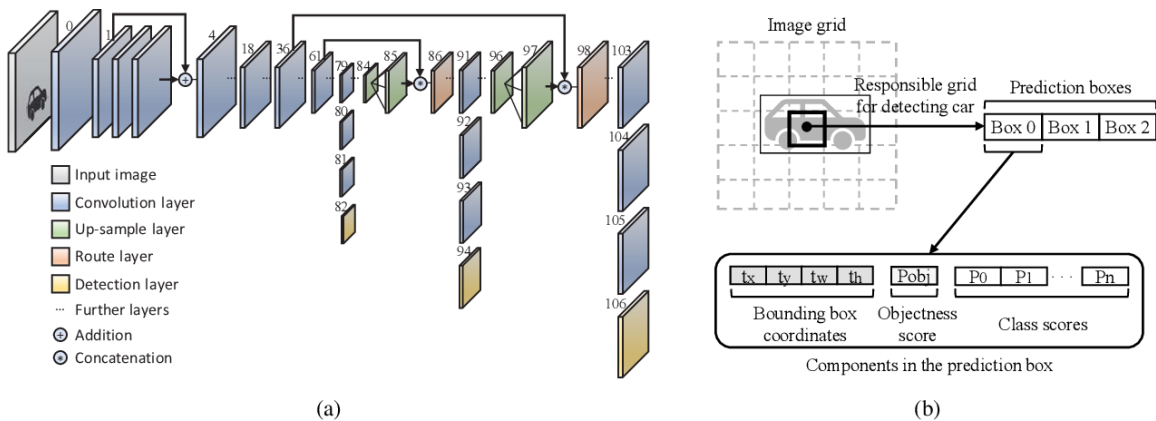


Figure 2.10 YOLO v3 Architecture

Chapter Three

Vision System

This chapter explains the Vision System used in this thesis and gives a description of the packages used. The first section explains the Machine Learning (ML) algorithm YOLO and Darknet, a framework of CNNs which acts as a basis of training source for vision system. This is followed by camera geometry explaining Pinhole camera geometry of the monochrome camera and the stereo camera to derive world coordinates from camera pixel values. This chapter is ended with the Formation Flight Network (FF-Net) which is created by using YOLO architecture and integrating it with stereo camera geometry to give real world coordinates of objects detected relative to camera position along with the training process of FF-Net.

3.1 You Only Look Once (YOLO)

There are two kinds of algorithms used for object detection which are classified as

1. Algorithms based on Classification: In these algorithms the initial step it is selected from image intersecting regions and those regions are classified using CNN. The algorithms in this category are slow compared to others. This category contains R-CNN, Fast R-CNN and Faster R-CNN.
2. Algorithms based on Regression: In this category the classes and bounding boxes are

predicted by looking at the whole image at once. YOLO is the most common example of this category.

YOLO algorithm is a state-of-the-art, realtime, extremely fast and accurate object detection system. YOLO architecture uses Convolutional Neural Networks (CNN) and has been modified in this study to guarantee light processing and is suitable for real time application. With latest released version of YOLO, the tradeoff between speed and accuracy is achieved by simply changing the size of the model with minimum or no training required. General object detection algorithms use classifier or localizer approaches to perform detection and apply the model to an image at multiple locations and scales. However, YOLO uses a neural networks to divide the image into regions and predicts bounding boxes which are weighted by predicted probabilities for each region. Figure.3.1 shows the main steps used by the YOLO algorithm.

YOLO network is fed with input images to predict 3D tensors corresponding to 3 scales which are designed for different size object detection. In Figure.3.1 the scale 13x13 is taken as an example. For this scale, the input image is divided into 13x13 grid cells with each grid cell corresponding to a 1x1x255 voxel inside a 3D tensor.

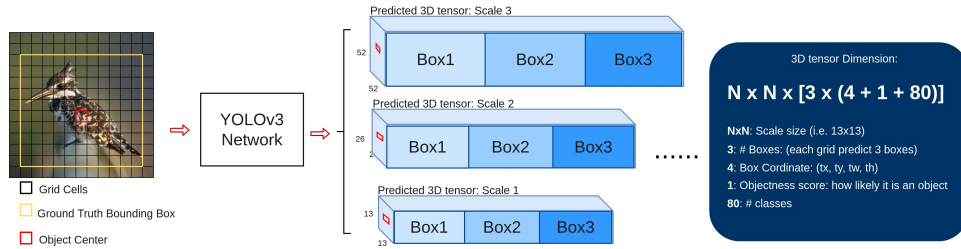


Figure 3.1 YOLO Network Architecture Redmon, 2016

Here, 255 is taken by using $3 \times (4 + 1 + 80)$ equation values from third part of the image. The final values in a 3D tensor are also shown in the third stage of the Figure.3.1. The method uses K-mean clustering to classify the total boxes from Common Objects in Context (COCO) data-set to 9 clusters before training. This results in 9 sizes chosen from 9 cluster,

3 for 3 scales shown in Figure.3.2. This Information, which is known beforehand, helps the system to learn to compute box coordinates precisely [Redmon, Divvala, et al., 2016].



Figure 3.2 YOLO CNN layers

Unlike Regions with Convolutional Neural Networks (R-CNN), YOLO looks at the whole image with single network. YOLO integrates 75 convolutional layers as a feature-learning based network which can handle variable image sizes. The algorithm does not use pooling and an additional convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to Pooling as described in Figure.3.3.

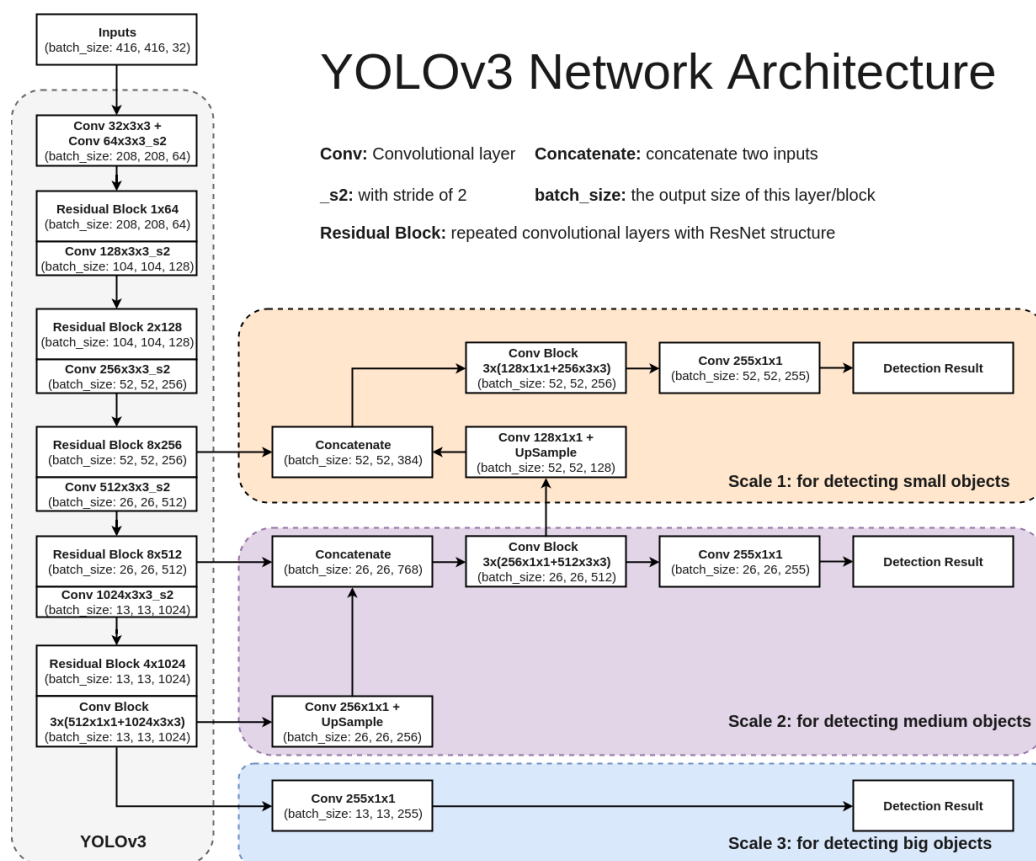


Figure 3.3 YOLO Architecture Taru, 2019

As shown in Figure.3.3, the algorithm works on a 13×13 feature map which is best for localizing finer grained features of smaller objects. Then, it adds a pass-through layer that brings features from an earlier layer at 26×26 resolution while R-CNN and Single Shot Multi-Box Detector (SSD) run at various features maps to get the range of resolutions. The pass-through layer similar to the identity mappings in Residual Neural Network (ResNet), stacks adjacent features into different channels instead of spatial locations to concatenate high resolution features with low resolution features. This turns the $26 \times 26 \times 512$ feature map into a $13 \times 13 \times 2048$ feature map to access fine grained features [Redmon, 2016]. The following equations as shown in Figure.3.4 describe how the network output is transformed to obtain bounding box predictions.

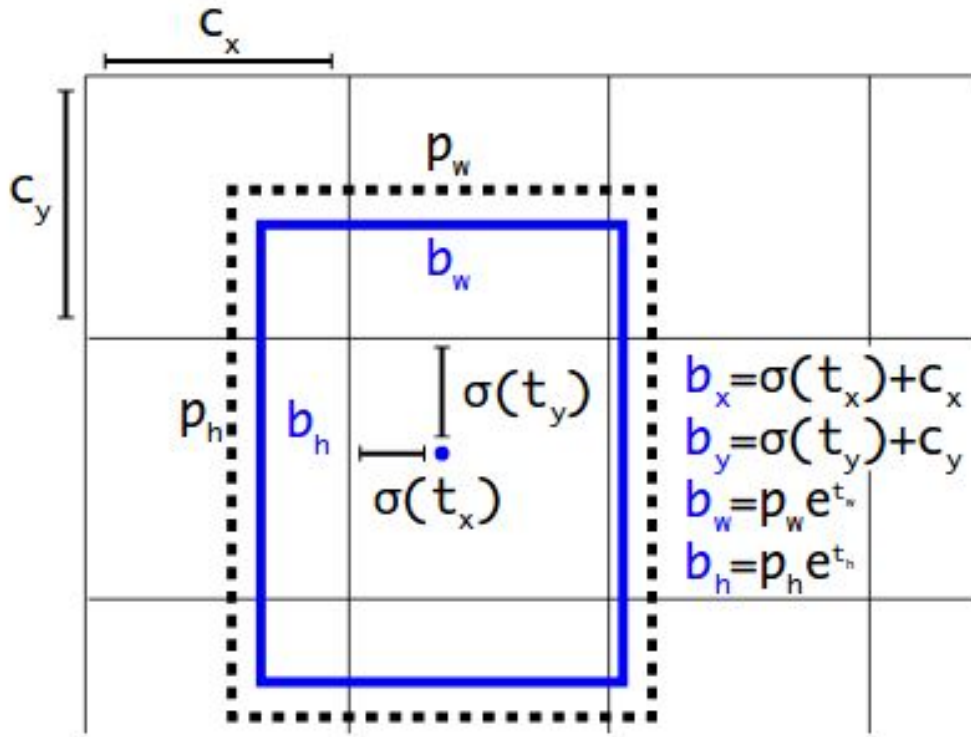


Figure 3.4 Accuracy and Speed tradeoff on VOC 2007 with different Networks

$$b_x = \sigma(t_x) + c_x \quad (3.1)$$

$$b_y = \sigma(t_y) + c_y \quad (3.2)$$

$$b_w = \rho_w e^{t_w} \quad (3.3)$$

$$b_h = \rho_h e^{t_h} \quad (3.4)$$

where b_x , b_y , b_w , b_h are x, y center coordinates, width and height t_x , t_y , t_w , t_h are network outputs, c_x , c_y are top-left coordinates of grid and p_w , p_h are anchors dimensions for the box. A log-space transform is applied to the output and then multiplied with an anchor to predict the dimensions of the bounding box. Height and width of the image are obtained by normalized resultant predictions, b_w , b_h [Redmon and Farhadi, 2018].

There are three versions of YOLO which are given below:

1. YOLO v1: Initial version of YOLO is released in May 2016, which sets as a core algorithm. This version of network is inspired by GoogleNet [Redmon, 2016]. It has 24 convolutional layers working as feature extractors and 2 dense layers for doing the predictions. The loss that the algorithm minimises takes into account the predictions of the locations of the bounding boxes, their sizes, the confidence scores for the said predictions and the predicted classes.
2. YOLO v2: This version was released in December 2016 and introduces anchor boxes and to detect small objects better [Redmon and Farhadi, 2016].
3. YOLO v3: This version is the recent release in April 2018, which is built on new Darknet framework to have 53 convolutional layers and can predict bounding boxes at different scales [Redmon and Farhadi, 2018].

Comparisons between different models is shown in Figure.3.5 and Figure.3.6.

3.1.1 Darknet

Darknet is an open source framework written in C / CUDA to train neural networks written by J. Redmon [Redmon, 2013–2016]. Darknet sets the architecture of the YOLO and is used

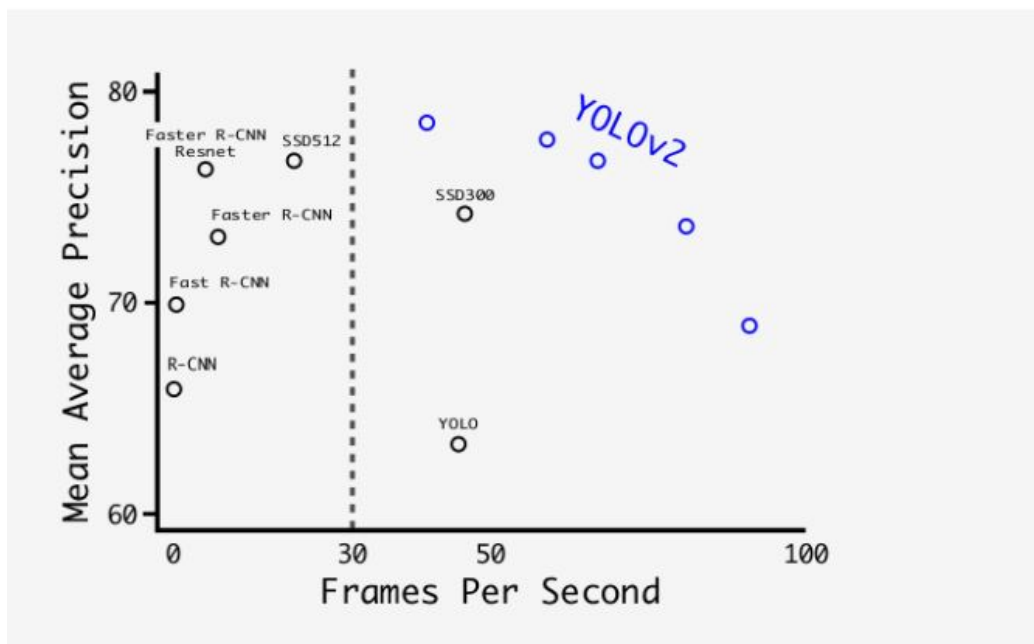


Figure 3.5 Accuracy and Speed trade-off on VOC 2007 with different Networks

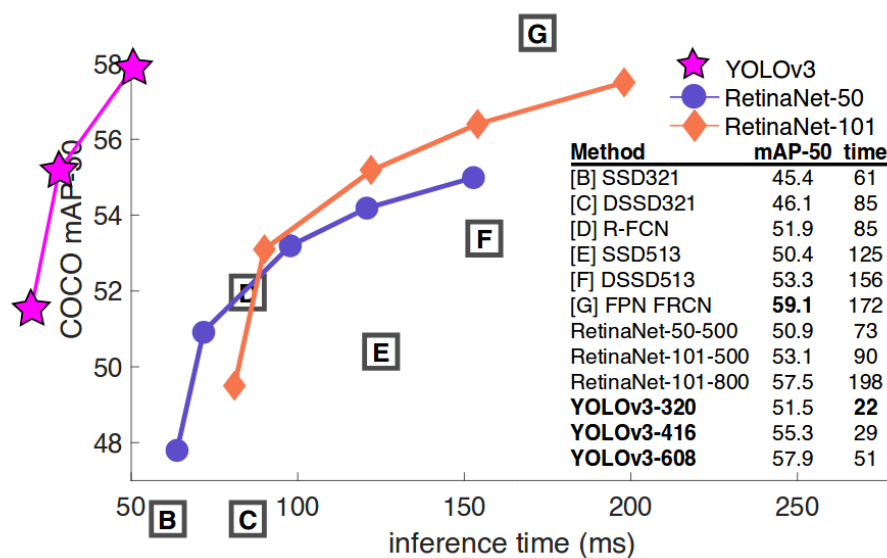


Figure 3.6 YOLO v3 performance comparison between Networks

for training it. This framework allows YOLO to make realtime predictions by allowing it to use GPU.

3.2 Camera Geometry

The camera is a mechanism by which a computer can see and record the world around. In this thesis a pinhole camera geometry is used to estimate the real-world coordinates of the aerial vehicle which is explained below.

One of the first type of cameras is the Pinhole camera which use "Camera Obscura" as fundamental principle. It uses the same science as present-day cameras use whereby light travels through a small hole in a dark box to form a picture. The mathematical relationship between the coordinates of a point in a three-dimensional space and its projection onto the image plane of an ideal pinhole camera, where the camera aperture is described as a point and no lenses are used to focus light, is described in the pinhole camera model [Fusiello, 2005]. Pinhole camera model can only be used as a first order approximation of the mapping from a 3D scene to a 2D image.

Figure.3.7 shows the coordinate frame of a camera with the center of projection at O, focal length f between camera center to image plane, a 3D point $Q = (X, Y, Z)$ is imaged on the camera's image plane at coordinate $q = (u, v, f)$ and the principal axis parallel to Z axis.

