

**DESIGN, DEVELOPMENT AND CONTROL OF AN AUTONOMOUS MODULAR
SNAKE ROBOT RECONFIGURABLE INTO QUADCOPTER**

(FLYING SNAKE ROBOT)

Submitted in

In partial fulfilment of requirement for the award of

Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

KHUSHDEEP SINGH

(BT13ECE095)

Under The Guidance Of

Dr. SUDHIR KUMAR

Assistant Professor , Department of Electronics and Communication Engineering

VISVESVARAYA NATIONAL INSTITUTE OF TECHNOLOGY

NAGPUR- 440010 (INDIA)

(2016-

2017)



**DEDICATED TO MY PARENTS SHRI BEANT SINGH AND SMT KULDEEP KAUR AND
SISTER GURPREET**



CERTIFICATE

This is to certify that the project work entitled “ **Design ,Development and Control of an Autonomous Modular Snake Robot Reconfigurable Into Quadcopter** aka Flying Snake”, is a bona-fide work written by:

Mr. Khushdeep Singh

BT13ECE095

In partial fulfilment of the requirements for the award of the Degree of **Bachelor of Technology** in **Electronics and Communication Engineering** from **Visvesvaraya National Institute of Technology, Nagpur** during 2016-2017.

Dr. Sudhir Kumar

Assistant Professor

Electronics and Communication
Engineering Department

VNIT, Nagpur

Dr. A.S.Gandhi

Head of the Department

Electronics and Communication
Engineering Department

VNIT, Nagpur



**VISVESVARAYA NATIONAL INSTITUTE OF
TECHNOLOGY, NAGPUR
2016-2017**

DECLARATION

The project work entitled “ Design ,Development and Control of an Autonomous Modular Snake Robot Reconfigurable Into Quadcopter aka Flying Snake” is my own work carried under the guidance of Dr.Sudhir Kumar, Assistant Professor in the Electronics and Communication Engineering Department, VNIT Nagpur. This work in the same form or in any other form is not submitted by me or anyone else for the award of any degree.

Khushdeep Singh

ACKNOWLEDGEMENTS

First and foremost, I express mine sincere gratitude to our guide, Prof. Dr.Sudhir Kumar, Assistant Professor, Department of Electronics and Communication Engineering, for his supervision.

I thank Dr.A.S.Gandhi, Head of the Department, Electronics and Communication Engineering for giving me the opportunity and motivation to proceed with our project.

I would also like to thank my friend from Mechanical Engineering Department- Parag Khanna without whose help, the completion of the project would have not been a reality. I also thank my seniors and mentors Rohan thakker, Ajinkya kamat, Anshul paigwar, Akash Singh of IvLabs for their constant support and Project Idea.

I finally wish to thank my family members for their constant support and encouragement which have been a motivating factor throughout the project. My heartfelt gratitude to all my friends who have stood by me and helped me through hardships.

Khushdeep Singh

ABSTRACT

Robots capable of manoeuvring on land like Snake robots and in air like quad copters are beneficial for various applications. The individual dynamics of both robots are studied and their drawbacks are aimed to be overcome with a Flying snake robot while retaining their characteristics and advantages. This thesis include Design, Development and control of semi-autonomous snake robot configurable into a quadcopter. The thesis documents the design as well as the flight controller implemented on Snake robot.

TABLE OF FIGURES

Figure 1-1 : <i>Some examples of Land based Mobile Robots</i>.....	2
Figure 1-2 : <i>Some examples of Mobile Robots in Air</i>.....	3
Figure 1-3 : <i>ACM Family of Snake Robots</i>.....	4
Figure 1-4 : <i>Humanoid Robots: ASIMO, HERMES and NAO</i>.....	5
Figure 2-1 : <i>A Desert Snake and a Robotic Snake performing Side Winding Motion</i>.....	8
Figure 2-2 : <i>Lateral Undulation performed by Snake Robot and Real Snake</i>.....	8
Figure 2-3 : <i>Rectilinear motion performed by Snake Robot and Real Snake</i>.....	9
Figure 3-1 : <i>Quadcopter fitted with a Camera and its Model</i>.....	11
Figure 3-2 : <i>Mathematical Model of a Simple Quadcopter in X configuration</i>.....	12
Figure 3-3 : <i>Control of Various Parameters by Motors' Speed Variation in Quadcopter</i>.....	13
Figure 3-4 : <i>PID control loop for one control parameter of Quadcopter</i>.....	14
Figure 3-5 : <i>Cascaded PID control structure, Involves all Control Parameters of Quadcopter</i>.....	14
Figure 4-1: <i>Construction of BLDC Motor</i>.....	17
Figure 4-2: <i>Working of BLDC Motor</i>.....	18
Figure 4-3 : <i>Numbering on a Propeller</i>.....	20
Figure 4-4 : <i>Effect of Pitch on the Flight</i>.....	21
Figure 4-5 : <i>Comparison between Bullnose and Standard propeller</i>.....	22
Figure 4-6: <i>An Electronic Speed Controller</i>.....	23
Figure 4-7: <i>Arduino Uno Board with Detailed Pin Layout</i>.....	24
Figure 4-8: <i>IMU- MPU6050</i>.....	25
Figure 4-9 : <i>Detailed Dimensions of Dynamixel AX-12 Actuator</i>.....	26
Figure 4-10 : <i>Various Goal positions and values for Dynamixel</i>.....	26
Figure 4-11: <i>Detailed OpenCM-9.04 Board</i>.....	27
Figure 4-12: <i>A Lipo battery with various Ratings</i>.....	28
Figure 4-13: <i>Naze32 board with detailed Pin layout</i>.....	31
Figure 4-14 : <i>FS-T6 6 Channel Transmitter and Receiver</i>.....	32
Figure 4-15 : <i>Various Modules connected together to form Flying Snake Robot</i>.....	39

Figure 4-16 : <i>Various Views of Flying Snake Robot in Open/Snake configuration</i>	39
Figure 4-17 : <i>Various Views of Flying Snake Robot in Closed/Quadcopter configuration</i>	40
Figure 4-18: <i>Comparison of various configurations of 3D Modelled Flying Snake Robot to Fabricated Real Robot</i>	40
Figure 4-19 : <i>BLDC connector under Simulation for Static Displacement under applied Load and Holes as fixture</i>	41
Figure 4-20 : <i>Representation of BLDC mounted on Horizontal Axis Dynamixel</i>	42
Figure 4-21: <i>Comparison of Symmetry between Flying Snake Robot and Standard Quadcopter</i>	42
Figure 4-22 : <i>Balancing to remove Mass Imbalance</i>	43
Figure 4-23: <i>Component Flow Diagram for Flying Snake Robot</i>	43
Figure 4-24 : <i>Flow chart for Working of Flying Snake Robot</i>	45
Figure 4-25 : <i>Flying Snake Robot in Snake configuration</i>	46
Figure 4-26 : <i>Flying Snake Robot in Quadcopter configuration</i>	46