We can get from Figure.3.8 from the above derived projections

$$\frac{f}{Z} = \frac{u}{X} = \frac{u}{Y} \quad (3.5)$$

$$u = \frac{fX}{Z} \quad (3.6)$$

$$v = \frac{fY}{Z} \quad (3.7)$$

which can be written as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (3.8)$$

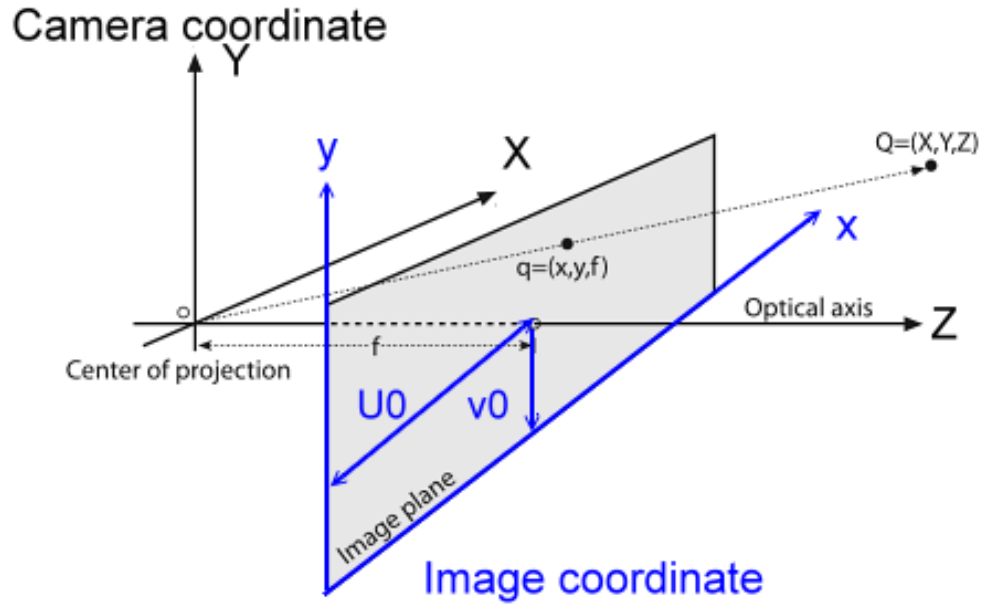


Figure 3.7 Pinhole Geometry

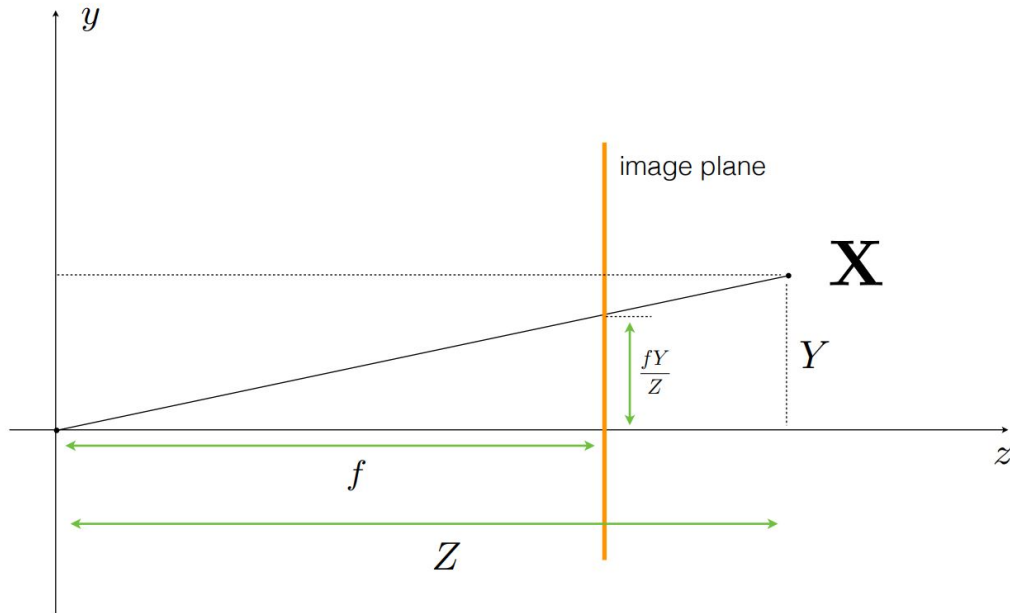


Figure 3.8 Pinhole Geometry

which can be written as $q = KQ$

In the above equation X, Y, Z , U, V, W and focal length f are measured in meters or millimeters. To convert to pixel distances the scale factor s_x and s_y are introduced and can

be given as

$$f_x = s_x * f \quad (3.9)$$

$$f_y = s_y * f \quad (3.10)$$

These coordinates are converted into pixel distances as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (3.11)$$

Often pixel coordinates are not given with respect to the frame at the center of the optical axis but are given in a positive quadrant. So, to keep the frame in the center a translation is performed resulting in:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad (3.12)$$

where,

- u, v or in some cases x, y are the projection point coordinates in pixels.
- f_x, f_y are focal lengths.
- c_x, c_y are image center coordinates.
- X, Y, Z are 3D point coordinates in world coordinate space

which gives the coordinates of pixel points as

$$u = \frac{f_x * X}{Z} + c_x \quad (3.13)$$

$$v = \frac{f_y * Y}{Z} + c_y \quad (3.14)$$

3.3 Formation Flight Network (FF-Net)

In this thesis, a detection system called Formation Flight Network (FF-Net) was implemented using Darknet framework [Redmon, 2013–2016] combined with a machine learning based neural network algorithm YOLO to detect a flying vehicle in the camera field of view. A stereo camera has been used in this thesis with TaraXL_ROS_Package to get the depth and integrate it in the vision system. In this section, first training of vision system is discussed followed by modification of the base network. The loss vs no of iterations determines the accuracy of the training of the model while mean Average Precision (mAP) determines average mean for each class.

3.3.1 FF-Net Training

The training of FF-Net involves collection of appropriate image data and sorting it to appropriate data-sets. To compile the training sets, images were collected using a variety of methods including collecting frames from old flight test data video footage, collecting live images from both indoor and outdoor flights and retrieving images of different kinds of drones from the world-wide web. By employing different kinds of data collection methods a data-set is constructed that captures vehicles under a variety of conditions with regards to color, type, size, illumination, occlusion, viewpoint, indoor and outdoor environments for Formation Flight Network (FF-Net) to identify aerial vehicles in all kinds of scenes and scenarios. These produced image data-sets were annotated and used to train the network. The collected image data-set is divided into two classes, one specific to leader aerial vehicle which is classified into Raft class and others in Drone class. Using these annotated data-sets FF-Net algorithm is trained using Darknet framework using large data-set to increase the robustness of the algorithm by detecting any multi-rotor regardless of orientation, angle and position in the camera Field of view (FOV). This network is trained using loss function defined to generate mean Average Precision (mAP). The training parameters are presented in

Figure.3.9. The FF-Net was trained on different types of drones with average loss of 0.4340 at 359,200th iterations.

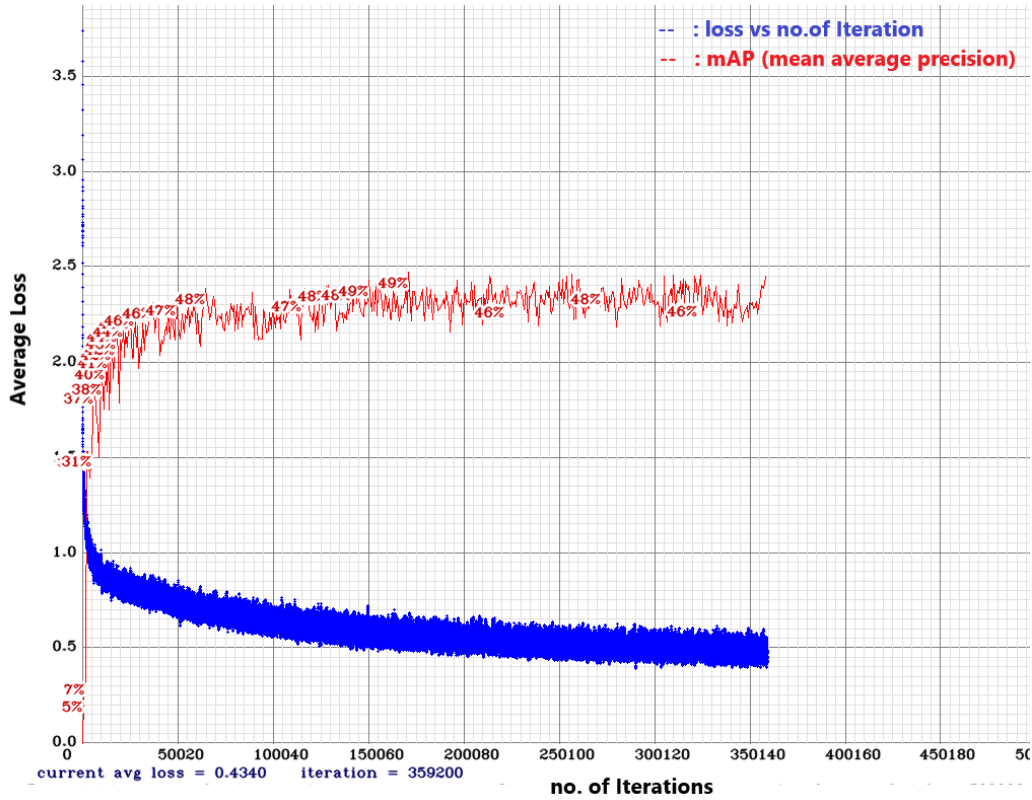


Figure 3.9 FF-Net Training Stats

3.3.2 FF-Net Depth inclusion

In this thesis, a pinhole camera geometry is considered to get the coordinates of 3D point in world frame using the obtained center pixel coordinates from detected bounding box. In addition to the geometry presented in Section 3.2, a stereo camera is considered as its geometry is similar to pinhole camera on each individual lense with depth as an additional feature. A stereo camera geometry similar to pinhole camera geometry is shown in Figure.3.10

The equations obtained in Section 3.2 are then rearranged to get the global frame coordinates of a point when pixel coordinates are known leading to

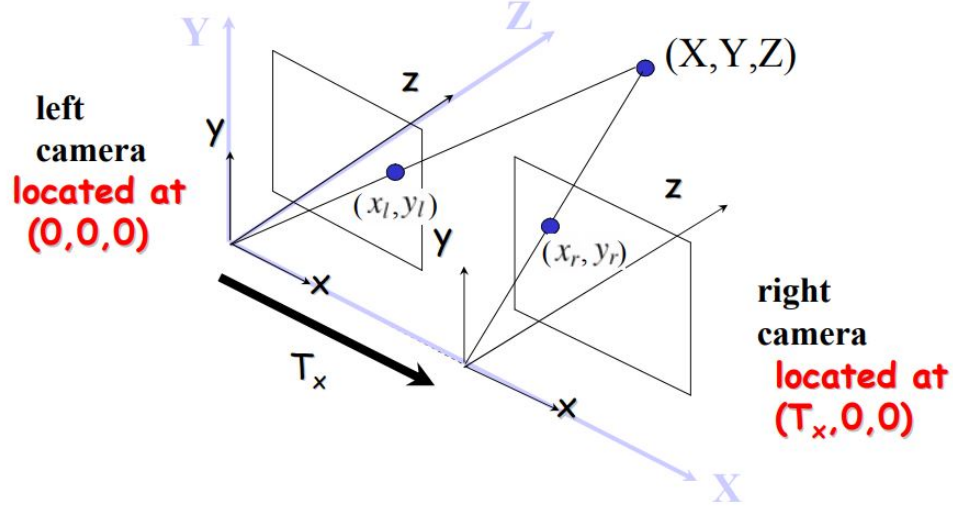


Figure 3.10 Camera geometry of Stereo camera

$$X = \frac{(u - c_x) * Z}{f_x} \quad (3.15)$$

$$Y = \frac{(v - c_y) * Z}{f_y} \quad (3.16)$$

where,

- u, v are the Center pixels of the detected object by FF-Net
- f_x, f_y are obtained from the camera calibration matrix of TaraXL stereo camera.
- c_x, c_y are also obtained from the camera calibration matrix of TaraXL stereo camera.
- X, Y, Z are coordinates of the detected object in world frame with respect to camera.

FF-Net use the predicted center coordinates b_x, b_y of the object obtained from the bounding boxes and a further process is to obtain relative position of the detected object in the camera frame X, Y, Z by adding the depth from the stereo camera. This is then converted into a global frame coordinates which are used by the follower vehicle to estimate and maintain the tracking position with a specific clearance. The results of the vision system are shown in the experimental results section.

Chapter Four

Formation Flight Control System Design

4.1 Formation Flight in Unmanned Vehicles

As it is proven to be beneficial to use unmanned vehicles in formations in the both aerial and space applications, more attention is now paid to various control problems associated with formation flight in unmanned vehicles. There are three methods commonly used in formation control:

1. **Leader - follower Structure:** In this structure one of the unmanned vehicles in formation is designated as leader and all other vehicles are considered as followers [Wu, Chen, and Yuan, 2017]. This structure is widely popular in controlling and managing formation flights missions as it is easy to track the position and orientation of the leader.
2. **Virtual Leader Formation Structure:** In this structure each individual vehicle receives same trajectory information of the virtual leader and the entire formation is considered as a single structure which makes it easy to define the formation behaviour.
3. **Behavioral Structure:** In this structure several behaviours of individual aerial vehicles such as target keeping, collision avoidance and formation are prescribed and to make control action of each individual unmanned vehicle in formation a weighted average of the control of each behaviour.

In this thesis, a formation flight control architecture based on Leader - Follower Structure was implemented as part of this effort based on the work presented in [Rice et al., 2016]. The formation flight (FF) geometry assumes decoupling lateral, vertical and forward clearances with respect to the leader vehicle reference frame. Two quad-rotors are flown in a leader follower configuration where the leader is programmed to follow a set of GPS waypoints. The follower vehicle then keeps formation with the leader in a desired formation geometry with forward f_c , lateral l_c and vertical v_c clearance. The orientation of the quad-rotor is derived from the difference in the yaw between the leader and follower. The horizontal geometry is defined by a forward distance, f and a lateral distance, l as shown in Figure.4.1. vertical geometry is then defined by the vertical distance error, h . Equation.4.1 describes the transformation on position errors between the follower and leader to a local reference frame that use forward and lateral clearances.

$$\begin{bmatrix} l \\ f \end{bmatrix} = \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} x_L - x \\ y_L - y \end{bmatrix} - \begin{bmatrix} l_c \\ f_c \end{bmatrix} \quad (4.1)$$

where,

l is the lateral distance between the leader and the follower, f is the forward distance between the leader and the follower.

δ is difference in yaw angles of leader and follower, defined as:

$$\delta = \psi_L - \psi \quad (4.2)$$

The vertical distance error, h , can be obtained using the vertical distance error relationship:

$$h = z_L - z - h_c \quad (4.3)$$

The rate of change of forward and lateral geometry with respect to time can be derived as:

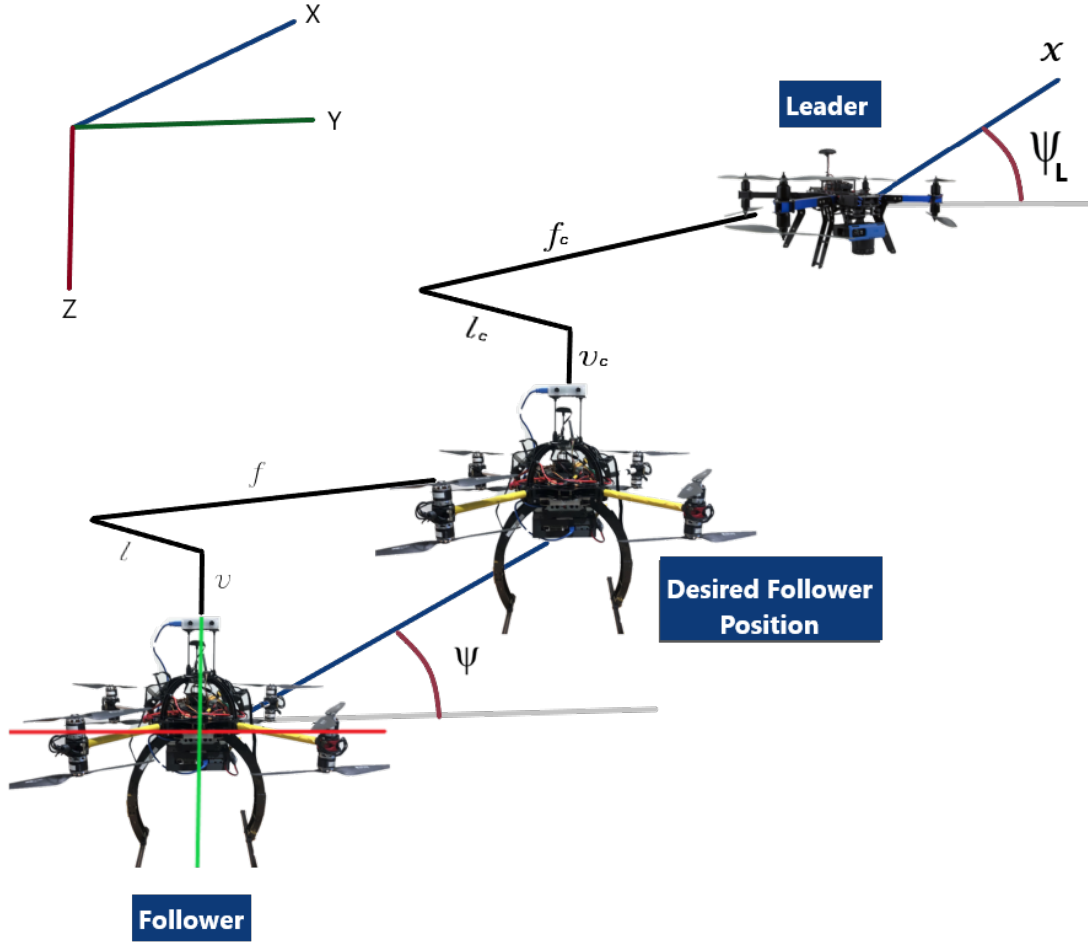


Figure 4.1 Formation flight geometry

$$\begin{bmatrix} \dot{l} \\ \dot{f} \end{bmatrix} = \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} + \dot{\delta} \begin{bmatrix} l_c \\ f_c \end{bmatrix} \quad (4.4)$$

where $\dot{\delta}$ is the rate of change in yaw angles, and δ is the difference between yaw angles from the leader and the follower vehicles. To obtain the required acceleration for the follower to maintain the formation geometry, the second derivative of lateral and forward clearances must be calculated:

$$\begin{aligned}
\begin{bmatrix} \ddot{l} \\ \ddot{f} \end{bmatrix} &= \begin{bmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} \\
+ \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} a_{x_L} - a_x \\ a_{y_L} - a_y \end{bmatrix} &+ \dot{\delta} \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} + \ddot{\delta} \begin{bmatrix} f + f_c \\ -l - l_c \end{bmatrix}
\end{aligned} \tag{4.5}$$