LIST OF TABLES

Table 2-1 : Parameters for various Snake Robot Gaits.....	10
Table 4-1: Advantages and Disadvantages (in Focus) of Snake robots.....	17
Table 4-2: Advantages and Disadvantages (in Focus) of Quadcopters.....	18
Table 4-3: Description and different Views of CAD model of 'Simple Connector for Dynamixel'.....	33
Table 4-4 : Description and different views of CAD model of 'Special Connector for Dynamixel'.....	34
Table 4-5 : Description and different views of CAD model of 'BLDC Connector'.....	34
Table 4-6 : Different views of CAD model of the Sub-Assembly 'Dynamixel with Simple Connector'.....	35
Table 4-7 : Different views of CAD model of the Sub-Assembly 'Dynamixel with Special Connector'.....	36
Table 4-8 : Different views of CAD model of the Sub-Assembly 'BLDC(With Propeller) and BLDC Connector'.....	36
Table-4.9 : Different views of CAD model of the Sub-Assembly IV.....	37
Table 4-10 : Different views of CAD model of the Sub-Assembly V, 'The Module of the Flying Snake Robot'.....	38

LIST OF ACRONYMS USED

Aka- also known as

COM- Centre of Mass

VNIT-Vesvesvaraya National of
Technology

ACM- Active Cord Mechanism

ASIMO-Advanced Step in Innovative
Mobility,

HERMES- Highly Efficient Robotic Mechanisms
and Electromechanical System

PID- Proportional , Integral and Derivative

BLDC- Brushless Direct Current

ESC- Eletronic speed controller

IMU- Inertial Measurement Unit

MPU- Memory Processing Unit

LiPo- Lithium Polymer

A-Ampere

s- seconds

mm- multimeters

WWII- World War 2

PPM-Pulse Position Modulation

MHz- Mega Hertz

USB- Universal Serial Bus

mAh- Milli Ampere Hour

FC- Flight Controller

DoF- Degrees of Freedom

No. – Number

GPIO- General Purpose Input/Output

3D- 3 dimensions

CAD- Computer Aided Design

UGV- Unmanned Ground vehicle

UAV- Unmanned Aerial vehicle

Etc- et cetra

PARC

CKbot

VTOL- Vertical take-off and Landing

DC- Direct Current

EMI- Electromagnetic Interference

RPM- Rotations per minute

AC- Alternating current

V- Voltage

PWM- Pulse Width Modulation

CONTENTS	
Abstract	
Table of Figures	
List of Tables	
List of Acronyms used	
Contents	
1.Introduction	
1.1 Mobile Robots	
1.1.1 Mobile Robots in Land	
1.1.2 Mobile Robots in Air	
1.2 Locomotion in Mobile Robots	

1.2.1 Apodal robots	4.2.8 LiPo battery		
1.2.2 Podal Robots	4.2.9 Flight Controller		
1.3 Modular Robots	4.2.10 (Flysky)FS-T6 Transmitter and Receiver		
1.4 Motivation	4.3 Mechanical Parts and Assembly		
1.5 Design Statement	4.3.1 Parts		
1.6 Thesis Outline	4.3.2 Assemblies		
2. Mobile Apodal robot on Land : Snake Robot	4.3.3 Final Assembly		
2.1 Biological Study	4.3.4 Fabricated Flying Snake Robot		
2.1.1 Skeletal Structure	4.4 Design Aspects of the Flying Snake Robot		
2.1.2 Locomotion	4.4.1 Major Design and Control Challenge: Asymme		
2.2 Snake Robots	4.5 Flying Snake Robot Working:		
2.2.1 Snake Robot Locomotion Gaits	4.5.1 Working as Snake		
2.2.2 Kinematics of Snake Robot	4.5.2 Working as Quadcopter		
2.3 Applications	5. Results		
3. Mobile robot In air : Quadcopter	6. Future Work		
3.1 Quadcopter-Definition	7 . Project Codes		
3.2 Flight Control & Balancing	7.1 Codes for Arduino		
3.3 Working of Quadcopter	7.1.1 Interfacing mpu6050 with arduino		
3.4 Mathematical analysis and Control Strategy A	7.1.2 Code to run BLDC motor via ESC		
3.5 Applications	7.1.3 Code to receive values from different channels(
4. Description of the flying snake robot	Transmitter Used		
4.1 Basic Concept	7.1.4 Final Arduino code to control Flying Snake I		
4.2 Components Involved	7.2 Code for OpenCM9.04		
4.2.1 Brushless Motors- BLDC	8.Referenes		
4.2.2 Propellers			
4.2.3 Electronic Speed controllers			
4.2.4 Arduino Uno			
4.2.5 Inertial Measurement Unit			
4.2.6 Dynamixel AX-12A			
4.2.7 Robotic OpenCM9.04			

1: INTRODUCTION

Robots may be defined as a machine, especially one programmable by a computer, capable of carrying out a complex series of actions automatically. Robots can be guided by an external control device or the control may be embedded within. The rise of technology has lead to rapid development in the Robotics and Automation Industry and it is rather crystal clear from the trends in past 15 years that 21st century is all about robotics and integration of various technologies for use of automation in all possible sectors. Current study in robotics majorly revolves around giving them better structure(Human-like or other animals like), improving their locomotion and finding ways for them to be completely autonomous(Artificial Intelligence). This thesis resonates with the quest to make robots better in locomotion. At present such investigation concentrates not only on the locomotion of these robots, but also on their capabilities to form different structures or change their shapes.

1.1 Mobile Robots:

Mobile robots have the capability to move around in their environment and are not fixed to one physical location. Mobile robots can be autonomous which means they are capable of navigating an uncontrolled environment without the need for physical or electro-mechanical guidance devices. Alternatively, mobile robots can rely on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space. Mobile robots have become more commonplace in commercial and industrial settings. Hospitals have been using autonomous mobile robots to move materials for many years. Warehouses have installed mobile robotic systems to efficiently move materials from stocking shelves to order fulfilment zones. Mobile robots are also a major focus of current research and almost every major university has one or more labs that focus on mobile robot research. Mobile robots are also found in industrial, military and security settings. Domestic robots are consumer products, including entertainment robots and those that perform certain household tasks such as vacuuming or gardening. In nature the movement of animals is adapted to the environment in which they live. This gives them flexibility to perform various tasks which are impossible of a fixed robot. Mobile robots that employ mechanisms for motion.

Mobile robots may be classified under two types: Mobile Robots on Land and Mobile Robots in Air.

1.1.1 Mobile Robots in Land:

Land or home robots are usually referred to as Unmanned Ground Vehicles (UGVs). They are most commonly wheeled or tracked, but also include legged robots with two or more legs (humanoid, or resembling animals or insects).

An **unmanned ground vehicle (UGV)** is a vehicle that operates while in contact with the ground and without an onboard human presence. UGVs can be used for many applications where it may be inconvenient, dangerous, or impossible to have a human operator present. Generally, the vehicle will have a set of sensors to observe the environment, and will either autonomously make decisions about its behaviour or pass the information to a human operator at a different location who will control the vehicle through teleoperation. Unmanned robotics are being actively developed for both civilian and military use to perform a variety of dull, dirty, and dangerous activities.



Figure 1-1 : Some examples of Land based Mobile Robots

1.1.2 Mobile Robots in Air:

An **unmanned aerial vehicle (UAV)**, commonly known as a **drone**, is an aircraft without a human pilot aboard. UAVs are a component of an unmanned aircraft system (UAS); which include a UAV, a ground-based controller, and a system of communications between the two. The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator, or fully or intermittently autonomously, by onboard computers. Compared to manned aircraft, UAVs were originally used for missions too dull, dirty or dangerous for humans. While they originated mostly in military applications, their use is rapidly expanding to commercial, scientific, recreational, agricultural, and other applications, such as policing and

surveillance, product deliveries, aerial photography, agriculture, smuggling and drone racing. Civilian drones now vastly outnumber military drones, with estimates of over a million sold by 2015, so they can be seen as an early commercial application of Autonomous Things, to be followed by the autonomous car and home robots.



Figure 1-2 : *Some examples of Mobile Robots in Air*

1.2 Locomotion in Mobile Robots :

One of the research areas in robotics is that of locomotion; giving the robots locomotive capabilities so that they can move from one place to another. These robots receive the generic name of mobile robots. At the same time the study of locomotion is performed at the two levels mentioned below. Investigations of the superior level start with the supposition that robots can move, without taking into account the mechanisms that make it possible (feet, wheels ,etc) and concentrates on the task of the superior level such as path planning, vision, collaboration, etc.. The same happens in animals, in the study of the inferior level of locomotion, robots can be classified according to the effectors employed to move them: wheels , caterpillar tracks, feet or the body. These are the four classical categories for the study of locomotion; nevertheless the classification is not definitive. The themes for investigation at the lower level of locomotion are the properties of the various effectors, how to achieve the co-ordination of the actuators, the different gaits , algorithms of control, etc.

1.2.1 Apodal robots:

In contrast to terrestrial movement by means of feet some living beings use corporal movements. The robots that use this kind of movement are known as apodal robots. These robots possess characteristics that make them unique, the same as their counterparts: snakes, worms and maggots. On one hand is the ability to change their form. Compared with the rigid structures of the rest of the robots, the apodals can bend and adapt to the form of the terrain on which they

move. On the other hand their section is very small compared to their size, which permits them to enter small tubes or orifices and get to places inaccessible to other robots.

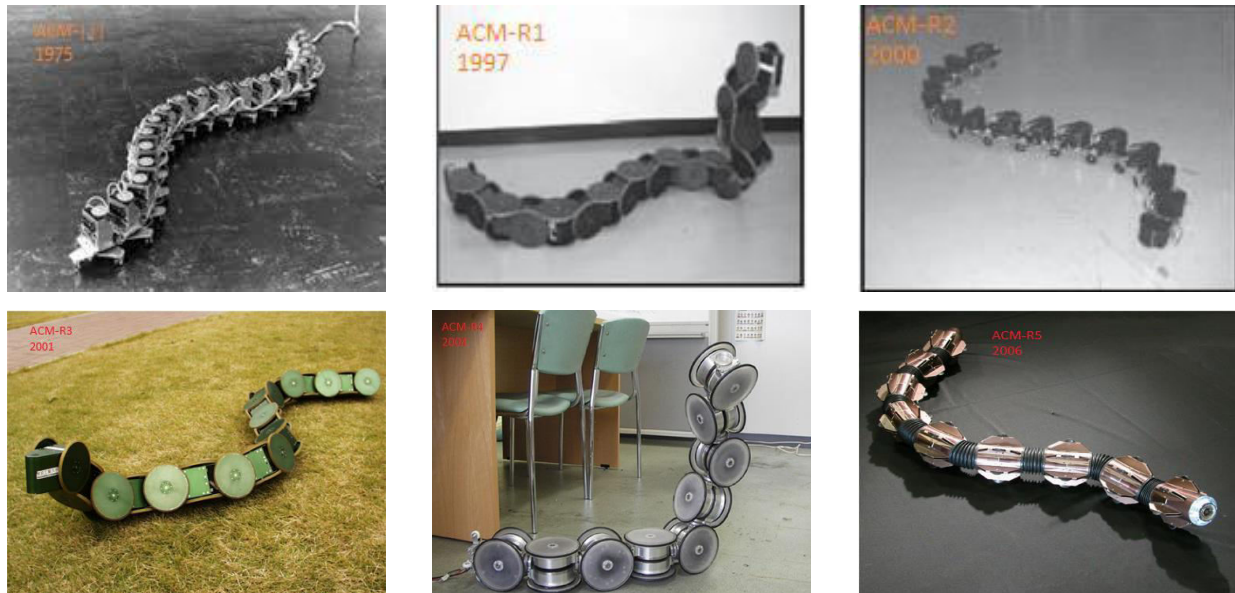


Figure 1-3 : ACM Family of Snake Robots

Hirose, of the Tokyo Institute of Technology pioneered studies of snake's bio-mechanics for its applications to robotics.–The first snake robot called ACM-2 (Active cord mechanism) was implemented in 1976. It was only in 1997, that a new prototype ACM-R1 was created. ACM-R1 is the same as ACM-2 , however, it is more compact. The next prototype from the ACM family includes the ACM-R2.This allows the robot to take a three- dimensional form. ACM-R3 has an alternating pitch and yaw freedom at the joints. ACM-R4 was designed for industrial application and uses an active wheel for motion. The latest ACM-R5 was designed in 2006 and is an amphibious snake robot.

Other institute pursuing active research in the field of apodal robots is CMU. The research at CMU has produced a number of snake robots including CMU's MODSnake.

1.2.2 Podal Robots :

The field of podal robots is as much more mature as compared with its snake counterparts. These are the ones who walk by feet and are recently been used for research purposes in order to have implementations in space explorations, search and rescue operations etc. Some of them include:- ASIMO , HERMES , Elvis , PINO etc.



Figure 1-4 : *Humanoid Robots: ASIMO, HERMES and NAO.*

1.3 Modular Robots :

In his 1995 doctoral thesis Mark Yim proposed a new approach to the locomotion problem. The traditional solution is focused on designing a specific robot based on the analysis of a terrain's characteristics. What Yim proposed was using robots based on modules with the capability of re-assembling themselves into different forms. In this way, these new modular robots could change their form adopting different configurations and gaits according to the terrain where they were operating at a particular moment.

This concept can be extended to robots that do not attach and detach their modules.

Polypod was the first modular robot. The next iteration of Polypod is named as Polybot and was developed in Xerox PARC. CKbot is another modular robotic system developed by Mark Yim is one of the most sophisticated modular robotics systems today.

1.4 Motivation :

Snake robots are land manoeuvres used for search and rescue operations, underwater excavations, spying , Invasion prevention, Pipe inspections and Navigate on land, water and climb, but these tasks may involve complex gait performance. On the other hand Qudcopters are air manoeuvres having high speeds and used in applications like Research Platform ,Military, for commercial purposes etc. but find difficulty to get through constrained regions. Thus in order to make exploration easier on both land and air , it is essential to impart snake robot with flying properties.

1.5 Design Statement : The Flying snake robot should have the following features:

- Able to perform snake gait motions like Side-winding, Caterpillar forward and backward motion, Lateral Undulation and Rotation.
- The dynamixel actuators must be able to sustain the back torque generated due to Brushless DC motors used for flying.
- Robot should configure itself into an symmetric ‘square ‘ configuration and is able to hover.
- The transformation from snake-to-quadcopter and vice versa should be controlled by an external remote controller.