The accelerations of follower in x direction, a_x and y direction, a_y are denoted as

$$a_x = -\frac{u}{m}(\sin(\theta)) \tag{4.6}$$

$$a_y = \frac{u}{m}(\sin(\phi)) \tag{4.7}$$

where m is the mass and u is the total thrust output of the follower. A small angle assumption is taken into consideration and the above equation can be written as:

$$a_x = -\frac{u}{m}(\theta) \tag{4.8}$$

$$a_y = \frac{u}{m}(\phi) \tag{4.9}$$

Substituting this in equation [4.5] we get

$$\begin{aligned}
\begin{bmatrix} \ddot{l} \\ \ddot{f} \end{bmatrix} &= \begin{bmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} \\
+ \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} a_{x_L} - (-\frac{u}{m}(\theta)) \\ a_{y_L} - (\frac{u}{m}(\phi)) \end{bmatrix} &+ \dot{\delta} \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} + \ddot{\delta} \begin{bmatrix} f + f_c \\ -l - l_c \end{bmatrix}
\end{aligned} \tag{4.10}$$

$$\begin{aligned}
\begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} -\frac{u}{m}(\theta) \\ \frac{u}{m}(\phi) \end{bmatrix} &= \begin{bmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} \\
&+ \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} a_{x_L} \\ a_{y_L} \end{bmatrix} + \dot{\delta} \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} + \ddot{\delta} \begin{bmatrix} f + f_c \\ -l - l_c \end{bmatrix}
\end{aligned} \tag{4.11}$$

$$\begin{aligned}
\begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} -(\theta) \\ (\phi) \end{bmatrix} &= \frac{u}{m} \left(\begin{bmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} \right. \\
&+ \left. \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} a_{x_L} \\ a_{y_L} \end{bmatrix} + \dot{\delta} \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} + \ddot{\delta} \begin{bmatrix} f + f_c \\ -l - l_c \end{bmatrix} \right)
\end{aligned} \tag{4.12}$$

$$\begin{aligned}
\begin{bmatrix} -(\theta) \\ (\phi) \end{bmatrix} &= \frac{u}{m} \left(\begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix}^{-1} \begin{bmatrix} \cos(\delta) & \sin(\delta) \\ -\sin(\delta) & \cos(\delta) \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} \right. \\
&+ \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix}^{-1} \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix} \begin{bmatrix} a_{x_L} \\ a_{y_L} \end{bmatrix} \\
&+ \dot{\delta} \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix}^{-1} \begin{bmatrix} \dot{f} \\ -\dot{l} \end{bmatrix} \\
&+ \ddot{\delta} \begin{bmatrix} \sin(\delta) & -\cos(\delta) \\ \cos(\delta) & \sin(\delta) \end{bmatrix}^{-1} \begin{bmatrix} f + f_c \\ -l - l_c \end{bmatrix} \left. \right)
\end{aligned} \tag{4.13}$$

Finally, the desired pitch and roll angles can be calculated as:

$$\begin{aligned}
\begin{bmatrix} -\dot{\theta}_d \\ \dot{\phi}_d \end{bmatrix} &= \frac{m}{u} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} V_{x_L} - V_x \\ V_{y_L} - V_y \end{bmatrix} \dot{\delta} + \frac{m}{u} \begin{bmatrix} a_{x,L} \\ a_{y,L} \end{bmatrix} \\
&+ \frac{m}{u} \begin{bmatrix} \sin(\delta) & \cos(\delta) \\ -\cos(\delta) & \sin(\delta) \end{bmatrix} \left(\dot{\delta} \begin{bmatrix} \dot{f} \\ \dot{l} \end{bmatrix} + \ddot{\delta} \begin{bmatrix} \dot{f} + \dot{f}_c \\ -\dot{l} - \dot{l}_c \end{bmatrix} - \begin{bmatrix} \ddot{l}_d \\ \ddot{f}_d \end{bmatrix} \right)
\end{aligned} \tag{4.14}$$

where \ddot{l} and \ddot{f} can be obtained from an outer-loop controller:

$$\begin{bmatrix} \ddot{l} \\ \ddot{f} \end{bmatrix} = -K_{dist} \begin{bmatrix} l \\ f \end{bmatrix} - K_{spd} \begin{bmatrix} \dot{l} \\ \dot{f} \end{bmatrix} \quad (4.15)$$

where K_{dist} and K_{spd} are two gain parameters that can be designed using linear control approaches.

An inner-loop controller is also implemented to provide tracking capabilities required to minimize attitude errors:

$$\tau_{pitch} = K_{cmd}(\theta_d - \theta) + K_q(q) \quad (4.16)$$

$$\tau_{roll} = K_{cmd}(\phi_d - \phi) + K_p(p) \quad (4.17)$$

$$\tau_{yaw} = K_\delta(\delta) + K_r(r_l - r) \quad (4.18)$$

$$\tau_z = K_z(z_L - z) + K_{V_z}(V_{z_L} - V_z) + TH \quad (4.19)$$

where TH represents the minimum thrust required to maintain the position. The gains for inner-loop and outer-loop controllers are outlined in in Table 4.1

Inner-loop controller	pitch	$K_{cmd} = 1.5$	$K_q = 0.0315$
	roll	$K_{cmd} = 1.5$	$K_p = 0.0315$
	yaw	$K_\psi = 0.6$	$K_r = 0.03$
	thrust	$K_z = 0.8$	$K_{V_z} = 12.5$
Outer-loop controller	$K_{f_{dist}} = 2.5$ $K_{f_{spd}} = 5.85$ $K_{l_{dist}} = 2.5$ $K_{l_{spd}} = 5.85$		

Table 4.1 Controller gains

Finally, a control command w to each motor can be calculated for the follower by using the following control allocation with a combination of τ_{pitch} , τ_{roll} , τ_{yaw} , τ_z .

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \tau_{pitch} \\ \tau_{roll} \\ \tau_{yaw} \\ \tau_z \end{bmatrix} \quad (4.20)$$

4.2 Simulation in MATLAB/Simulink

Before deploying the Formation Flight control algorithm in the aerial vehicle it is validated in a MATLAB/Simulink simulation environment. Two simulated quad-rotors with high fidelity mathematical models are used in this simulation environment in which one acts as a leader aircraft and other as a follower with formation flight control laws implemented that are derived in Section.4.1. One of the simulated models of aerial vehicles is explained in subsections below followed by sensor models and different functional blocks. This section ends with explaining control law blocks used for formation flight.

4.2.1 Simulated Quad-rotor model

The simulation model of quad-rotor as shown in Figure.4.2 is a mathematical model of Leader aircraft. This model contains a simulated model of a quad-rotor in x8 configuration with sensor model block to simulate on-board sensors and control law block designated to give required control inputs to the vehicle to follow waypoints. This setup consists of two quad models. The leader quad-rotor which is following predefined waypoints and the follower quad-rotor which is following the leader with the derived equations.

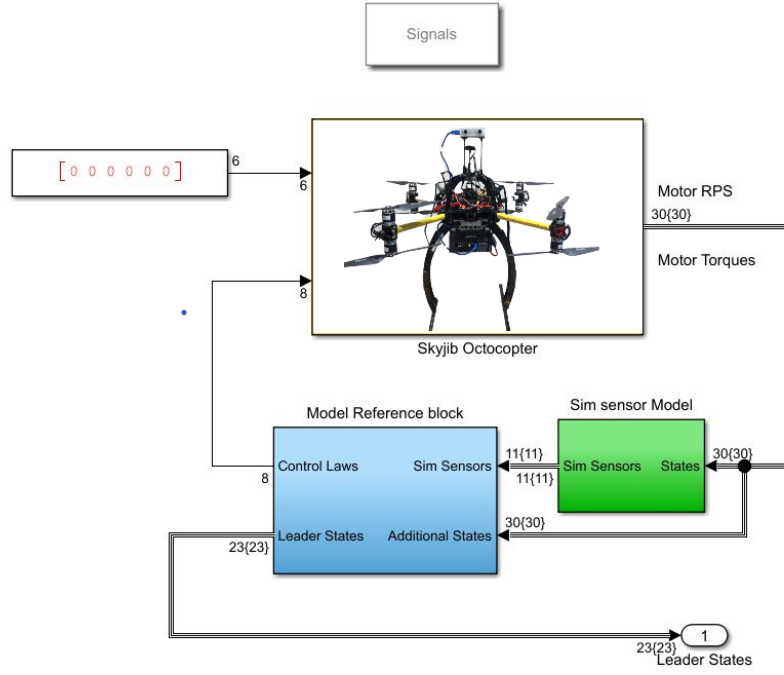


Figure 4.2 Simulink Skyjib x8 model

4.2.2 Simulation Environment

Figure.4.3 shows the simulation environment with both the aerial vehicles, Formation Flight control laws block, visualization block and variable clearance component. Top left "Skyjib leader" block acts as a leader vehicle and its states are given to Formation Flight control laws block which provides the follower vehicle (Skyjib Follower) with required control inputs to follow it.

Figure.4.4 shows the control allocation block along with inner and outer loop controllers. Inner loop controller contains PID controllers and outer loop controller contains Formation Flight geometry as shown below in Figure.4.5

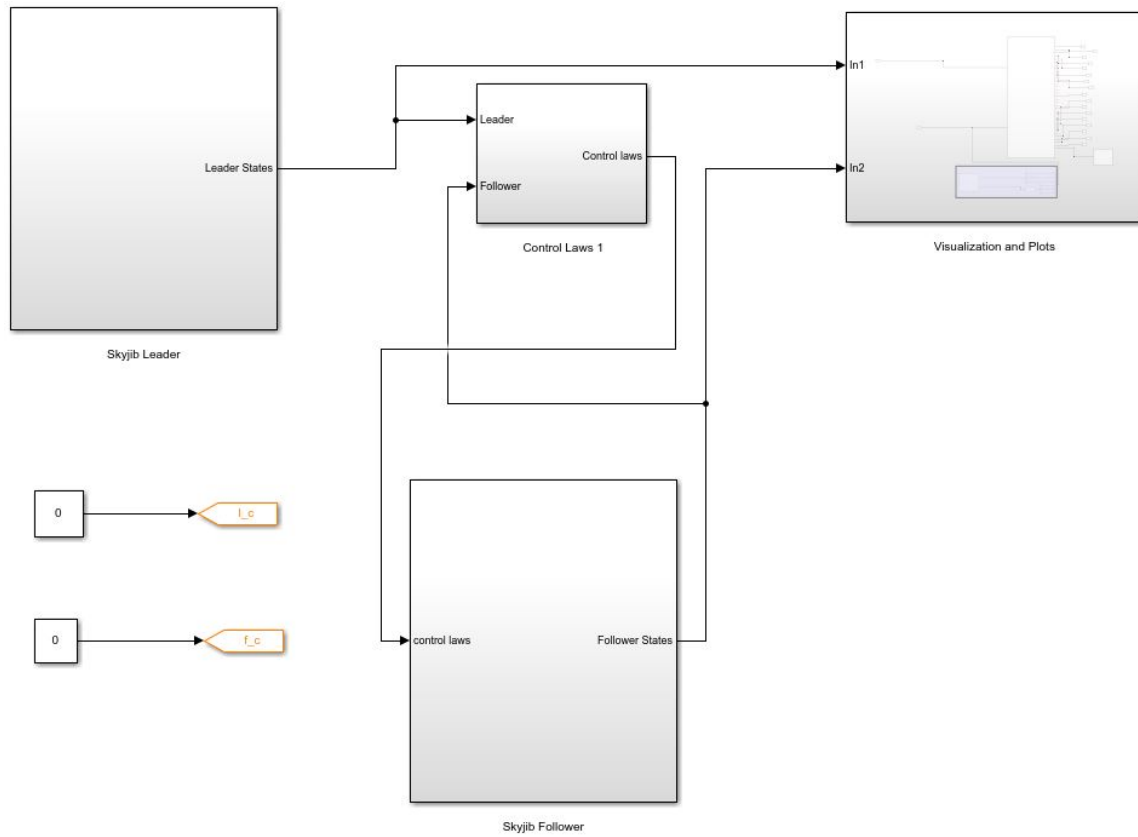


Figure 4.3 Formation Flight model with both the vehicles

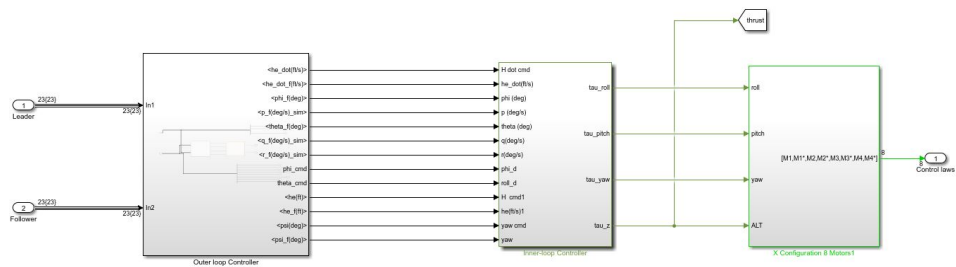


Figure 4.4 Control Allocation

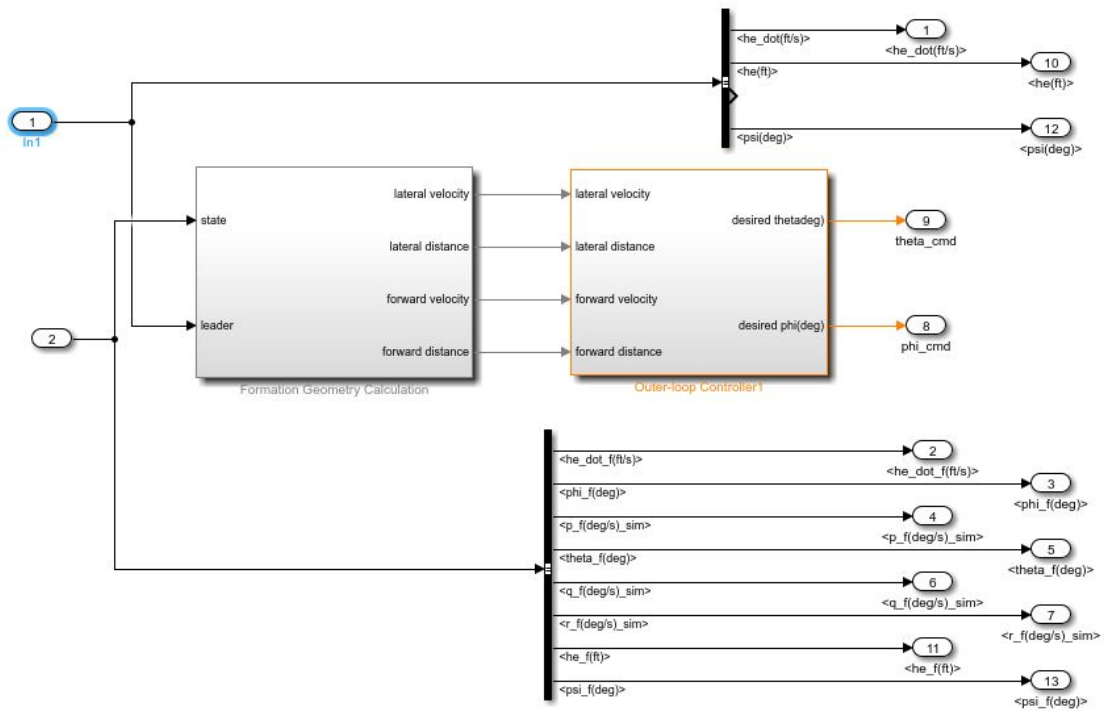


Figure 4.5 Formation flight block in Outer loop controller

Chapter Five

Communication Architecture

Communication between systems is a key feature addressed in this chapter. Robot Operating System (ROS) is used in this research to tackle the flow between all the subsystems. A brief explanation of ROS is given in the following section followed by the subsystems and packages used in the Research.

5.1 Robot Operating System (ROS)

As the growth of robots increased drastically in 21st century the complexity of the robots increased along with it. Robots operate under real-world, real-time conditions where actuators and sensors are read and controlled. Various types of robots have various types of sensors, actuators and hardware which uses different architectures making the code hard to interact between each other. One other issue working with different types of robots is the size of the code as it contains a large stack containing from driver-level software to localisation, flight codes and controls which cannot be handled by a single researcher [Kramer and Scheutz, 2007]. To overcome these challenges, facilitate research in autonomous robotics and make an easily intractable robot environment a lot of architectures were created from time to time that support various aspects of the agent development process, ranging from the design of an agent architecture, to its implementation on robot hardware, to executing it on the robot

[Dattalo, 2018]. These frameworks were designed for a particular purpose.

One of such robot environments is ROS, which is chosen in this research as it contains all the required libraries to solve the problem of communication required. ROS is an open source set of libraries and tools that help to write application specific or robot specific software which can interface between processes [Quigley et al., 2009]. It is a flexible framework containing a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms and provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management. ROS is a Meta Operating system that provides a structured communications layer and assumes there is an underlying operating system of a heterogeneous compute cluster that will assist it in carrying base tasks. ROS as a Meta Operating system cannot be classified as a simple framework as it provides a huge amount of functionalities of operating system and a cluster of libraries but not fully. ROS is based on five philosophical goals:

1. Peer-to-peer
2. Tools-based
3. Multi-lingual
4. Thin
5. Free and Open-Source

As ROS is freely available to a large population due to its open nature it also needs an operating system that is open source so the operating system and ROS can be modified as per the requirements of the application. Hence, it is initially created to run ROS on Linux particularly Debian and Ubuntu. The main feature of ROS is the way it runs in a system and the way it communicates.