1.6 Thesis Outline :

The thesis deals with the design and fabrication of an Flying Snake robot. The proceeding chapter deals with the detailed study of snake skeleton and their various locomotion. The later part deals with the kinematics of the snake robots and their applications in the present day environments. The next chapter deals with Quad-rotors, their working principles, mathematical analysis, control strategies, Flight control , balancing and finally their applications. Next chapter deals with the fabrication of Flying snake robot. Firstly the pros and cons of Snake Robots and Quadcopter are discussed. Then follows the components used in the project and the finally the fabricated structure of the robot. Lastly the design aspects and working are specified with the future scope at the end. Next chapter talks about the results followed by the chapter providing software coding for snake gaits performance and hovering. Lastly references are provided chapterwise.

2. MOBILE APODAL ROBOT ON LAND : SNAKE ROBOT

2.1 Biological Study :

Snakes are diverse creatures that occupy a wide range of habitats. They also have a wide range of locomotive capabilities, ranging from crawling and burrowing to climbing and even swimming. While snakes all have a similar structure, they do exist in a variety of sizes and aspect ratios. For example, snakes such as Boidae family (Boas and Pythons) tend to have thicker, heavier bodies, while snakes in families such as Leptotyphlopidea family (Thread snakes and Worm snakes) tend to have thinner body types. Snakes also range in length from more than 20 feet for reticulated pythons and anacondas, to substantially less than 1 foot long for many of the smaller varieties.

2.1.1 Skeletal Structure :

The design a snake is a simple structure that is repeated many times. Snakes bodies are elongated forms that consist of a long backbone made of many vertebrae. In fact, there are only three different kinds of bones in the entire snake skeleton: the skull , the vertebrae, and the ribs. Snake backbones consist of 100-400 vertebrae, and the design of each vertebrae allows small motions in both the lateral and vertical directions. They do not allow any twisting, however, and thus act as compliant universal joints. Each vertebrae itself only allows a small amount of angular motion, but the motions of many vertebrae allow snakes to drastically curve their bodies. Each vertebrae allows rotation of 10-20 degrees in the horizontal plane, and between 2-3 degrees in the vertical plane.

2.1.2 Locomotion :

Snake inspired locomotion provides the following advantages over traditional forms of locomotion in both animals and machines.

- Due to their elongated form and lack of legs, snakes have compact cross-sections and thus can move through very thin holes and gaps. In addition to their thinner cross section, snakes also have the ability to climb up and over obstacles that are much taller than their body height. These properties are very desirable when moving through complex and cluttered environments.

- Gaits used by snakes for locomotion are very stable. Because their bodies are constantly in contact with the ground at many different points, it is difficult to knock them over, especially since they have a low center of mass and do not lift their bodies off the ground much during locomotion.
- Snakes have redundant designs that rely on the same kind of joint (and structure) that is repeated many times. This means that if one joint fails, the snake can continue to locomote. The simplicity of the design also means that the snake does not have any fragile appendages that can easily break.
- Snakes are very versatile and can be both locomotors and manipulators, as they can use their bodies to wrap around objects to grasp them. Since one structure can do both things, the need for different mechanisms to achieve different tasks is eliminated.
- Despite frictional opposition to their locomotion, snakes actually have been shown to consume a comparable amount of energy to other biological forms with similar sizes, weights and speeds.

2.2 Snake Robots :

Snake-inspired robots were introduced in the early 1970's by Shigeo Hirose. Since that time, numerous snake-inspired robot designs have been conceived and prototyped. Although the various robot designs follow the common theme of mimicking snake locomotion, they may differ greatly in physical configuration and purpose. Some robots use powered wheels or treads, while others may use passive wheels or no wheels at all. Some designs are even amphibious, travelling effortlessly between ground and water environments. Snake- inspired robots have been proposed for missions ranging from exploration to search and rescue to military reconnaissance and surveillance.

2.2.1 Snake Robot Locomotion Gaits :

There are four major snake locomotion gaits: Lateral undulation, Crotaline (Sidewinding), and Rectilinear progression. The majority of snake-inspired robot designs use either lateral undulation or rectilinear progression.

- **Side-winding Motion**

This motion is performed by desert snakes which move side ways by lifting a part of their body from the ground. Hence there are only a few (shifting) points of contact between the snake and the ground. This really helps the snake when it has to move in the desert where the ground is really hot.



Figure 2-1 : A Desert Snake and a Robotic Snake performing Side Winding Motion

- **Lateral Undulation**

This is the typical motion performed by snakes. Here the snake is actually producing a propagating wave motion. The key to the snake moving forward using this motion lies in the an-isotropic friction present in the skin of the snake. The friction coefficient of the snake is high along the direction perpendicular to its body and less in the direction along the body. This difference in friction causes the snake to move forward.

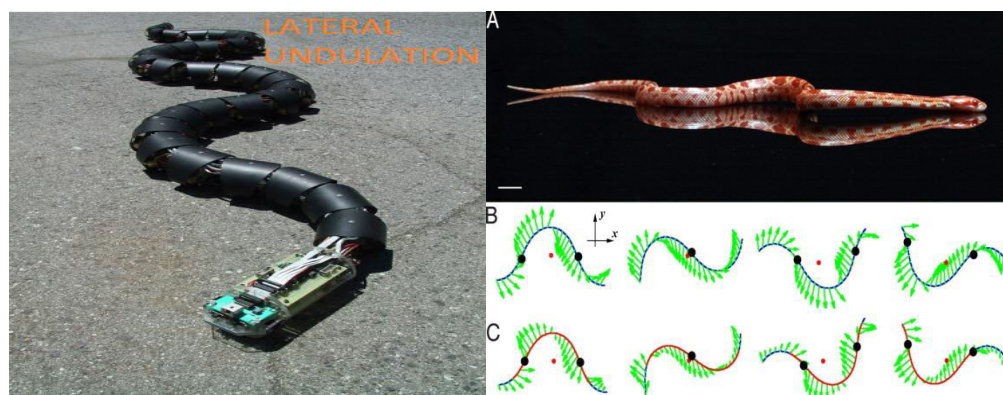


Figure 2-2 : Lateral Undulation performed by Snake Robot and Real Snake.

- **Rectilinear motion**

The slowest mode of snake locomotion is rectilinear locomotion, which is also the only one where the snake does not need to bend its body laterally, though it may do so when turning. In this mode, the belly scales are lifted and pulled forward before being placed

down and the body pulled over them. This motion is also known as Caterpillar Motion as the snake moves like a caterpillar.



Figure 2-3 : Rectilinear motion performed by Snake Robot and Real Snake.

2.2.2 Kinematics of Snake Robot:-

The following section describes the kinematics of a modular snake robot. The snake robot has alternately placed 'horizontal' and 'vertical' motors which basically describe the direction of their rotation axis. To generate the gaits for various motions, a curve fitting algorithm is used. Let θ represent the joint angle of the links. Assume the joint angles to be zero when links are parallel. Then the joint angles in the vertical and horizontal joints can be calculated using the following equation:

$$\theta_{vt} = A_{ver} \sin(\omega_{ver} t + (i - 1)\delta_{ver}) + O_{ver}$$

$$\theta_{ht} = A_{hor} \sin(\omega_{hor} t + (i - 1)\delta_{hor}) + O_{hor}$$

Table 2-1 : Parameters for various Snake Robot Gaits

GAIT	A	Ω	Δ	δ_0	O
Lateral Undulation	$A_{ver}=60^0$ $A_{hor}=0$	$\omega_{ver}=5\pi/6$ s $\omega_{hor}=0$	$\delta_{ver}=2\pi/3$ $\delta_{hor}=0$	$\delta_0=0$	$O_{ver}=0$ $O_{hor}=0$
Sidewinding	$A_{ver}=30^0$ $A_{hor}=30^0$	$\omega_{ver}=5\pi/6$ s $\omega_{hor}=5\pi/6$ s	$\delta_{ver}=2\pi/3$ $\delta_{hor}=2\pi/3$	$\delta_0=0$	$O_{ver}=0$ $O_{hor}=0$
Rolling	$A_{ver}=60^0$	$\omega_{ver}=5\pi/6$ s	$\delta_{ver}=0$	$\delta_0=30^0$	$O_{ver}=0$

	$A_{\text{hor}}=60^0$	$\omega_{\text{hor}}=5\pi/6 \text{ s}$	$\delta_{\text{hor}}=0$		$O_{\text{hor}}=0$
Corkscrew	$A_{\text{ver}}=30^0$	$\omega_{\text{ver}}=5\pi/12 \text{ s}$	$\delta_{\text{ver}}=2\pi/3$	$\delta_0=0$	$O_{\text{ver}}=0$
	$A_{\text{hor}}=60^0$	$\omega_{\text{hor}}=5\pi/6 \text{ s}$	$\delta_{\text{hor}}=2\pi/3$		$O_{\text{hor}}=0$

A = Amplitude; **w** = Frequency of sine wave; **t** = Time; **i** = Link no.; **δ** = phase diff. between two consecutive links in same plane; **δ_0** = phase diff. between 2 adjacent links in horizontal and vertical plane; **O** = default Orientation of the robot ; **ver, hor** = Vertical, Horizontal.

2.3 Applications :

- **Search & Rescue operations** : Snake robots are currently being developed to assist search and rescue teams. Robots need to make their way through the rubble of collapsed structures, as well as archaeological explorations.
- **Navigate on land, water and climb** : The locomotive flexibility of snake robots plays a vital role when a task requires a number of different obstacles to overcome. For example, if a robot is to carry a camera to the top of a tree that is growing in water, it needs to move over ground to the water's edge, swim to the tree, and then climb the tree. Snake robots can perform such difficult combinations.
- **Pipe inspections** :- Snake robots are useful in environments that tends to be long and thin like pipes or highly cluttered like rubble. Robot serves in inspection, maintenance, and repair of a diverse set of pipes and pipe-like structures.
- **Spying**:- A low-cost military robot 'Serpentine Spy' has been developed and can be dropped out of helicopters to carry out reconnaissance missions. The ground-hugging robot make a versatile battlefield spy.
- **Invasion Prevention** :- Snake robots can be used by animal control officers to subdue rabid or invasive creatures. Raccoons, barn cats, and large rodents typically respond to the snake robot's presence with attacks upon which the snake robot will emit an electrical shock and paralyze the aggressor.

3.MOBILE ROBOT IN AIR : QUADCOPTER

3.1 Quadcopter-Definition :

A **quadcopter**, also called a **quadrotor** helicopter or **quadrotor**, is an aircraft that becomes airborne due to the lift force provided by four rotors usually mounted in cross configuration, hence its name. It is an entirely different vehicle when compared with an helicopter, mainly due to the way both are controlled. Helicopters are able to change the angle of attack of its blades, quadcopters cannot.

At present, there are three main areas of quadcopter development: military, transportation of goods and people) and Unmanned Aerial Vehicles (UAVs). UAVs can be classified into two major groups: heavier-than-air and lighter than air. These two groups self divide in many other that classify aircrafts according to motorization, type of liftoff and many other parameters. Vertical take-off and Landing(VTOL). UAVs like quadcopters have several advantages over fixed wings airplanes. They can move in any direction and are capable of hovering and fly at low speed. In addition, the VTOL capability allows deployment in almost any terrain while fixed – wing aircraft require a prepared airstrip for takeoff and landing. Given these characteristics ,quadcopters can be used in search and rescue missions, meterology, penetration of hazardous environments(example exploration of other planets) and other applications suited for such an aircraft. Also, they are playing an important role in research areas like control engineering, where they serve as prototypes for real life applications.

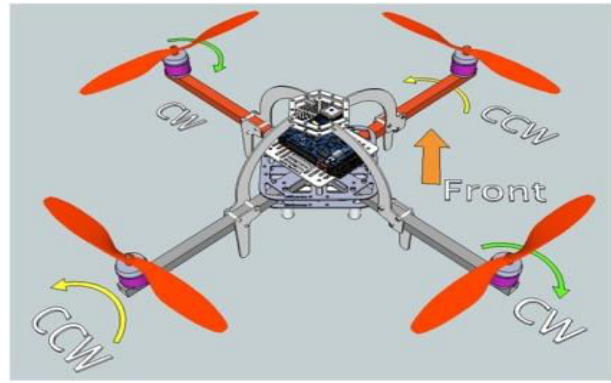


Figure 3-1 : Quadcopter fitted with a Camera and its Model

3.2 Flight Control & Balancing:

An airborne aircraft has no external support. So it has no reaction external forces and body forces. It has to be self-balanced and should cancel all the forces and torques arising internally as well as externally. Thus as detailed Flight Dynamics Analysis is necessary.

Flight dynamics is the study of the performance, stability and control of vehicle flying through the air. It is concerned with how forces acting on the vehicle influence its speed and attitude with respect to time. It is the science of air vehicle orientation and control in three dimensions, the three critical flight dynamics parameters are the angle of rotation in three dimensions about the vehicle's centre of mass, known as **roll, pitch and yaw**.

Balancing of the forces, the roll, pitch and yaw orientations should be easily controlled.

Most common type of the rotorcrafts is the helicopter, which has a single large rotor. Because of the torque generated, unbalance is created in yaw direction & the rotor tends to rotate the aircraft body in opposite direction. To balance this torque, helicopter has a tail rotor which creates opposing torque. Quadcopters, on the other hand has 4 rotors whose speed is needed to be adjusted accordingly to change above parameters.

3.3 Working of Quadcopter:

Each rotor in a quadcopter is responsible for a certain amount of thrust and torque about its centre of rotation, as well as for a drag force opposite to the rotorcraft's direction of flight.