ROS starts with the ROS Master. The Master provides a way to connect a network of processes (nodes) with a central hub and allows all other ROS pieces of software to find and talk to each other where every node is responsible for one task. Nodes communicate with each other using messages passing via logical channels called topics. Each node can send or get data from the other node using the publish/subscribe model. By providing service on request, or by using publisher or subscriber connections a network is created and communicates via predefined message types. Some of the features are mentioned below:

1. Any modification to the interface can be carried-out on the go.
2. It is easy to connect different modules or packages from different software developers by just implementing right message connectors to the master.
3. It provides Inter Process Communication (IPC) and Remote Procedure Call (RPC) systems.
4. It allows parallel solving of many problems and also mixes multiple outputs of multiple components into one.
5. It acts as a cross platform for various programming languages.

Other software interfaces like Mobile Robot Programming Toolkit (MRPT), Carnegie Mellon Robot Navigation Toolkit (CARMEN), Microsoft Robotics Developer Studio (RDS), Lightweight Communications and Marshalling (LCM) also provide some of the features, but not all. One of the important feature of ROS is run-time computational graph which is a pear-to-pear network of ROS processes that are loosely coupled using the ROS communication infrastructure [Quigley et al., 2009]. The fundamental communication graph concepts are nodes, master, parameter server, messages, services, topic, and bags.

An outline of ROS packages and nodes are shown in Figure.5.1. The most relevant components integrated in this application are described as follows:

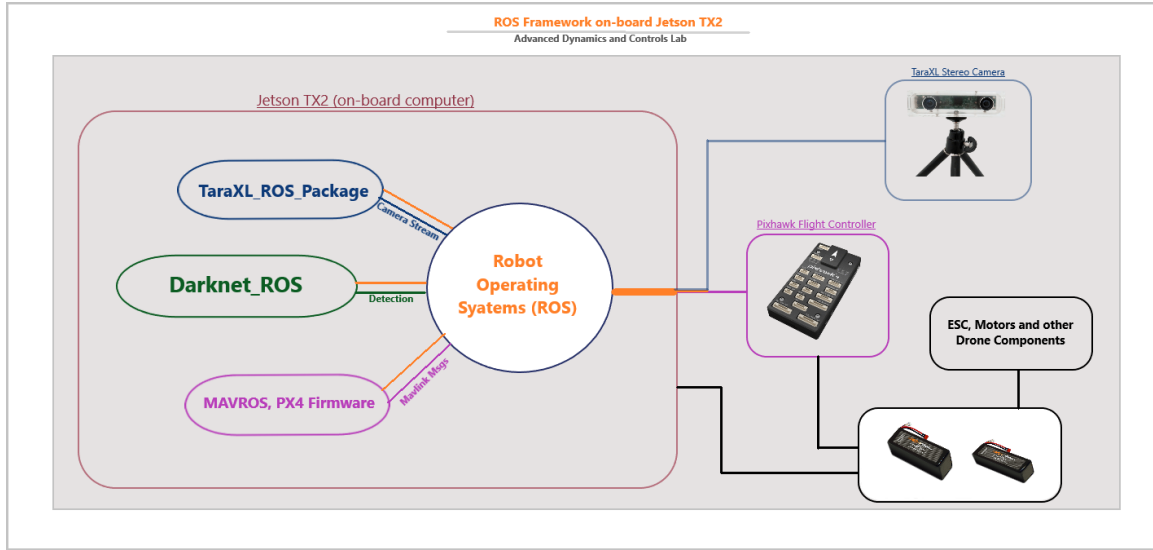


Figure 5.1 ROS Framework on-board Jetson TX2

5.1.1 Stereo Camera

The on-board camera TaraXL is interfaced with a Jetson computer using a software suit TaraXL SDK developed by e-con systems systems, 2019. However, a TaraXL_ROS package is used to interface the camera with vision system and python APIs [taraxl_ros]. Some of the published topics of the package are rectified images, depth image, disparity image, pointcloud, Inertial Measurement Unit (IMU) inclination and raw data from the camera:

- /taraxl/left/image_rect - Rectified left image
- /taraxl/right/image_rect - Rectified right image
- /taraxl/left/image_raw - Unrectified left image
- /taraxl/right/image_raw - Unrectified right image
- /taraxl/stereo/disparity/image - Disparity image
- /taraxl/depth/image - Depth image
- /taraxl/stereo/pointcloud - pointcloud

- /taraxl/imu/data_raw - Raw IMU data - linear acceleration and angular velocity
- /taraxl/imu/inclination - IMU inclination data w.r.t 3 axes x,y and z

5.1.2 MAVROS and PX4

MAVROS is a MAVlink [Koubaa et al., 2019] extendable communication node for ROS that can convert between ROS topics and MAVLink messages. This allows vehicles to communicate with Ground Control Station using mavlink protocol [Ermakov, 2017]. MAVROS is used to communicate between a low cost autopilot PX4 flight controller and Jetson TX2 onboard vision computer. MAVROS also handles the frame translations Aerospace north-east-down (NED) from Flight Control Unit (FCU) to ROS east, north, up (ENU) frames which is simply carried by applying a rotation of 180 deg about ROLL (X) axis and for local translation by applying 180 deg about ROLL (X) and 90 deg about YAW (Z) axes. Some of feature of MAVROS package are:

1. Connection to all MAVLink supported devices e.g. PX4, ArduPilot.
2. Communication with flight controller via serial port, UDP or TCP.
3. Modification of Parameters on the fly.
4. Internal proxy for Ground Control Station using serial, UDP, TCP communication protocols.
5. Add new Waypoints to the mission.
6. PX4Flow support (by mavros_extras).
7. OFFBOARD mode support.
8. Geographic coordinates conversions.

The nodes used to communicate between PX4 and Jetson TX2 are pose, altitude and battery.

5.1.3 Darknet_ROS

Darknet_ROS is an integrated package that supports the interface between Darknet library and ROS [jelonic, 2018]. It is developed for object detection on GPU and CPU. This package has been modified to include depth output from TaraXL camera and to get the X, Y, Z relative coordinates of the target. This package publishes number of objects detected, bounding box of the detected target and the detected image. Original published topics are:

- object_detector - Publishes the number of detected objects.
- bounding_boxes - Publishes an array of bounding boxes that gives information of the position and size of the bounding box in pixel coordinates.
- detection_image - Publishes an image of the detection image including the bounding boxes.

This packages is modified to give one more published topic:

- object_coord - Publishes the object coordinates relative to the camera position by using `"/taraxl/left/image_rect"` and `"/taraxl/depth/image"` topics from TaraXL ros package.

5.1.4 Vicon system

For this research effort, a Vicon indoor facility was available at the Advanced Dynamic and Control Lab at ERAU. The facility, used to validate vision data, is equipped with a Virtual-Reality Peripheral Network (VRPN) that broadcasts the tracking information to any computer connected to the wireless network. Therefore, Vicon tracking position of the vehicle can be simultaneously compared with the developed vision-based estimation algorithm. The on-board computer Jetson TX2 uses Vrpn_client_ROS packages [ros-drivers, 2017] to subscribe to the published topics such as tracker pose, acceleration and orientation.

Chapter Six

Simulation Environment For Vision System In AirSim

6.1 UAS Simulator

Experimenting with unmanned systems are expensive and not safe. So, all the algorithm, experimental setups and the performance of unmanned systems are tested and analyzed in a simulated environment with Software In The Loop (SITL). Which gives a test-bed to test and operate Planes, Rovers and Multi-rotors without any hardware built with autopilot code and other sensors using C, C++ or Python based language. SITL simulates the vehicle to run the autopilot and acts a practical tool to avoid the crashes and misbehaving in-flight on a real system and is self-contained to avoid loss of simulated data. Using a simulator avoids hazardous situations saving the cost of equipment and provide a real-time estimation of the system [Shah et al., 2018]. Usually any simulator can be connected using User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) communication protocols. An ideal simulator should provide mission specific modes and also a combination of operating modes such as:

1. System behavioural analysis and predicting.
2. Artificial intelligence and machine learning training support.

3. Allow testing and evaluation of different control algorithms.
4. Hardware In The Loop, Software In The Loop and Human In The Loop simulations.

The capabilities of a simulator must allow a customized [Driss et al., 2018], flexible and selected formulation of scenarios to include a variety of

1. various kinds of missions.
2. Testing and evaluating different objectives.
3. Different levels of agent and system autonomy.
4. Different levels of risk and event occurrence probability.
5. Different levels of system intelligence.
6. Various kinds of payload and communication for system analysis.

Some of the important feature of UAS simulator are:

1. Create a real-time environment to fly in.
2. Provide support to various autopilots like PX4, Aedupilot.
3. Allow a multitude of sensors like cameras, Lidar, GPS and other sensors.
4. Provide different kinds of configuration of vehicles i.e. quad-copter, octa-copter.
5. Simulate physical aspects like wind speeds, turbulence, air density, clouds, precipitation and other fluid mechanics constraints.
6. Support multiple vehicles with individual controls and dynamics.
7. Able to interface with various languages.
8. Compatible with different platforms Like windows, Linux, android and mac.

9. Simulate UAS physics in order to provide accurate represented data.
10. Supports Motion Capture (MOCAP) to simulate UAVs motion planning.
11. Available open source or easily accessible.

6.1.1 Comparison of Various UAS Simulator

There is no one go to simulator to give all the functions as most of them are designed for different purposes. Some of the simulators provide Hardware In The Loop (HITL) simulations to integrate radio controller and autopilots in real-time. Figure.6.1 shows basic architecture of most simulators. Some of the simulators available are XPlane, Flightgear, Gazebo, AirSim, JMAVSim, UE4Sim and Drone code.

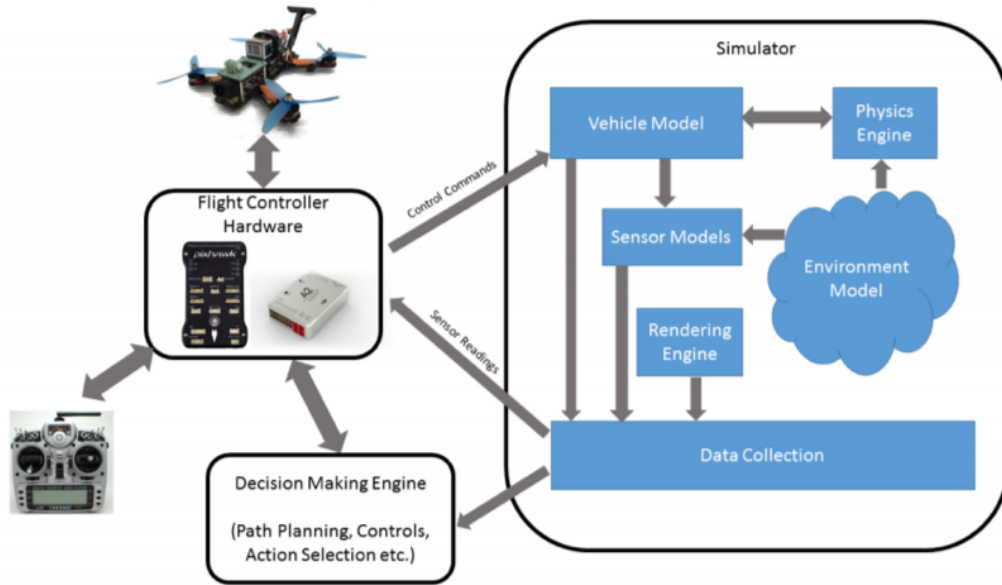


Figure 6.1 Basic architecture of simulator

Some of the comparisons between commonly used UAS simulators are given in Table.6.1.

	Gazebo	AirSim	JMAVSim
Vehicle interface	Multicopter and Any robots	Multicopters	Multicopters
Availability	Open-Source	Open-Source	Open-Source
MAVLink	Yes	yes	yes
multiple Platforms	Linux	Windows, Linux and mac	windows
Autopilot	Ardupilot, PX4	Any MAVLink compatible device	
Motion Capture	no	yes	no
SITL - HITL	yes	yes	yes
Obstacles	yes	yes	no
ROS interface	yes	yes	yes
Ease of deployment	high	medium	high

Table 6.1 Comparison between different simulators

6.2 AirSim

In this thesis, to replicate realistic environment and to support the initial design and tuning of the vision-based tracking algorithm, a simulation environment was implemented using Microsoft Aerial Informatics and Robotics Simulation (Airsim) software. Airsim is integrated with Unreal Engine (UE) tool which is an open-source tool for simulating vehicle translational and rotational motion using modern high-quality engine and realistic physics library. AirSim is developed by Microsoft to test and develop machine learning algorithms in February 2017. AirSim provides interface with MAVLink to obtain live data and communication with the simulated vehicle and is possible to run SITL and HITL which is shown in Figure.6.2.

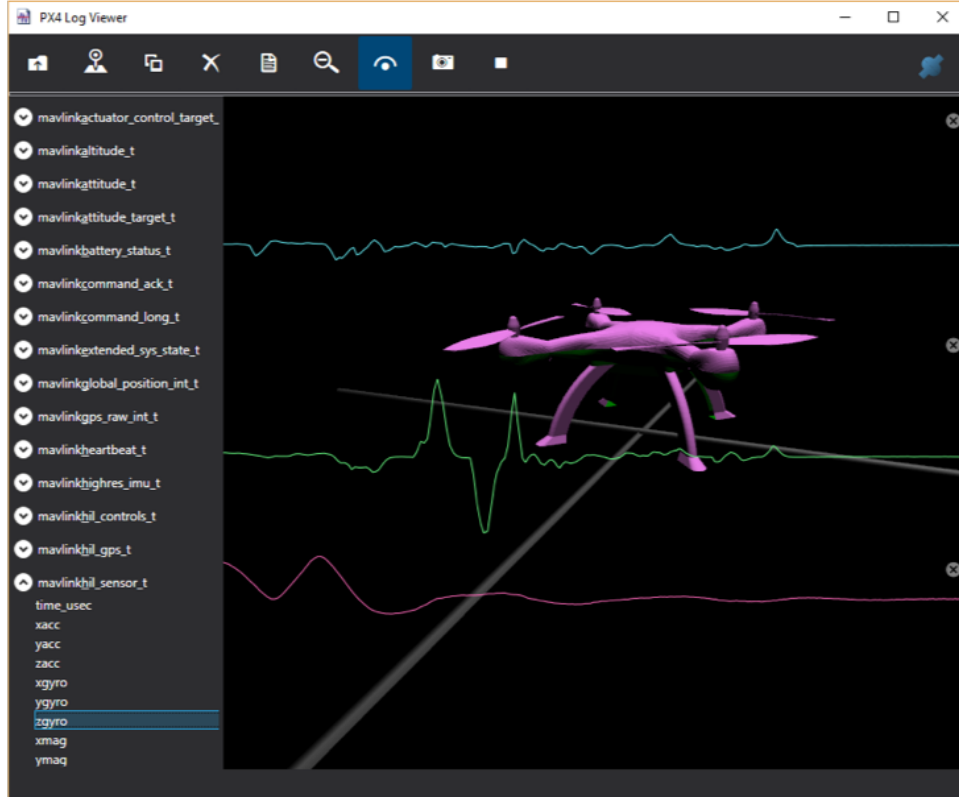


Figure 6.2 MAVLink analysis of a quad-copter

The simulator is built on game development tool Unreal Engine 4 (UE4) as a base to render the simulated environment more photo-realistic. AirSim is able to interface with Ardupilot and Pixhawk firmware using MAVLink communication protocol and provides Application programming interfaces (APIs) for both python and C/C++ programming languages. It provides monocular and stereo camera along with other sensors like Lidar, GPS and accelerometer along with support for multiple vehicles with individual controls. Figure.6.3 shows the user interface with sensor output display. AirSim also supports ROS communication along with integration of flight sensors such as IMU, LiDAR, GPS, Barometer, among others [Shah et al., 2018].

AirSim is used as initial stage to simulate multiple vehicles while using the stereo camera data for detection of a leader vehicle within formation flight. Two quad-rotor vehicles are simulated while the follower transmit live images and depth maps through ROS as shown in

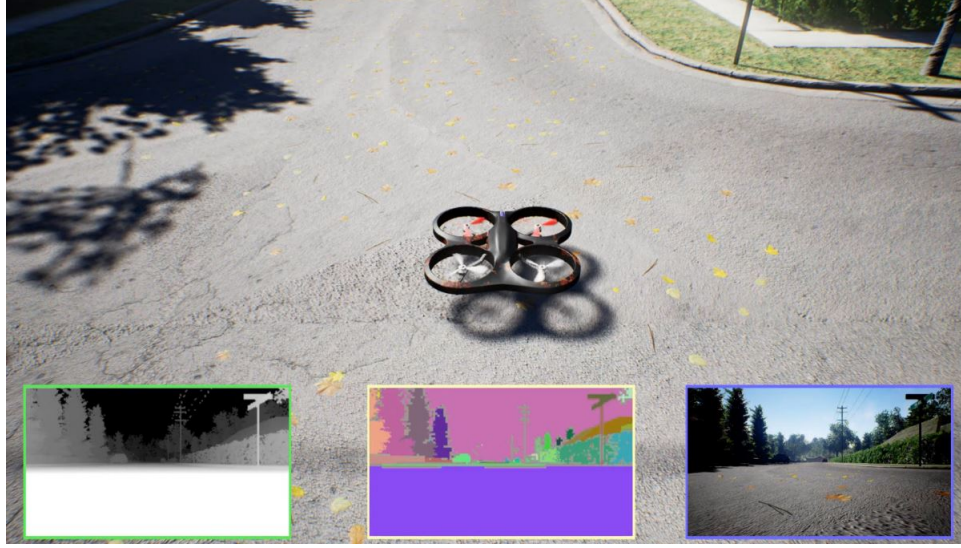


Figure 6.3 User display showing sensor output

Figure.6.4.