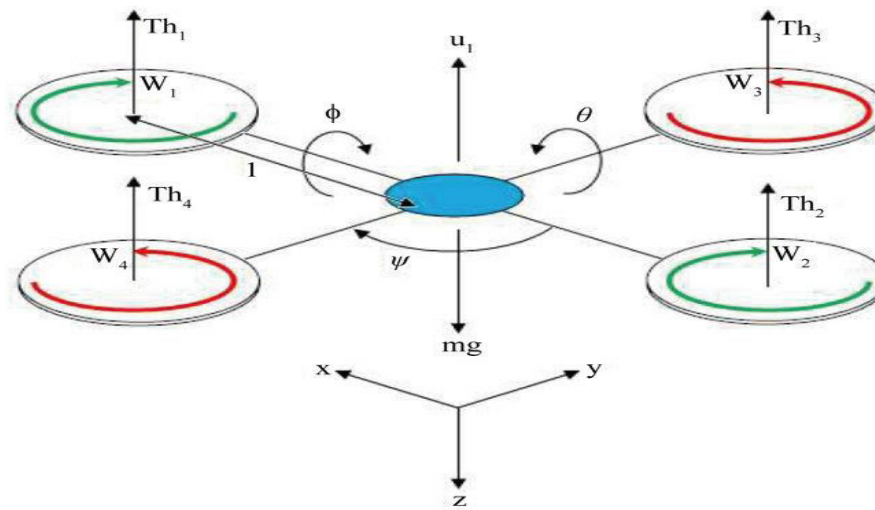


Figure 3-2 : Mathematical Model of a Simple Quadcopter in X configuration.

The quadcopter's propellers are not all alike. In fact, they are divided in two pairs, two pusher and two puller blades, that work in contra-rotation. As a consequence, the resulting net torque can be null if all propellers turn with the same angular velocity, thus allowing for a aircraft to remain still around its centre of gravity.

In order to define an aircraft's orientation (or attitude) around its centre of mass, aerospace engineers usually define three dynamic properties, the angles of yaw, pitch and roll. Roll, pitch and yaw refer to rotations about the respective axes starting from a defined equilibrium state.

This is very useful because the forces used to control the aircraft act around its center of mass, causing it to pitch, roll or yaw. Changes in the pitch angle are induced by contrary variation of speeds in Forward and Backward pairs of propellers, resulting in forward or backwards translation. If we do this same action for Right and Left pairs of propellers, we can produce a change in the roll angle and we will get lateral(right/left) translation. Yaw is induced by mismatching the balance in aerodynamic torques. So, by changing these three angles in a quadcopter we are able to make it manoeuvre in any direction.

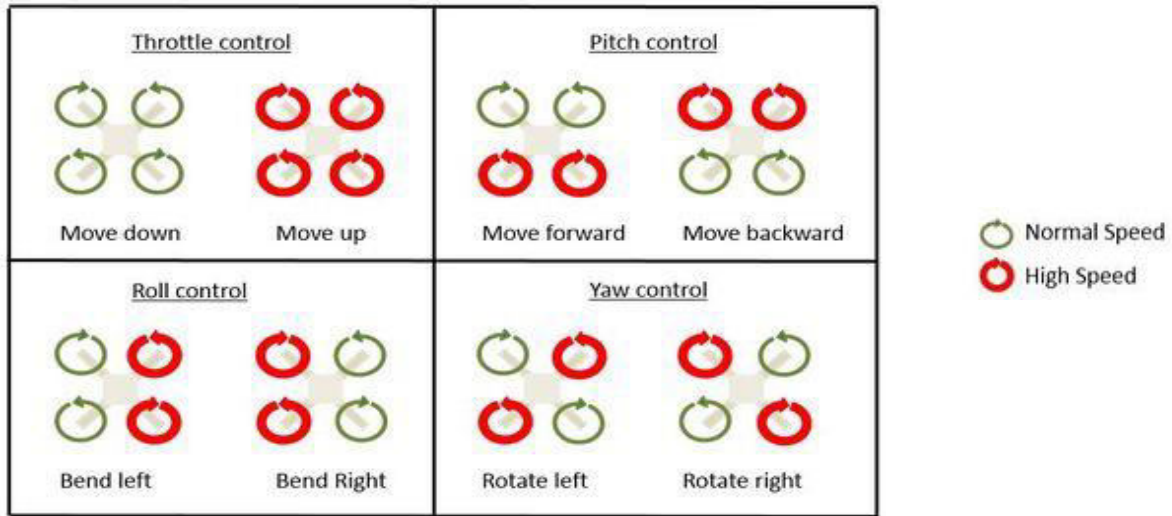


Figure 3-3 : Control of Various Parameters by Motors' Speed Variation in Quadcopter.

3.4 Mathematical analysis and Control Strategy Applied for the Quadcopter:

The quadcopter has six degrees of freedom that can be divided into two parts:

- Translational- translational motion occurs in x,y,z directions.
- Rotational- rotational motion occurs about x,y,z directions and named as roll(Φ), pitch (θ) and yaw (ψ) respectively.

The control system for quadcopter include 4 actuators, which exert forces in various directions, and generate rotational forces or moments about the center of gravity of the aircraft, and thus rotate the aircraft in pitch, roll or yaw, as well as giving net upward thrust to it. The thesis present a simple control structure for the quadcopter to be controlled by an external commanding unit(Remote Controller), thus leading to a stable Semi-Autonomous flight.

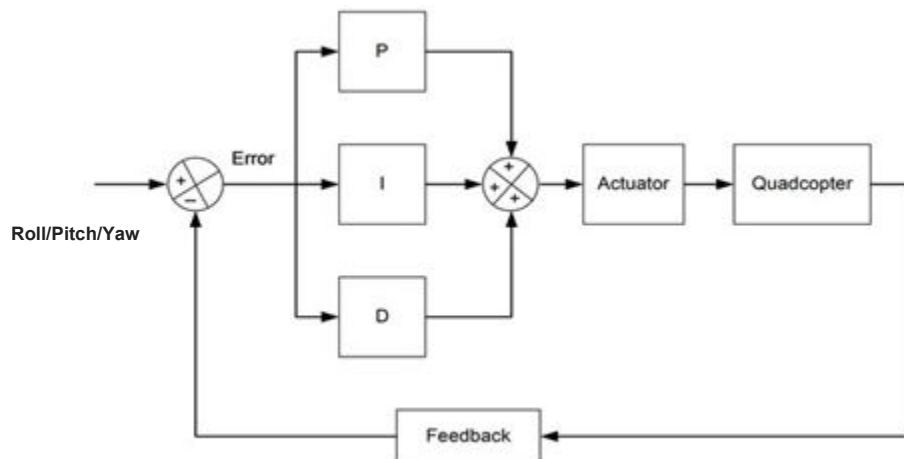


Figure 3-4 : PID control loop for one control parameter of Quadcopter.

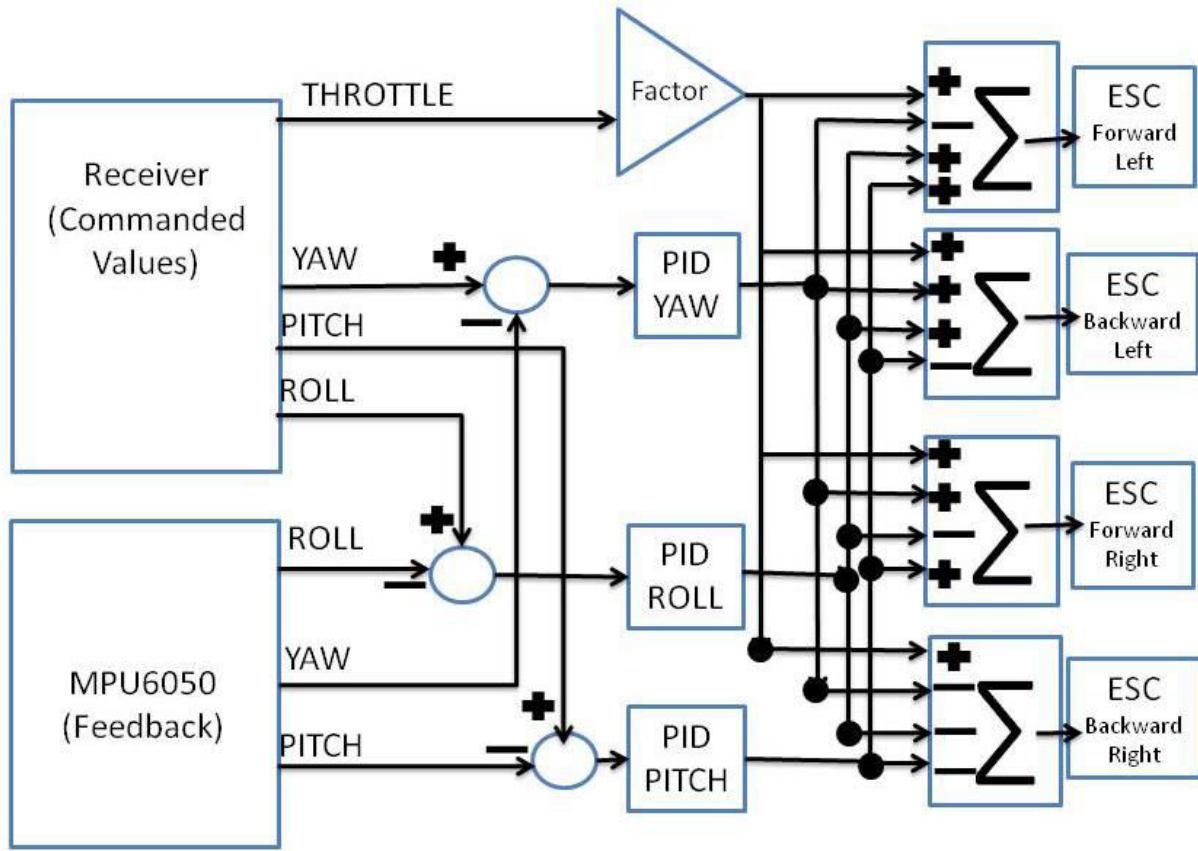


Figure 3-5 : Cascaded PID control structure, Involves all Control Parameters of Quadcopter.

Thus, Different Motors (which are controlled by ESCs) receive following speeds:

Motor-Forward-Left = map((throttle + roll_output + pitch_output - yaw_output).

Motor-Backward-Left = map((throttle + roll_output - pitch_output + yaw_output).

Motor-Forward-Right=map(throttle - roll_output + pitch_output + yaw_output).

Motor-Backward-Right= map(throttle - roll_output - pitch_output - yaw_output).

[Mapping is needed to be done for getting values in the ESC input range. The control of motors used in quadcopter and their control via ESCs is discussed later in the thesis]

Above, the **roll_output**, **pitch_output**, **yaw_output** implies the PID output for roll, pitch and yaw respectively which is given by :

$$X_{\text{Output}}(t) = K_P * X_{\text{Error}}(t) + \left(K_I * \int_0^t X_{\text{Error}}(\tau) * d(\tau) \right) + K_d * \frac{\partial X_{\text{Error}}(t)}{\partial t}$$

$$X_{\text{Error}}(t) = X_{\text{Commanded}}(t) - X_{\text{Actual}}(t)$$

(substituting X by roll, pitch or yaw)

Kp, Ki, Kd are the **coefficients for Proportional, Integral and Differential** respectively.

3.5 Applications:

- **Research Platform** : Quadcopters are a useful tool for university researchers to test and evaluate new ideas in a number of different fields, including Flight Control Theory, navigation, real time systems, and robotics.
- **Military** : Militaries around the world use unmanned combat air vehicle for routine surveillance and to carry out basic attack strategies. Quadcopters are currently used in the real world for surveillance and reconnaissance by law enforcement agencies and the military as well as search and rescue missions in urban environments.
- **Commercial** : The largest use of the quadcopters has been in the field of aerial imagery. Quadcopter UAVs are suitable for this job because of their autonomous nature and huge cost savings.
- **Recent Developments** : In the last few decades, small scale UAVs have become more commonly used for many applications. The need for aircrafts with greater manoeuvrability and hovering ability has led to current rise in quadcopter research.

4. DESCRIPTION OF THE FLYING SNAKE ROBOT

Design, Development and Control of an Autonomous Modular Snake Robot reconfigurable into Quad-Copter aka **Flying Snake Robot** involves imparting flying properties to a snake robot along with its land locomotion.

4.1 Basic Concept :

Snake Robots resemble biological snakes and their small cross section to length ratio allows them to move into and manoeuvre through tight spaces. Their ability to change body shape allows them to perform a wide range of behaviours, such as climbing stairs or tree trunks.

Snake robots can be used to explore unknown terrains but the navigation can include complex gait performance like climbing stairs, countering obstacles by going around them. In order to make snake navigation simpler we aim to impart flying properties along with its locomotion.

Our design includes eight actuators for snake locomotion and four BLDC motors. The robot can perform a variety of standard snake locomotion gaits and configures itself into a 'square' shape to fly as a quadcopter.

Thus, the invention combines the advantages of both quadcopter and snake robot. The following tables illustrate the advantages of both snake Robots and quadcopters which are aimed to be achieved by their integration as Flying Snake Robot, as well as their disadvantages which are aimed to be overcome.

Table 4-1: Advantages and Disadvantages (in Focus) of Snake Robots.

Advantages	Disadvantages
Snake robots can move across uneven terrain, since it is not dependent on wheels.	Snake robots are much slower than natural rivals and wheeled robots.
Snake robots can be very modular with many redundant segments.	Difficult to control high number of degrees of freedom.
Snake robots due to high manoeuvrability can go thru compact/small spaces like holes, crevices easily.	There is no integral platform for attaching payloads.

Table 4-2: Advantages and Disadvantages (in Focus) of Quadcopters.

Advantages	Disadvantages
Very high navigating speeds are easily achievable.	Battery power is restricted, thus very less 'Operational Time' as compared to other robots.
Less complicated	Symmetric structural design is preferred along with proper mass distribution.

4.2 Components Involved :

This section discusses various components involved in the Flying Snake Robot.

4.2.1 Brushless Motors- BLDC :

Brushless DC motors (BLDC) are a bit similar to normal DC motors in the way that coils and magnets are used to drive the shaft. Though the BLDC motors do not have a brush on the shaft which takes care of switching the power direction in the coils, and this is why they are called brushless. Instead the BLDC motors have three coils on the inner (centre) of the motor, which is fixed to the mounting. On the outer side it contains a number of magnets mounted to a cylinder that is attached to the rotating shaft. So the coils are fixed which means wires can go directly to them and therefore there is no need for a brush.