Figure 6.4 Two quads flying in AirSim

A python code was created to receive this images and send them to the tracking algorithm to detect the leader along with the estimation of position and orientation relative to the follower. AirSim do not yet support any customised low level controller to be implemented on the vehicles and this development is still in research stages. However, in this paper

AirSim was used to test and validate the performance and functioning of the vision system in a real-time environment. The system configuration running Airsim is given in Table.6.2

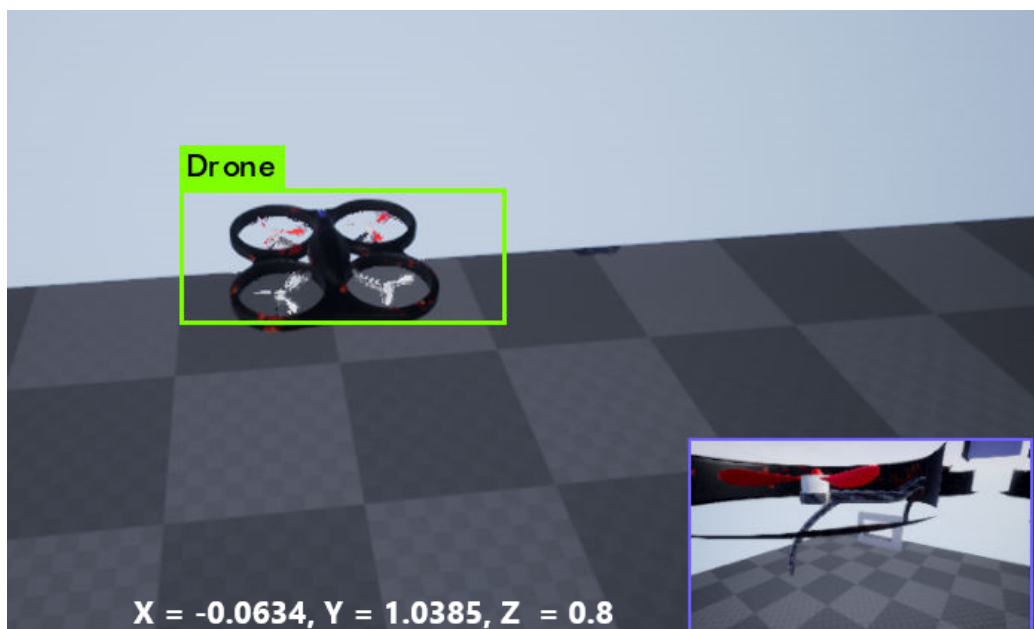


Figure 6.5 FF-Net detection in AirSim

Airsim Computer Configuration	
Operating system	Ubuntu 18.04 LTS
RAM Memory	128 GB
Processor	Intel Xeon(R) Gold 6148 CPU @ 2.40GHZ X 80
Graphics	P4000/PCLe/SSE2

Table 6.2 System Configuration

Chapter Seven

UAS Research Testbed and Facilities

This chapter explains the UAS research test-bed and the testing facilities used to test and validate the algorithms. The first section of this chapter explains in detail the research vehicles used and hardware components such as on-board computer, flight controller, camera on-board, power and propulsion system along with the firmware used on-board and software used in the ground control station. This is followed by a comprehensive explanation of the indoor and outdoor flight testing facilities used to for flight tests.

7.1 Research Vehicles

Two commercial aerial platforms, a 3DR and a SkyJib quadrotors, are used as research testbed vehicles shown in Figure.7.1. Within the formation flight configuration, SkyJib platform is considered as a follower as it allows higher payload for easier integration of vision system and 3DR is considered as leader. Table.7.1 gives the basic configurations of both the vehicles. Below sections explain these vehicles in detail.



Figure 7.1 Skyjib and 3DR with all Hardware Components

3DR Quadrotor and SkyJib Quadrotor		
Items	Dimensions (mm)	Dimensions (mm)
Propeller arms	240 mm	420 mm
Brushless motors	45 mm (height) , 28 mm (dia.)	56 mm (height) , 45 mm (dia.)
Propellers	10 x 4.7	15 x 5
Battery	150 x 47 x 30 mm	140 x 90 x 43 mm
Weight	3 kg	8.07 kg

Table 7.1 Quadrotors Dimensions and Mass Properties

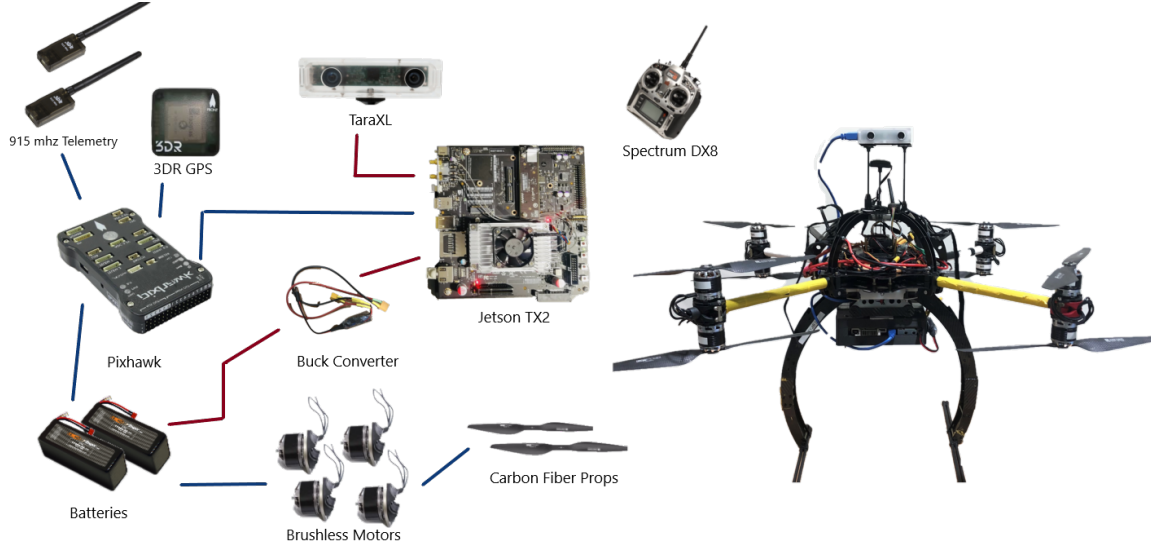


Figure 7.2 Skyjib x8 with Jetsin TX2 on-board

7.2 On-board Flight Hardware

7.2.1 Pixhawk(PX4) Autopilot Flight Controller

The flight controller used in this research is a low cost Pixhawk 1 which is shown in Figure.7.3. PX4 autopilot is an open-source autopilot system oriented toward inexpensive autonomous aircraft. This was initially developed in Computer Vision and Geometry Lab of ETH Zurich (Swiss Federal Institute of Technology) in 2009. Now it is manufactured and marketed by 3D Robotics [sUAS, 2013]. The flexible PX4 middle-ware running on the NuttX Real-Time Operating System brings multi-threading and the convenience of a Unix / Linux like programming environment to the open source autopilot domain, while the custom PX4 driver layer ensures tight timing. These facilities and additional headroom on RAM and flash will allow Pixhawk the addition of completely new functionalities like programmatic scripting of autopilot operations.

Some Features of Pixhawk are given below followed by specification and interface in Table.7.2.

1. 32 bit ARM Cortex M4 Processor running NuttX RTOS



Figure 7.3 Pixhawk

2. 14 PWM / Servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
3. Abundant connectivity options for additional peripherals (UART, I2C, CAN)
4. Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply
5. Backup system integrates mixing, providing consistent autopilot and manual
6. External safety switch, Multicolor LED main visual indicator, High-power, multi-tone piezo audio indicator, microSD card for long-time high-rate logging.

PX4 v1.8.2

The Pixhawk firmware used for testing in this research is a stable version 1.8.2 . Updates and main features of PX4 v1.8.2 are mentioned below:

1. Fusion of Visual Inertial Odometry in EKF2 (video)
2. Interface for external Obstacle Avoidance systems (video)

Pixhawk Specification	
Specifications	32bit STM32F427 Cortex M4 core with FPU, 168 MHz, 256 KB RAM, 2 MB Flash, 32 bit STM32F103 failsafe co-processor, ST Micro L3GD20H 16 bit gyroscope, ST Micro LSM303D 14 bit accelerometer / magnetometer, MEAS MS5611 barometer
Interfaces	5x UART (serial ports), one high-power capable, 2x with HW flow control, 2xCAN, Spektrum DSM / DSM2 / DSM-X® Satellite compatible input, Futaba S.BUS® compatible input and output, PPM sum signal, RSSI (PWM or voltage) input, I2C®, SPI, 3.3 and 6.6V ADC inputs, External microUSB port
Dimensions	Weight: 38g (1.31oz), Width: 50mm (1.96), Thickness: 15.5mm (.613), Length: 81.5mm (3.21)

Table 7.2 Pixhawk Specifications sUAS, 2013

3. Significantly improved performance on racing drones (users need to reconfigure, [link](#))
 - (a) Improved filtering and reduced control latency
 - (b) Added Airmode
4. Improved flight performance on VTOL (Tiltrotors, Tailsitters)
5. Support for building natively on Windows ([link](#))
6. Significant EKF2 improvements
 - (a) Hardening of the estimator for situations where GPS accuracy is limited
 - (b) improved sensor selection logic enabling simultaneous use of optical flow and GPS
 - (c) Added the EKF2_MAG_TYPE parameter for environments with high magnetic interferences

7. Wind Estimator
8. Support for structure scanning
9. High Latency telemetry support (Iridium)
10. Precision landing framework (including IRLock driver)

The configuration of the airframe used in the research is shown in Figure.7.4

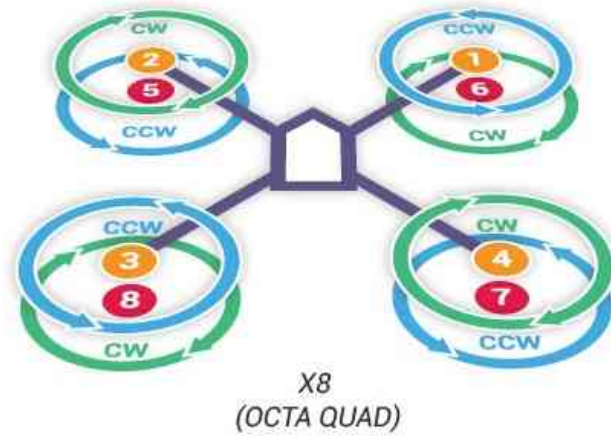


Figure 7.4 Pixhawk Motor configuration for both Vehicles

7.2.2 GPS Receiver Module with Digital Compass

The UBlox GPS + Compass module is the most commonly used GPS for ArduPilot compatible autopilots. The external UBLOX GPS includes the HMC5883L digital compass, convenient method of mounting the compass away from sources of interference that may be present in the confines of the vehicle Ardupilot, 2019c. It features active circuitry for the ceramic patch antenna, rechargeable backup battery for warm starts, and I2C EEPROM for configuration storage. Features and specifications of GPS used in the research is shown in Table.7.3



Figure 7.5 GPS Receiver Module with Digital Compass

UBLOX GPS Specification	
Features and Specifications	<ul style="list-style-type: none"> ublox LEA-6H module 5 Hz update rate 25 x 25 x 4 mm ceramic patch antenna LNA and SAW filter Rechargeable 3V lithium backup battery Low noise 3.3V regulator I2C EEPROM for configuration storage Power and fix indicator LEDs APM compatible 6-pin DF13 connector Exposed RX, TX, 5V and GND pad 38 x 38 x 8.5 mm total size, 16.8 grams.

Table 7.3 UBLOX GPS Specifications

7.2.3 915 mhz Telemetry

A SiK Telemetry Radio shown in Figure.7.6 is used to connect and monitor the aerial vehicle with a ground control station in a laptop. This telemetry is a small, light and inexpensive open source radio platform that typically allows ranges of better than 300m. The radio uses open source firmware which has been specially designed to work well with MAVLink

packets and to be integrated with the Mission Planner, Copter, Rover, and Plane. Some specifications are mentioned in Table.7.4.

Telemetry Specification	
Antenna connectors	RP-SMA connector
Output Power	100mW (20dBm), adjustable between 1-20dBm
Sensitivity	117dBm sensitivity
Interface	Standard TTL UART
Connection status	LED indicators Demonstrated

Table 7.4 Telemetry Specifications Ardupilot, 2019b



Figure 7.6 915mhz Telemetry

7.2.4 Batteries

A MAXAMP 6S 22.2 volts LiPo batteries as shown in Figure.7.7 are used to power the on-board Jetson TX2 computer, Flight controller (Pixhawk), Camera and other drone hardware peripherals on the Skyjib X8 configuration and a HRB 5000mah 50c 11.1V LiPo batteries are used for 3DR quad-rotor is shown in Figure.7.8. Both are explained below.

In MAXAMP 8000mah 6S 22.2 volts LiPo battery the power load is split between two



Figure 7.7 MAXAMP 6S 22.2 volts LiPo batteries used with Skyjib X8

cells(cores) instead of one. Each cell has the thickest and lowest resistance tabs available. The 150c burst rate gives enough power for acceleration.



Figure 7.8 HRB 5000mah 50c 11.1V lipo battery used with 3DR Aerial Vehicle

HRB 5000mah 50c 11.1V lipo battery shown in Figure.7.8 offers very high power and very long run time for RC models. It enhances driving experience no matter what type of driving. These batteries packs are made with superior Lithium Polymer raw material and advanced stacking technology enables a single cell of capacity to reach 5000mah.

7.2.5 Spectrum DX8 Transmitter and DSMX Remote Receiver

A Spectrum DX8 transmitter and DSMX remote receiver is used to communicate and pass the commands to on-board flight controller. This transmitter is capable of using 8 channels which are custom programmed for the flight tests performed in the research. The user input from transmitter is transmitted to the on-board receiver which is connected to flight controller executing commands.

7.3 Jetson TX2 On-board Computer

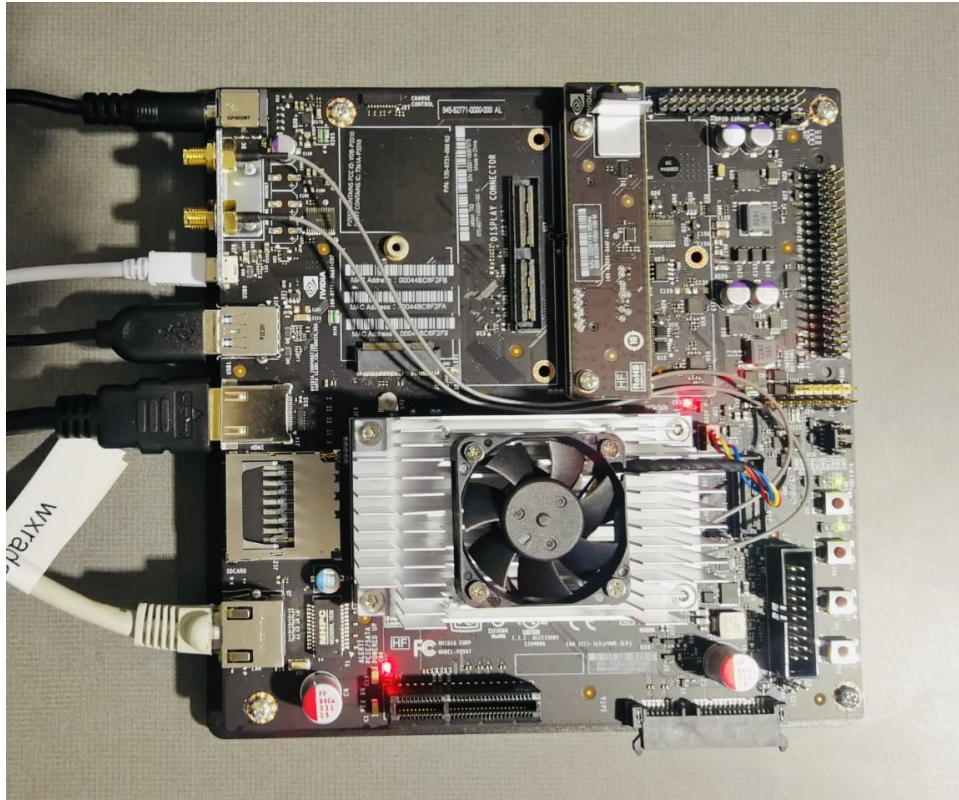


Figure 7.9 Jetson TX2 development Board

All the image processing and machine learning is performed in real time by the embedded computing device Jetson TX2 (Fig.7.9). Jetson TX2 features an integrated 256-core NVIDIA Pascal GPU, a hex-core ARMv8 64-bit CPU complex, and 8GB of LPDDR4 memory with a

128-bit interface [NVIDIA, 2019]. The CPU complex combines a dual-core NVIDIA Denver 2 alongside a quad-core ARM Cortex-A57. Jetson uses NVIDIA cuDNN and TensorRT libraries with support for Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and online reinforcement learning accelerates state-of-the-art deep neural network (DNN). Robots and drones can use Jetson autopilot integration to control them to operate safely in realworld and perceive it. Jetson TX2 provides processing and memory features to perform complex tasks by using a NVIDIA Pascal™ Architecture GPU, 2 Denver 64-bit CPUs + Quad-Core A57 processor [Jetson, 2019]. main characteristics of Jetson computer are summarized in Table.7.5.

Mode	Mode Name	Denver 2	Frequency	ARM A57	Frequency	GPU Frequency
0	Max-N	2	2.0 GHz	4	2.0 GHz	1.30 GHz
1	Max-Q	0		4	1.2 GHz	.085 GHz
2	Max-P Core-All	2	1.4 GHz	4	1.4 GHz	1.12 GHz
3	Max-P ARM	0		4	2.0 GHz	1.12 GHz
4	Max-P Denver	1	2.0 GHz	1	2.0 GHz	1.12 GHz

Table 7.5 NVIDIA Jetson Operating Modes

Complete specifications of jetson are presented in 7.6. A case shown in Figure.7.9 is 3D printed in the ADCL to integrate it into the drone gimble system and also help as a protection case (Figure.7.10).

7.4 TaraXL Stereo Camera

A TaraXL USB stereo camera was integrated to the NVIDIA Jetson TX2 and NVIDIA GPU cards. This 3D stereo camera is based on MT9V024 stereo sensor from ON Semiconductor. TaraXL is bundled with a proprietary CUDA® accelerated Stereo SDK called Tara XL SDK that runs on the GPU of NVIDIA® Tegra processors which has a capability to provide 3D

NVIDIA Jetson TX2 Specifications	
CPU	ARM Cortex-A57 (quad-core) @ 2GHz + NVIDIA Denver2 (dual-core) @ 2GHz
GPU	256-core Pascal @ 1300MHz
Memory	8GB 128-bit LPDDR4 @ 1866Mhz 59.7 GB/s
Storage	32GB eMMC 5.1
Encoder	4Kp60, (3X) 4Kp30, (8X) 1080p30
Decoder	(2X) 4Kp60
Camera	12 lanes MIPI CSI-2 2.5 Gb/sec per lane 1400 megapixels/sec ISP
Display	2X HDMI 2.0 / DP 1.2 / eDP 1.2 2X MIPI DSI
Wireless	802.11a/b/g/n/ac 22 867Mbps Bluetooth 4.1
Ethernet	10/100/1000 BASE-T Ethernet
USB	USB 3.0 + USB 2.0
PCLe	Gen 2 1X4 + 1X1 or 2X1 + 1X2
CAN	Dual CAN bus controller
Misc I/O	UART, SPI, I2C, I2S, GPIOs
Socket	400-pin Samtec board-to-board connector, 50X87mm
Thermals	-25C to 80C
Power	7.5 W

Table 7.6 NVIDIA TX2 Jetson Specifications

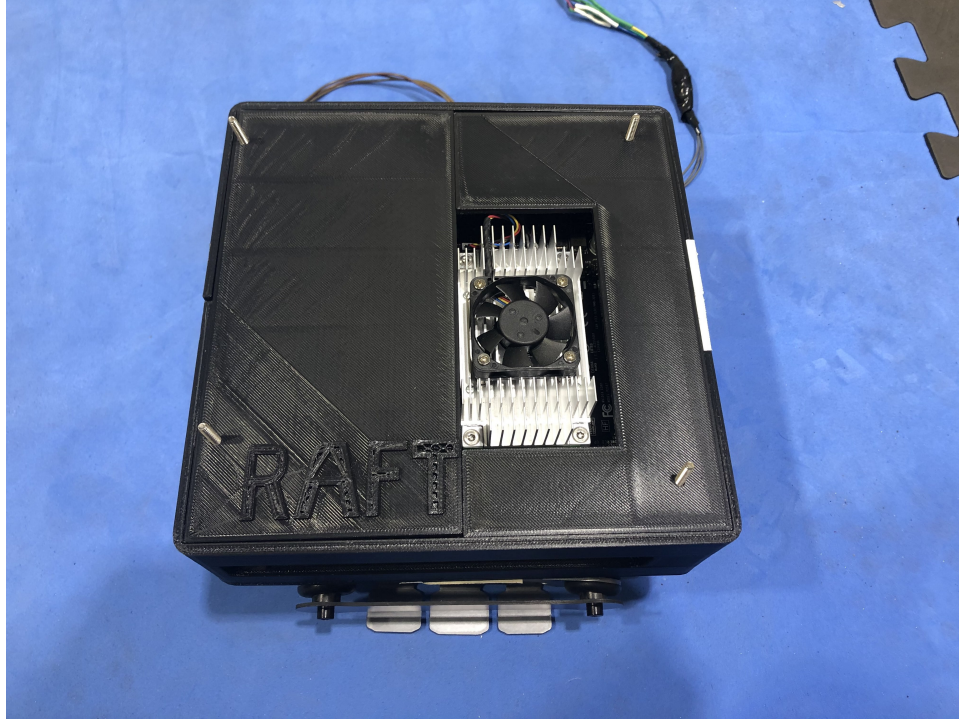


Figure 7.10 Jetson TX2 development Board with 3D Printed Case

Depth map for 752 X 480 @ 50 fps. TaraXL with a form factor 100 x 30 x 35mm consists of two OnSemi's 1/3 inch MT9V024 CMOS image sensors separated by an 'inter-ocular distance' or 'base line' of 60 mm [systems, 2019].

Taraxl is connected to Jetson TX2 using USB 3.0 interface to stream uncompressed Stereo WVGA format (1504*480) at 60 fps which are processed by TaraXL SDK to generate the depth map of the scene. Jetson TX2 uses TaraXL SDK and TaraXL_ROS packages to communicate and get data from the camera. Two ROS nodes are subscribed in the object detection process to get the depth map along with the detected image outputs. The setup for Taraxl is shown in Figure.7.11.

The different cameras considered for this research are compared in the Table.7.7.

Given the specifications above in Table.7.6, TaraXL is chosen because of its ROS compatibility and its depth sensing.



Figure 7.11 Jetson and TaraXL Camera

Camera	TaraXL	CU130
Depth range	500 - 3000 [mm]	-
Resolution	QVGA (2*320) x 240 VGA (2*640) x 480 WVGA (2*752) x 480	VGA (640 x 480) FHD (1920 x 1080) 13MP (4224x3156)
Frame rate	60 fps 60 fps and 30 fps 60 fps and 30 fps	60 fps and 30 fps 30 fps and 15 fps 5 fps and 2.5 fps
Colour	Monochrome	Colour
Size [mm]	95 x 17 x 27	30 x 30 x 31.3
ROS capability	Yes	NO

Table 7.7 NVIDIA TX2 Jetson Specifications

7.5 Ground Control Station

All the initial updates of the firmware and calibration of the aerial vehicle is carried out by Ground Control Station (GCS) [Driss et al., 2018]. Some of the basic features of GCS are Mission Planning, Navigation and Position Control, Payload Control, Communication and data exchange is detailed below:

Mission planning: GCS handles the path planning and mission plans for UAV to execute depending on planned trajectories.

Navigation and position control: GCS controls and displays live information of the vehicle along with the GCS and attitude.

Payload control: Cameras and Sensors connected on-board and to the Gamble System can be controlled during the mission execution using GCS. Both GCS applications used in the research are explained below and also shown in Table.7.8.

	Mission Planner	QGroundControl
Interface	Graphical	Graphical
Availability	Open-Source	Open-Source
MAVLink	Yes	yes
multiple PLaforms	Windows, Android	Windows, Linux, Android, MAC
Autopilot	Ardupilot, PX4	Any MAVLink compatable device

Table 7.8 Specifications of Mission Planner and QGC

7.5.1 Mission Planner (MP):

Mission Planner (MP) is written in Python language by Michael Osborne only compatible for windows is shown in Figure.7.12. Mission planner is an open-source application which provides Graphical display of vital functions like battery, GPS information, video stream and attitude. It also allows the user to download the log files and examine them Ardupilot, 2019a.



Figure 7.12 Mission Planner User Interface

7.5.2 QgroundControl (QGC)

Unlike MP, QGC is written in C++ using QT libraries by Lorenz Meier and is comparable with Windows, Linux, Android and Mac. It is flexible to use with many flight controllers and anything compatible with MAVLink protocol and offer opportunity to monitor and control aerial vehicle shown in Figure.7.13. The graphical interface of QGC include 2D map which can manage and track single and multiple aerial vehicles. It offers control of vehicle position, live video stream, attitude, GPS and to plan autonomous missions QGroundControl, 2019.

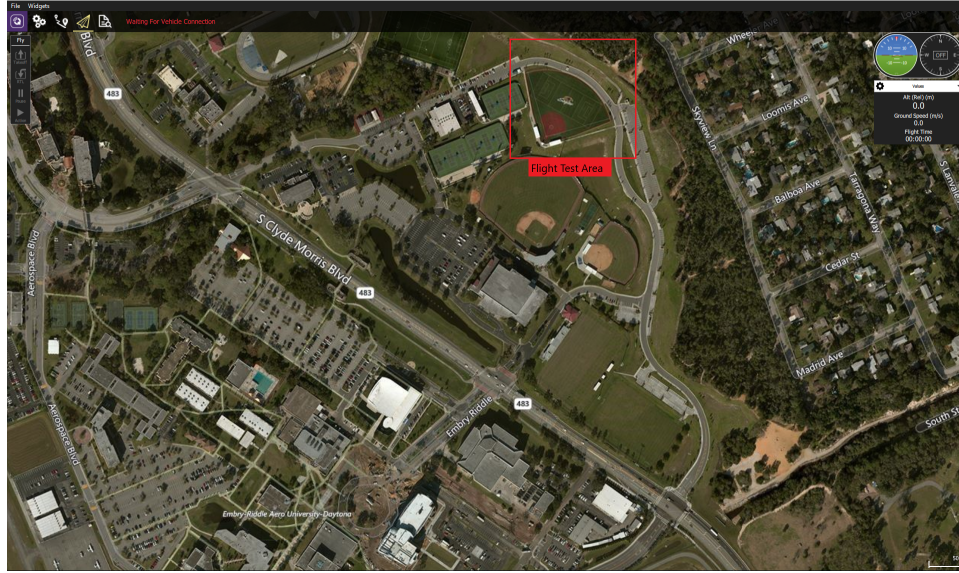


Figure 7.13 QGroundControl

7.6 Flight Testing Facilities

The vision system in this research is validated using both indoor and outdoor flights. Initial validation of the model is carried out at Vicon Indoor Testing Facility located at Advanced Dynamics and Control Lab (John Mica Engineering & Aerospace Innovation Complex (MicaPlex), ERAU) and outdoor flight tests are conducted at Outdoor Testing Facility at ERAU Intramural Soft-ball Field details of which are stated below.

7.6.1 Vicon Indoor Testing Facility

Vicon indoor test facility as shown in Figure.7.14 is a 3D motion capture room to obtain vehicle's position and orientation in real time. The facility shown in F features 12 infrared cameras that triangulates the 3D position of the target by processing the captured cues from each camera. This system is also configured to use the Virtual- Reality Peripheral Network (VRPN) streaming protocol, making it compatible from a software point of view. This information is passed through a router where it is then transmitted to anyone subscribed to the IP. In this study, Jetson TX2 was setup with ROS and VRPN to receive data from



Figure 7.14 Vicon System

Vicon system in real-time. This allowed validation of FF-net algorithm for detecting and tracking the leader vehicle.

The placement of the leader and follower in Vicon System in 3D orthogonal view (Figure.??) and 3D perspective view (Figure.7.15).

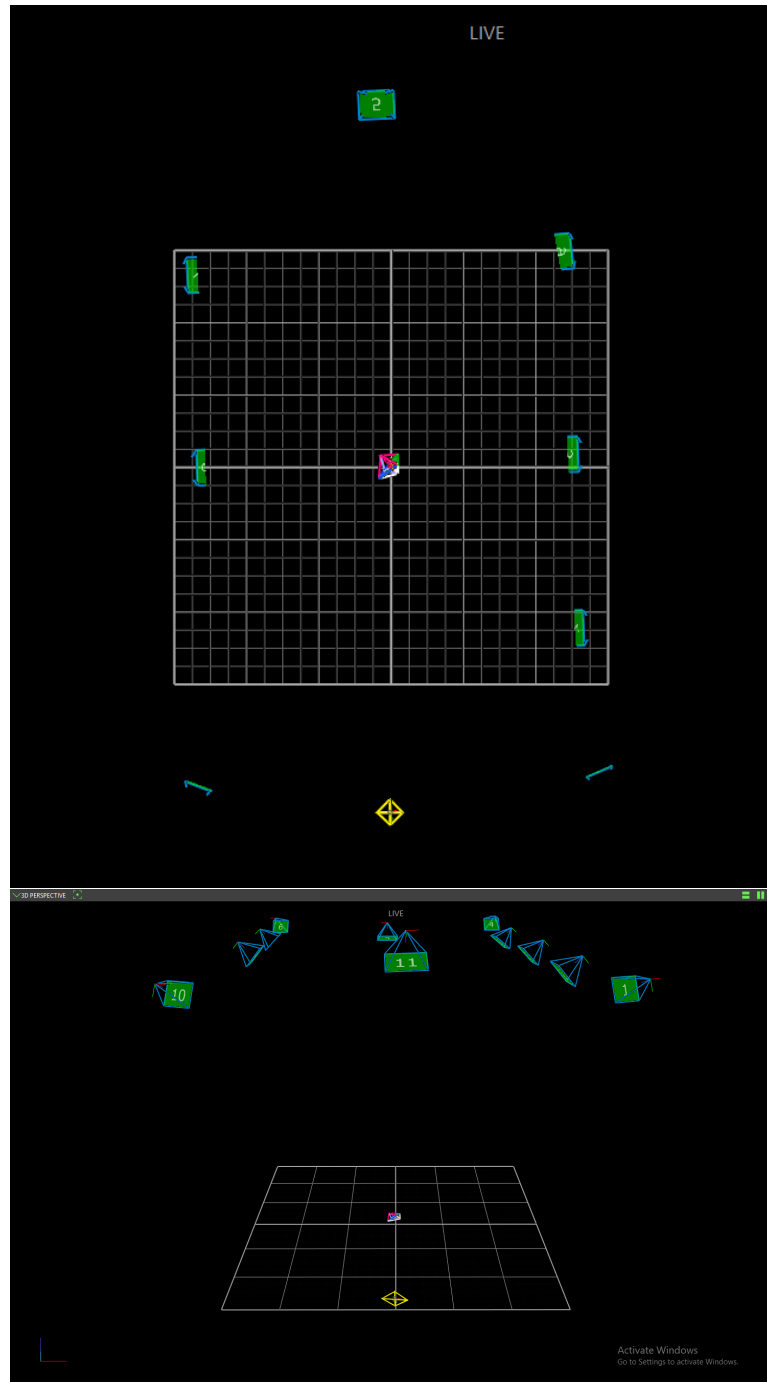


Figure 7.15 3D Orthogonal view and Perspective view of quad-rotor in Vicon system

7.6.2 Outdoor Testing Facility at ERAU Intramural Softball Field

Outdoor test flights were conducted at Outdoor Testing Facility at Embry-Riddle Aeronautical University (ERAU) Intramural Softball Field is shown in Figure.7.16. This is a designated testing facility to test unmanned autonomous vehicles in the University area. This field provides a wide area to test, good GPS coverage and soft flat landing surface for vehicle landing.



Figure 7.16 Outdoor Testing Facility at ERAU Intramural Softball Field

Chapter Eight

Experimental Results and Flight Test Program

In this chapter several flight tests and experimental results are presented. First section explain the results obtained from numerical simulations and gives the comparison of flight trajectories to validate the Formation Flight control architecture. This is followed by FF-Net detection in both indoor and outdoor environments. This chapter is ended with a section explaining the validation of formation flight detection system with Vicon system in Indoor testing facility and with GPS using flight tests performed outdoors at ERAU Intramural Softball field.

8.1 Formation Flight Numerical Simulations

Numerical simulations were performed with two quad-rotors flying in formation with a leader following a designated predefined trajectory while the follower is using formation flight control laws to follow the leader. In this simulation both the vehicles start at the same position and at the same time. As the leader vehicle starts to follow the predefined waypoints using a waypoint follower algorithm, follower gets the information of its states from the leader vehicle to track them and follow the leader. The given trajectory involves the vehicle to climb

to 45 meters and then follow the waypoints which represents the replication of a real-time outdoor flight test process. Figure.8.1 shows the leader aircraft with blue and follower with red line to show the trajectory. This shows the formation flight with two skyjib x8 models is achieved without clearance in the follower trajectory.

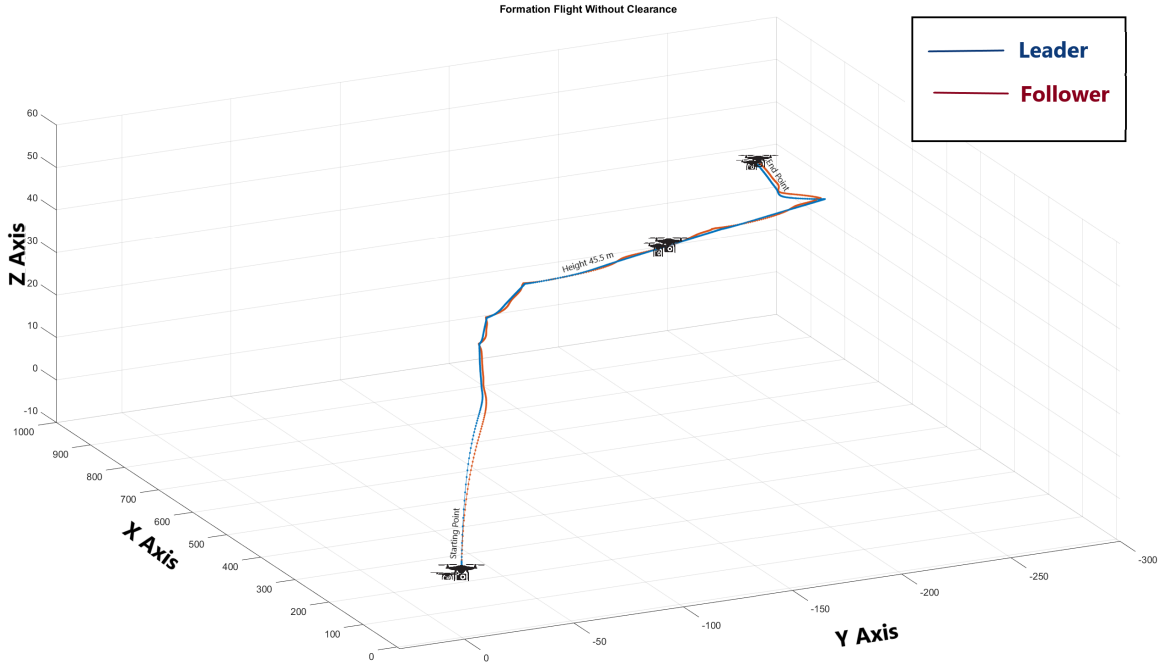


Figure 8.1 Formation Flight without Clearance

The tracking performance of the leader and follower vehicles in X, Y, Z is shown in Figure.8.2. This shows the individual tracking of the leader and follower trajectories without clearance.

Figure.8.3 shows fluctuations in *roll, pitch, yaw* orientations of follower vehicle to maintain its position to track and follow the leader aerial vehicle without any clearance. This shows that the formation flight control architecture presented in chapter four achieves a stability and tracks the leader vehicle even after performing some complex maneuvers involving changes in roll, pitch and yaw.

As described in Chapter 4, the formation flight control algorithm is developed to incorporate a predefined clearance in forward direction as well as in lateral direction. Which means

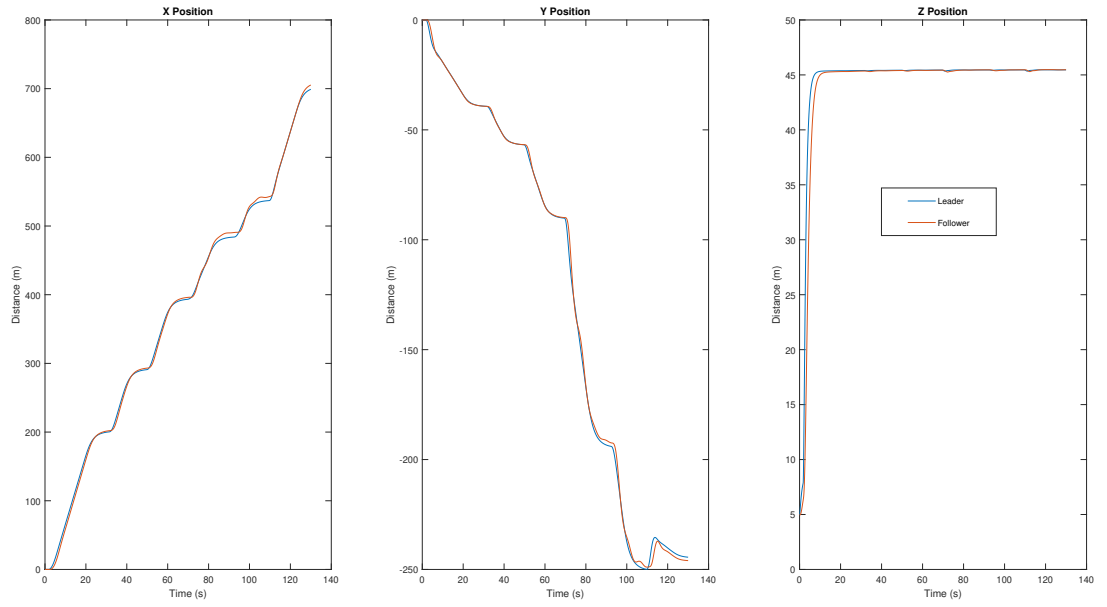


Figure 8.2 X, Y, Z tracking without clearance

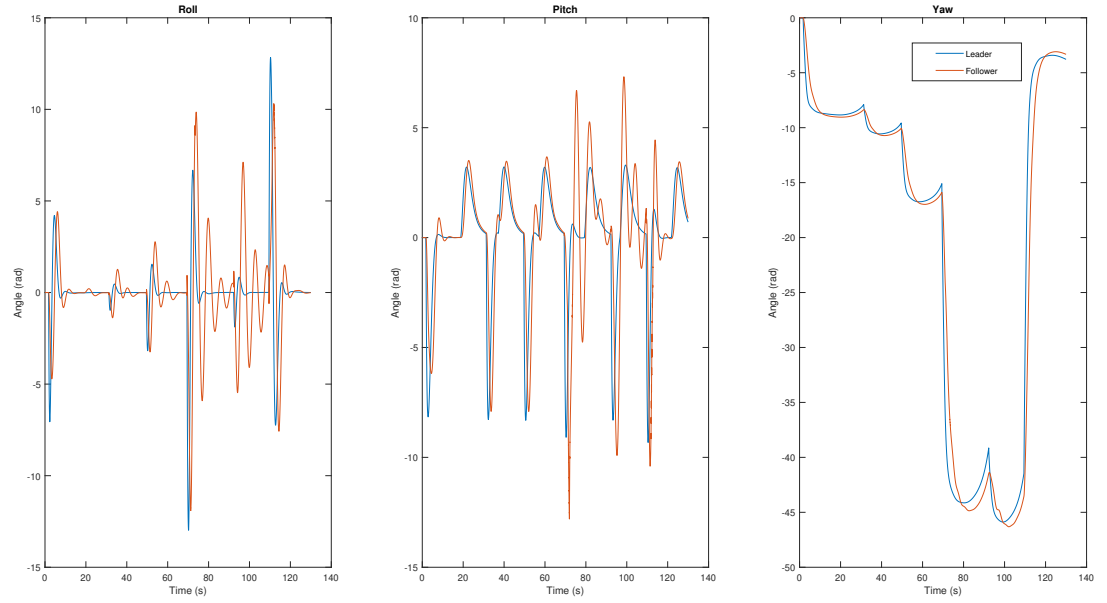


Figure 8.3 roll, pitch, yaw tracking without clearance

a clearance distance is added to the formation geometry to track the leader aerial vehicle at certain fixed distances from follower. Figure.8.4 clearly shows the trajectory generated by

formation flight control law with a predefined clearance in lateral direction. In this figure a distance of 10m in lateral direction can be seen at the end point even though both the leader vehicle and follower start at the same starting point.

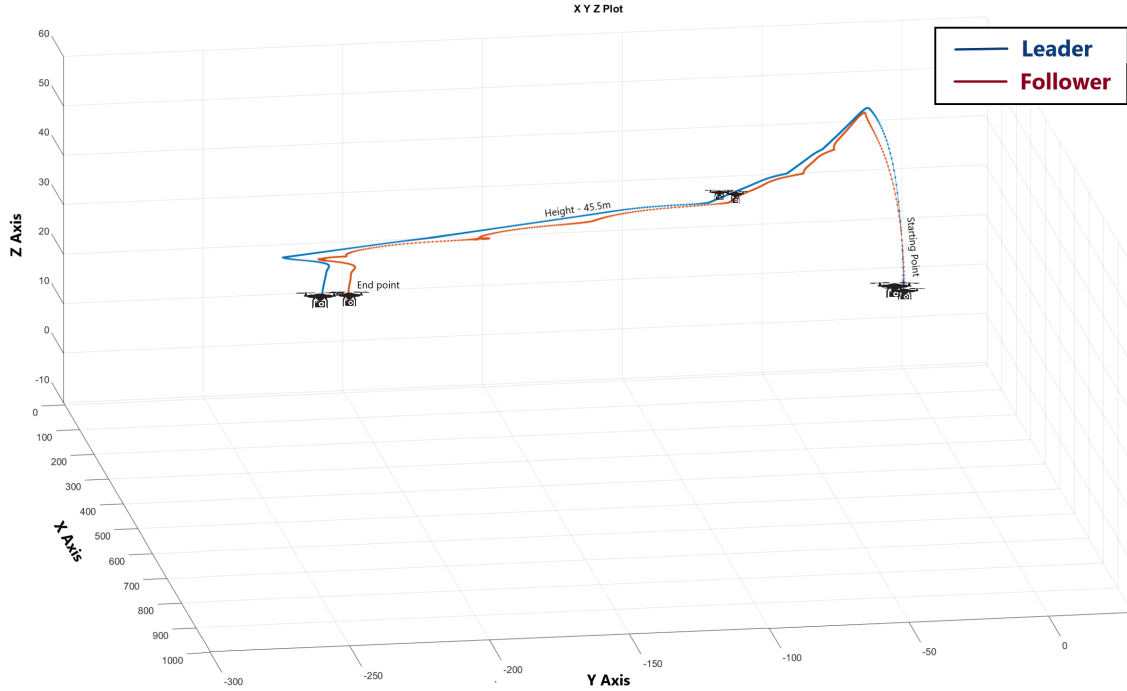


Figure 8.4 Formation Flight with Clearance

Figure.8.5 shows the individual tracking performance of the leader and follower formation flight in X, Y, Z positions with lateral clearance. From the Y -position trajectory graph presented this figure shows that the follower aerial vehicle is trying to keep the distance with the leader vehicle. X -position and Z -position tracking graphs shows the follower aerial vehicle following the leader with almost accurate tracking.

Figure.8.6 shows fluctuations in *roll, pitch, yaw* orientations of follower vehicle to maintain its position to track and follow the leader aerial vehicle with a predefined clearance in lateral direction. In this figure it can be seen that both roll and pitch almost tracks and converges until a complex maneuver is commanded at 70 sec of the simulation. The follower vehicle using formation flight control algorithm fluctuates to maintain the tracking position and converges eventually while yaw remain same in all configurations proving the stability of the

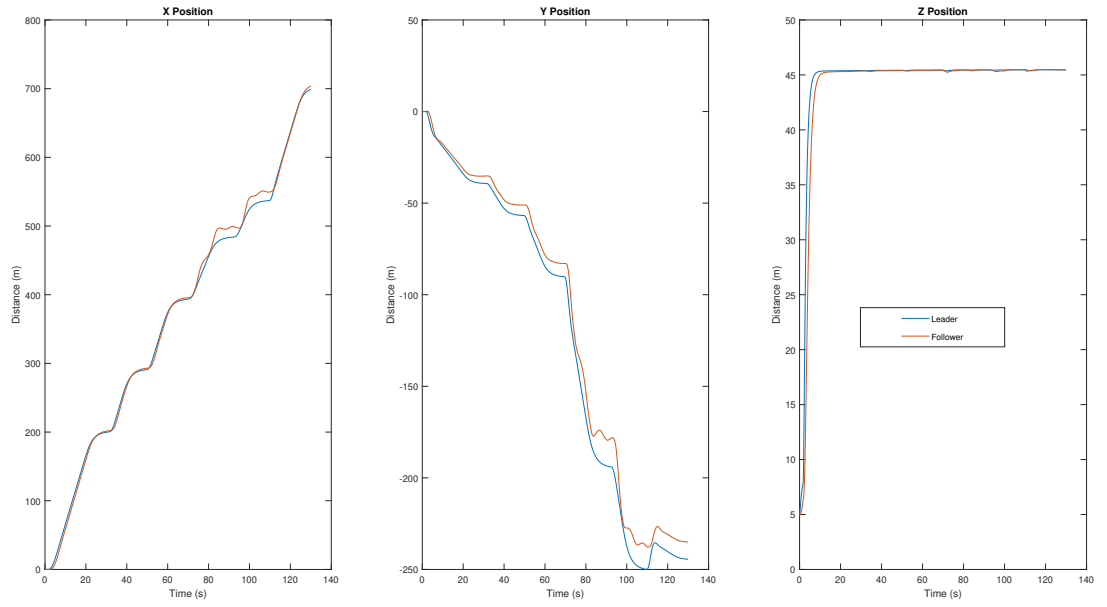


Figure 8.5 X, Y, Z tracking with clearance

controller even after performing some complex maneuvers involving changes in roll, pitch and yaw conditions.

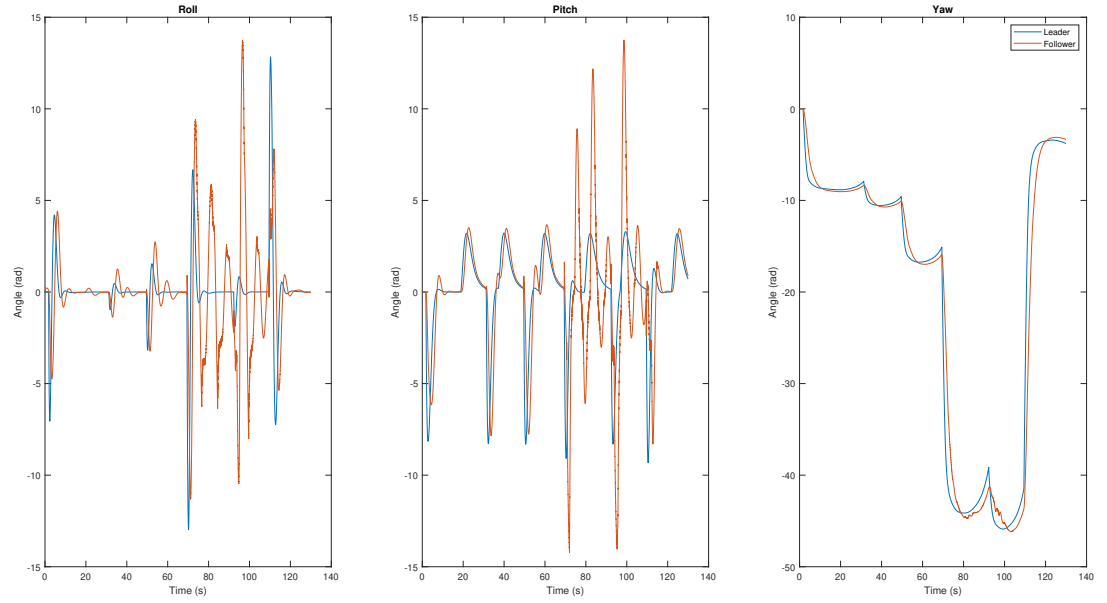


Figure 8.6 roll, pitch, yaw tracking with clearance

8.2 FF-Net Detection

Several indoor and outdoor flight tests were performed to validate the capabilities of FF-Net architecture to detecting and tracking the leader vehicle in a real time environment. Two general conditions considered in this thesis are indoor and outdoor flights. Most of the long range mission, surveillance and other applications require an UAS to fly in various outdoor conditions effecting the change in light, reflection of other objects, wind, blending in the background color and other conditions giving a challenge to detection system. To overcome these challenges the detection system is trained using data-sets involving samples of images in all conditions. Figure.8.7 show an example of outdoor detection using FF-Net. A bounding



Figure 8.7 outdoor FF-Net detection for class Raft

box is created around the detected object labeling the class of the detected object, in this case Raft is detected as leader aerial vehicle.

Figure.8.8 shows an example of the detection of a quad-rotor flying indoor in the Vicon testing facility.

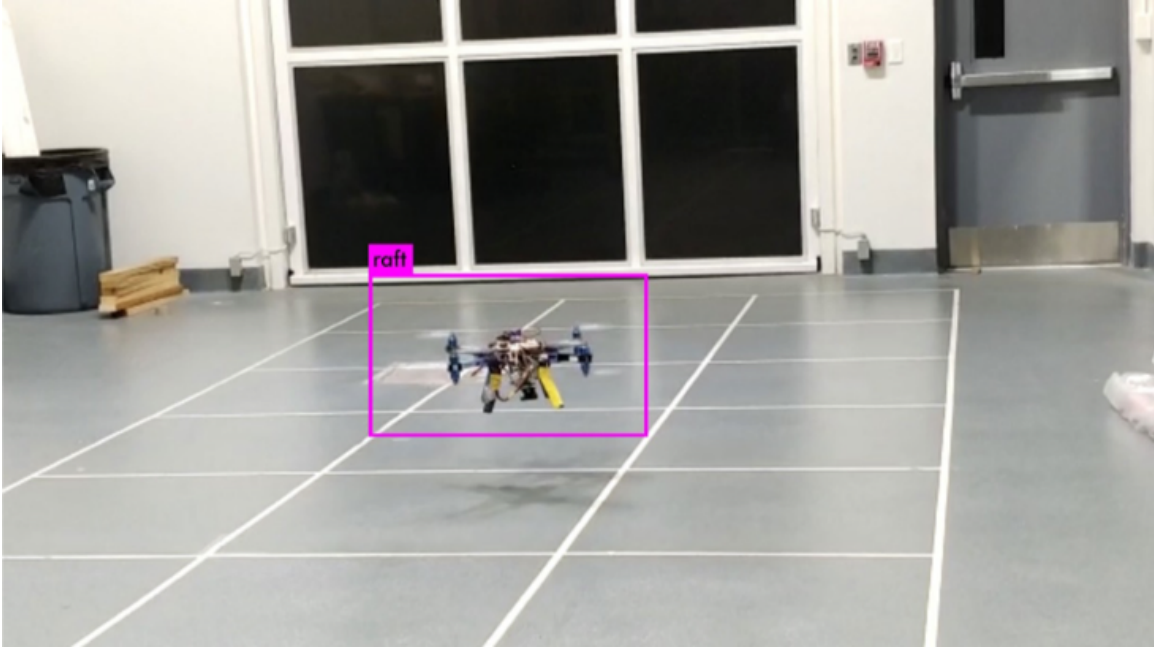


Figure 8.8 FF-Net detection for class Raft

8.3 FF-Net Flight Test Indoor and Outdoor

This section validates the vision system by comparing the results with Vicon system. The flight data collected from flight tests conducted in the vicon indoor testing facility at Advanced Dynamics and Controls lab and FF-Net detection data collected from vision system compared in Subsection.8.3.1. The results of the flight tests conducted at ERAU intramural soft-ball field to validate and compare the FF-Net detection with GPS data obtained from outdoor flight tests are shown in Subsection.8.3.2 .

8.3.1 Indoor Flight Results at Vicon Indoor Testing Facility

Numerous flight tests are conducted indoors at vicon indoor testing facility to test, tune and validate the FF-Net detection system. Vicon system provides an accurate tracking and trajectory of the vehicle at low sampling rates which acts as a good reference reference to test the vision system. The vision system data is collected using stereo camera and Jetson TX2 on-board skyjib x8 copter and the vicon system data is collected using VRPN_client_ROS

package on-board Jetson. This collected data is processed in MATLAB to get the correlations between them. Figure.8.9 shows a sample of results of a flight tests performed at the indoor testing facility. In Figure.8.9, red lines represent the actual trajectory of the leader quad-rotor while blue dots represent the position estimated by FF-Net. This figure is represented to show the real dimension of the drone trajectory and vision system detection in the indoor facility along with the global axis representation of the testing facility in X,Y,Z directions.

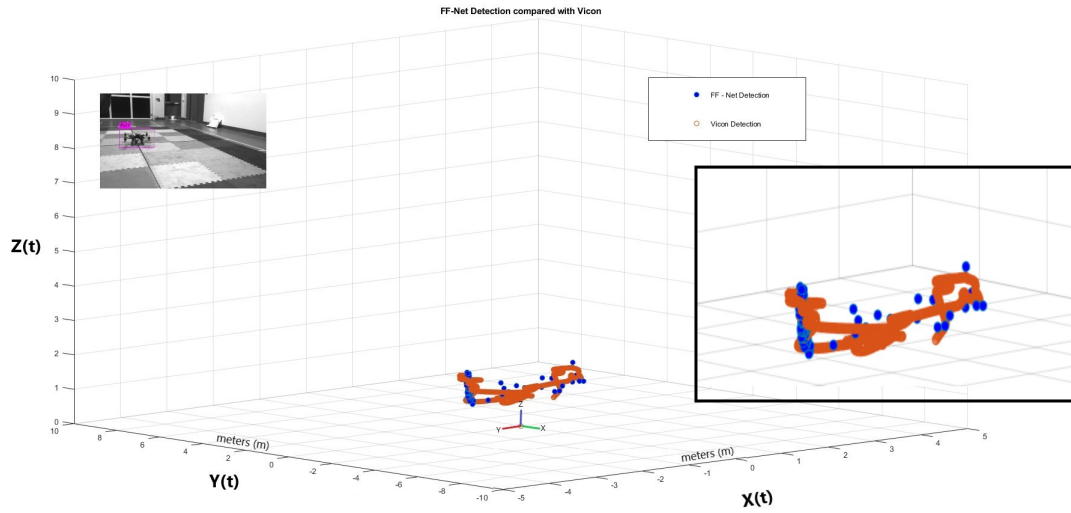


Figure 8.9 Compared results of the FF-Net with Vicon System

8.3.2 Outdoor Flight Results at ERAU Intramural Soft-ball Field

Outdoor flight tests are conducted to validate the FF-Net with the GPS data collected. The GPS data is collected using the UBlox GPS + Compass module and pixhawk on-board 3DR and the vision data from stereo camera and Jetson TX2 on-board Skyjib x8 copter.

The collected GPS data is analysed and a trajectory is created as shown in Figure.8.10 which shows the flight path while collecting GPS data during the flight test conducted at ERAU Intramural Soft-ball Field. Figure.8.11 shows the compare data of FF-Net detection system and GPS data collected. The GPS data collected is converted to homogeneous transformation coordinates to represent them in the local frame and a trajectory is generated.



Figure 8.10 Flight test data trajectory

This generated trajectory is then correlated with the detected object position from FF-Net detection system and plotted in Figure.8.11. In the figure, red lines represent the GPS trajectory of the leader quad-rotor while blue dots represent the position estimated by FF-Net. The sub-window in the figure shows the detection on leader aircraft in the camera frame using stereo camera.

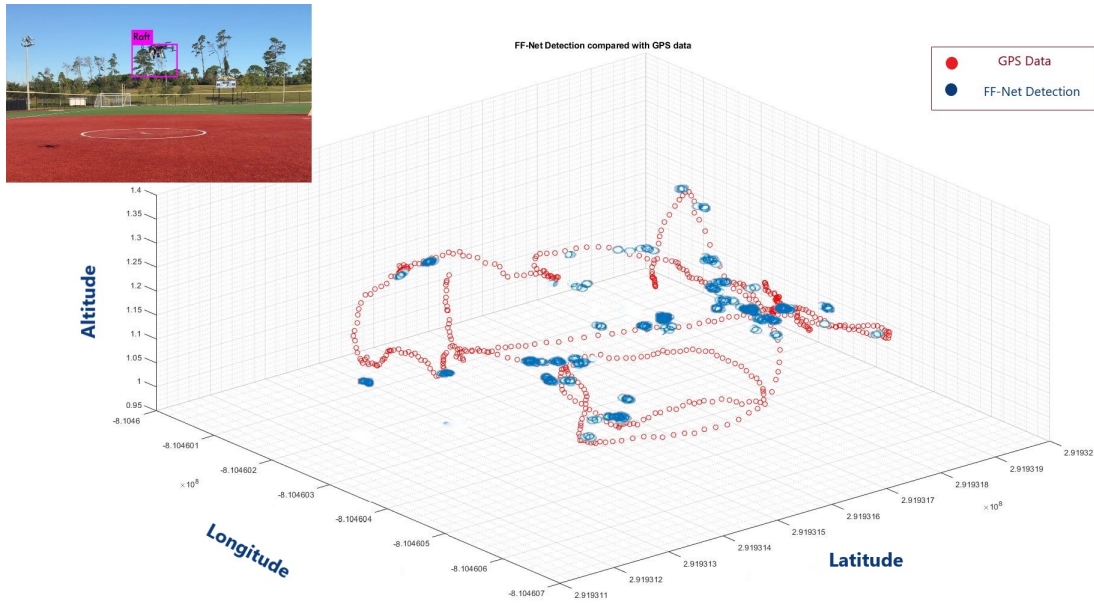


Figure 8.11 Compared results of the FF-Net with GPS data

Chapter Nine

Conclusions And Future Work

9.1 Conclusion

This thesis resulted in a wide study, development and implementation of formation flight control algorithm using a tracking algorithm, FF-Net, for relative navigation within a formation architecture using numerical simulations and experimental results obtained from flight tests conducted indoor and outdoor. This also resulted in development of a test-bed to validate the control algorithms and vision based algorithms which require high speed and performance using Jetson TX2 on-board computer, stereo camera and Pixhawk autopilot. Results for the developed controller without clearance and clearance in lateral direction and forward direction between the leader and follower aerial vehicle are presented in a simulation environment showing the performance and stability of the controller.

Numerical simulation results using MATLAB/Simulink and AirSim software were presented to demonstrate the capabilities of formation flight control laws to be integrated with FF-Net for detection, tracking and estimation of the position of a leader vehicle. Indoor flight tests were also performed a testing facility to evaluate accuracy and real-time capabilities of the integrated system for the position estimation of vehicle. Additionally, the outdoor tests provide the reliability of the system outdoor using GPS to compare its results.

Both indoor and outdoor from comparison of Vicon data and GPS data with FF-Net

algorithm show an acceptable performance and demonstrate the potential of this architecture to be used in different applications such as detection of cooperative and no-cooperative vehicles, proximity operations, among others.

9.2 Future Work

The work presented in this thesis can be extended to improve in many ways. The formation flight control algorithm although tested and evaluated in the simulation environment, it was not tested in a real flight. This can be tested performing multiple flight tests and tuning the controller in future to fly the multi-rotors in a stabilised formation.

More improvements can be achieved using a better camera and a better processor than Jetson TX2 as it provides only 12 frames per second when its at high performance mode. Vision system can also be improved with more optimised training and a larger data-set.

The vision system is trained to recognise any drone in the camera field of view using the second class "Drone" in the FF-Net detection system which is not used in this thesis as it uses a specific drone 3DR to be recognised with class "Raft". Using the "Drone" classification in FF-Net an algorithm can be developed to detect and follow any drone in shadow formation which is helpful for tracking cooperative and un-cooperative vehicles for military, air force and other industries.

The developed test-bed in this research can be used to test various control algorithms and vision based algorithms for various applications.

REFERENCES

- Ardupilot (2019a). *Mission Planner Ground Control Station*. URL: <https://ardupilot.org/planner/docs/mission-planner-ground-control-station.html>.
- (2019b). *SiK Telemetry Radio*. URL: <https://www.amazon.com/Readytosky-Telemetry-Standard-Version-Controller/dp/B01DHV4DVA>.
- (2019c). *UBlox GPS + Compass Module*. URL: <https://ardupilot.org/copter/docs/common-installing-3dr-ublox-gps-compass-module.html>.
- Blake, William and Dieter Multhopp (1998). “Design, performance and modeling considerations for close formation flight”. In:
- Bower, Geoffrey, Tristan Flanzer, and Ilan Kroo (June 2009). “Formation Geometries and Route Optimization for Commercial Formation Flight”. In: ISBN: 978-1-62410-130-4. DOI: [10.2514/6.2009-3615](https://doi.org/10.2514/6.2009-3615).
- Cobleigh, Brent (May 2002). “Capabilities and Future Applications of the NASA Autonomous Formation Flight (AFF) Aircraft”. In: ISBN: 978-1-62410-106-9. DOI: [10.2514/6.2002-3443](https://doi.org/10.2514/6.2002-3443).
- Dattalo, Amanda (2018). *ROS/Introduction*. URL: <http://wiki.ros.org/ROS/Introduction>.
- Driss, Aicha et al. (June 2018). “Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis”. In: pp. 1495–1500. DOI: [10.1109/IWCMC.2018.8450505](https://doi.org/10.1109/IWCMC.2018.8450505).
- Duan, Haibin et al. (Aug. 2013). “Hybrid Particle Swarm Optimization and Genetic Algorithm for Multi-UAV Formation Reconfiguration”. In: *Computational Intelligence Magazine, IEEE* 8, pp. 16–27. DOI: [10.1109/MCI.2013.2264577](https://doi.org/10.1109/MCI.2013.2264577).
- Elgendy, Mohamed (2019). *Deep Learning for Vision Systems*. ISBN: 9781617296192.
- Ermakov, Vladimir (2017). *MAVROS Package details*. URL: <https://github.com/mavlink/mavros>.
- Fusiello, Andrea (Jan. 2005). “Elements of Computer Vision: Multiple View Geometry.” In:
- Girshick, Ross (Apr. 2015). “Fast r-cnn”. In: DOI: [10.1109/ICCV.2015.169](https://doi.org/10.1109/ICCV.2015.169).

- Girshick, Ross et al. (Nov. 2013). “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. DOI: [10.1109/CVPR.2014.81](https://doi.org/10.1109/CVPR.2014.81).
- Harrison, Glennon J. (January 30, 2013). “Unmanned Aircraft Systems (UAS): Manufacturing Trends”. In: *CRS Report for Congress*.
- He, Kaiming et al. (Mar. 2017). “Mask R-CNN”. In:
- Institute, TU Delft Space (2018). *Proba-3, formation-flying with nano satellites*. URL: <https://spaceinstitute.tudelft.nl/showcase/formation-flying-with-nanosatellites/>.
- Isaaq, M. (2019). *List of Birds That Fly in V Formation*. URL: <https://www.liveanimalslist.com/interesting-animals/birds-that-fly-in-v-formation.php>.
- Jacques, Dave et al. (Aug. 2001). “An analytical study of drag reduction in tight formation flight”. In: DOI: [10.2514/6.2001-4075](https://doi.org/10.2514/6.2001-4075).
- jelonic, Marko B (2018). *darknet ROS Package details*. URL: https://github.com/leggedrobotics/darknet%5C_ros.
- Jetson (2019). *Jetson TX2 Module*. URL: <https://developer.nvidia.com/embedded/jetson-tx2>.
- King, Rachel and Ashok Gopalarathnam (Sept. 2005). “Ideal Aerodynamics of Ground Effect and Formation Flight”. In: *Journal of Aircraft - J AIRCRAFT* 42, pp. 1188–1199. DOI: [10.2514/1.10942](https://doi.org/10.2514/1.10942).
- Koubaa, Anis et al. (June 2019). *Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey*.
- Kramer, James and Matthias Scheutz (Feb. 2007). “Development environments for autonomous mobile robots: A survey”. In: *Auton. Robots* 22, pp. 101–132. DOI: [10.1007/s10514-006-9013-8](https://doi.org/10.1007/s10514-006-9013-8).
- Mason, William and Sergio Iglesias (Jan. 2002). “Optimum span loads in formation flight”. In: DOI: [10.2514/6.2002-258](https://doi.org/10.2514/6.2002-258).
- NASA (2017). *Global Hawk Aircraft*. URL: https://www.nasa.gov/multimedia/imagegallery/image%5C_feature%5C_2362.html.
- Ning, Andrew, Tristan Flanzer, and Ilan Kroo (Jan. 2010). “Aerodynamic Performance of Extended Formation Flight”. In: vol. 48. ISBN: 978-1-60086-959-4. DOI: [10.2514/6.2010-1240](https://doi.org/10.2514/6.2010-1240).
- NVIDIA (2019). *Harness AI at the Edge with the Jetson TX2 Developer Kit*. URL: <https://developer.nvidia.com/embedded/jetson-tx2-developer-kit>.
- Portugal, Steve (2016). “Lissaman, Shollenberger and formation flight in birds”. In: *Journal of Experimental Biology* 219.18, pp. 2778–2780. ISSN: 0022-0949. DOI: [10.1242/jeb.148114](https://doi.org/10.1242/jeb.148114).

- eprint: <https://jeb.biologists.org/content/219/18/2778.full.pdf>. URL: <https://jeb.biologists.org/content/219/18/2778>.
- Portugal, Steven (Sept. 2016). “Lissaman, Shollenberger and formation flight in birds”. In: *The Journal of Experimental Biology* 219, pp. 2778–2780. DOI: [10.1242/jeb.148114](https://doi.org/10.1242/jeb.148114).
- QGroundControl (2019). *QGroundControl User Guide*. URL: <https://docs.qgroundcontrol.com/en/>.
- Quigley, Morgan et al. (Jan. 2009). “ROS: an open-source Robot Operating System”. In: vol. 3.
- Ray, Ronald et al. (Sept. 2002). “Flight Test Techniques Used to Evaluate Performance Benefits During Formation Flight”. In: *Flight Mechanics Conference and Exhibit, Monterey, California, USA, AIAA*. DOI: [10.2514/6.2002-4492](https://doi.org/10.2514/6.2002-4492).
- Redmon, Joseph (2013–2016). *Darknet: Open Source Neural Networks in C*. <http://pjreddie.com/darknet/>.
- (2016). *YOLO: Real-Time Object Detection*. URL: <https://pjreddie.com/darknet/yolo/>.
- Redmon, Joseph, Santosh Divvala, et al. (June 2016). “You Only Look Once: Unified, Real-Time Object Detection”. In: pp. 779–788. DOI: [10.1109/CVPR.2016.91](https://doi.org/10.1109/CVPR.2016.91).
- Redmon, Joseph and Ali Farhadi (2016). “YOLO9000: Better, Faster, Stronger”. In: *arXiv preprint arXiv:1612.08242*.
- (2018). “YOLOv3: An Incremental Improvement”. In: *arXiv*.
- Ren, Shaoqing et al. (June 2015). “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39. DOI: [10.1109/TPAMI.2016.2577031](https://doi.org/10.1109/TPAMI.2016.2577031).
- Rice, Caleb et al. (July 2016). “Autonomous Close Formation Flight Control with Fixed Wing and Quadrotor Test Beds”. In: *International Journal of Aerospace Engineering* 2016, pp. 1–15. DOI: [10.1155/2016/9517654](https://doi.org/10.1155/2016/9517654).
- ros-drivers (2017). *VRPN_client_ROS Package*. URL: https://github.com/ros-drivers/vrpn%5C_client%5C_ros.
- Shah, Shital et al. (Jan. 2018). “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”. In: pp. 621–635. ISBN: 978-3-319-67360-8. DOI: [10.1007/978-3-319-67361-5_40](https://doi.org/10.1007/978-3-319-67361-5_40).
- sUAS (2013). *PX4 and 3D Robotics present Pixhawk: An Advanced, User-Friendly Autopilot*. URL: <https://www.suasnews.com/2013/08/px4-and-3d-robotics-present-pixhawk-an-advanced-user-friendly-autopilot/>.

- systems, e-con (2019). *Details of TaraXL stereo camera*. URL: <https://www.e-consystems.com/3d-usb-stereo-camera-with-nvidia-accelerated-sdk.asp>.
- Taru, Tejashwi Kalp (2019). *Training Yolo v3 model using custom dataset on Google colab*. URL: <https://www.techtejash.com/training-yolov3-using-custom-dataset-on-google-colab/>.
- Thomas, Carla (2001). *F/A-18 Autonomous Formation Flight (AFF)*. URL: <https://www.nasa.gov/centers/dryden/multimedia/imagegallery/AFF/EC01-0328-4.html>.
- Tice, Capt. Brian P. (1991). “Unmanned Aerial Vehicles - The Force Multiplier of the 1990s”. In: *Airpower Journal*.
- Vachon, M. et al. (Oct. 2003). “F/A-18 Performance Benefits Measured During the Autonomous Formation Flight Project”. In: DOI: [10.2514/6.2002-4491](https://doi.org/10.2514/6.2002-4491).
- Wagner, Eugene et al. (Aug. 2002). “Flight Test Results of Close Formation Flight for Fuel Savings”. In: ISBN: 978-1-62410-107-6. DOI: [10.2514/6.2002-4490](https://doi.org/10.2514/6.2002-4490).
- Weimerskirch, Henri et al. (Nov. 2001). “Energy saving in flight formation”. In: *Nature* 413, pp. 697–8. DOI: [10.1038/35099670](https://doi.org/10.1038/35099670).
- Wu, Falin, Jiemin Chen, and Liang Yuan (Mar. 2017). “Leader-Follower Formation Control for Quadrotors”. In: vol. 187, p. 012016. DOI: [10.1088/1757-899X/187/1/012016](https://doi.org/10.1088/1757-899X/187/1/012016).

APPENDIX



Figure 1 FF-Net detection with class "Drone"

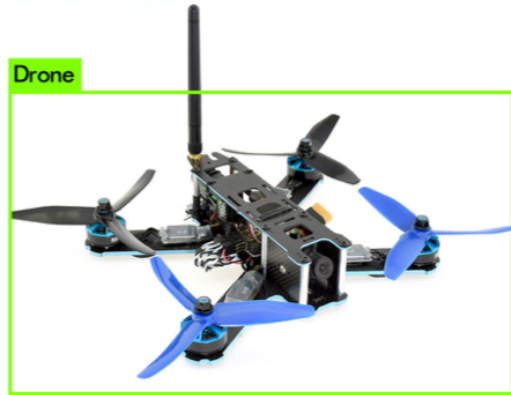


Figure 2 FF-Net detection with class "Drone"