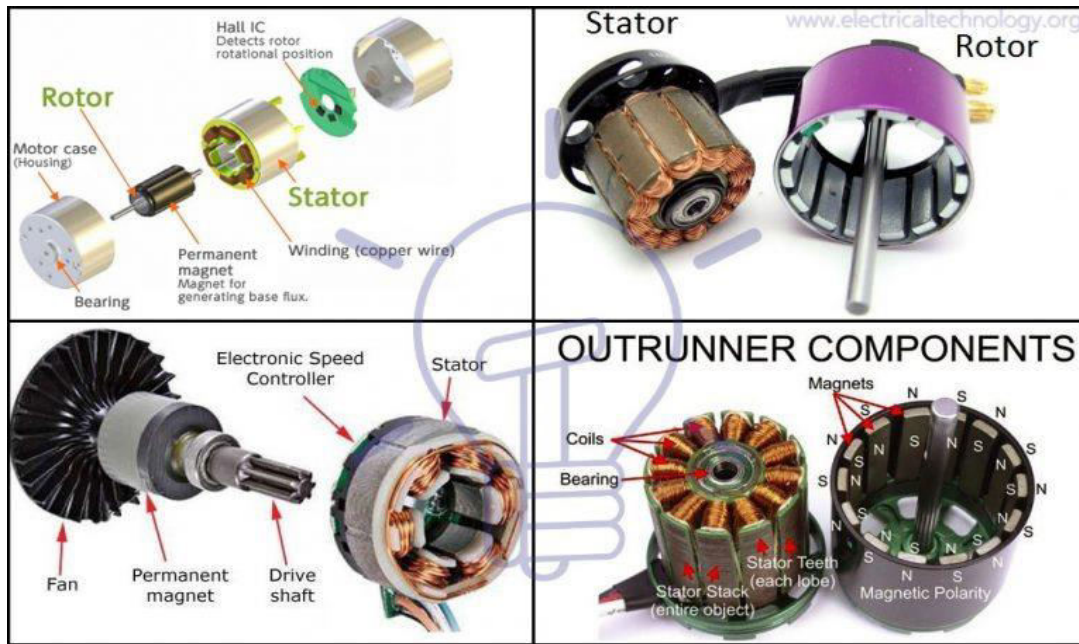


Figure 4-1: Construction of BLDC Motor

BLDC motors offer several advantages over brushed DC motors, including more torque per weight, increased reliability, reduced noise, longer lifetime (no brush and commutator erosion), elimination of ionizing sparks from the commutator, and overall reduction of electromagnetic interference (EMI). With no windings on the rotor, they are not subjected to centrifugal forces, and because the windings are supported by the housing, they can be cooled by conduction,

requiring no airflow inside the motor for cooling. This in turn means that the motor's internals can be entirely enclosed and protected from dirt or other foreign matter.

Quadcopters use BLDC motors because of much higher speeds and less power usage. The BLDC motors are more efficient as there is no power lost as there is in the brush-transition on the DC motors.

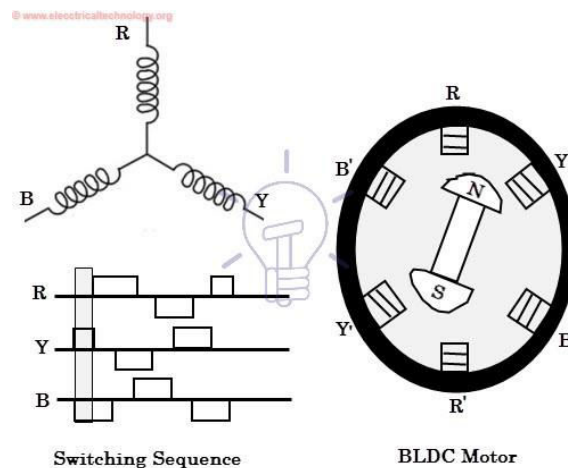


Figure 4-2: Working of BLDC Motor

In this diagram of the brushed motor we see that it is the coil that is rotating instead of the magnets as on the brushless.

Brushless motors come in many different varieties, where the size and the current consumption differ. When selecting the brushless motor we took care of the weight, the size & the kind of propeller we were going to use, so everything matched up with the current consumption.

Two key performance parameters of BLDC motors are the Motor constants K_v and K_m (which are numerically equal in SI units). The K_v ratings is an indication on how many RPMs the motor will do if provided with x-number of volts.

Advantages of BLDC Motor:

- High efficiency due to the use of permanent magnet rotor.
- High speed of operation even in loaded and unloaded conditions due to the absence of brushes that limits the speed.

- Smaller motor geometry and lighter in weight than both brushed type DC and induction AC motors.
- Long life as no inspection and maintenance is required for commutator system.

Disadvantages of BLDC Motor:

- Electronic controller required control this motor is expensive.
- Not much availability of many integrated electronic control solutions, especially for tiny BLDC motors.
- Requires complex drive circuitry.

Specifications of the motors used:

- Number of Motors: 4
- No Load Current: 10 V : 0.5 A
- Current Capacity: 12A / 60s
- Motor Dimension: 27.5 x 27mm , Shaft Diameter: 3.17mm
- KVA Rating(Power): 2200kva

4.2.2 Propellers :

On each of the brushless motors propellers are mounted. The four propellers are actually not identical. One pair of the diametrically opposite propellers is right tilted, while the other pair is left tilted. This corresponds to upward thrust produced by them when rotated clockwise or anti-clockwise. The propellers are generally organized by their “**numbers**”. These numbers look like “5045” or “5×4.5×3”. The first number (“5”) is the size of the prop in inches. The second number set (“45” or “4.5”) indicates the pitch of the prop in inches. In the former case, the number needs to be divided by 10. The last number (“x3”) will specify the number of blades on the prop. In many cases this is omitted and the prop is referred to as a “tri-blade” or “quad-blade”. If the number is followed by character 'R' (5X4R) as shown in Fig .. , this implies that the propeller is clockwise rotating, i.e. it produces upward thrust when rotated clockwise, else it is counter-clockwise rotating.



Figure 4-3 : *Numbering on a Propeller*

The flight is effected by the size, pitch, blade count and chord of propeller, as explained :

- **Size**

This is simply the length from tip to tip of the prop in inches. The prop size is restricted to the sizes the frame supports. For instance, a frame that supports 6” props will support 6” props and down, but will not support 7” props. Longer propellers generate more thrust when rotated at the same speed. This means faster acceleration but also means more power being pulled from the same motor. Extra length does not necessarily mean faster fly-speed – that is more determined by pitch. Shorter propellers are able to spin up and slow down faster, which translates to increased agility of the aircraft. Mini-quads almost exclusively use 4”, 5” and 6” props.

- **Blade Pitch**

This is the “bite” of the propeller, specifying how much air it “screws” through in one full revolution. More pitch means you have more thrust when your aircraft is travelling at high speed, but also means you have decreased thrust when it is not moving. An aircraft can move no faster than the pitch of the prop allows it to. The speed at which propeller will generate zero thrust can be determined with the following equation: $\text{MaxRPM} * \text{Pitch} / 720$. This will give the maximum speed of aircraft in feet per second. The multirotor will be much slower than that calculated speed due to its high drag – probably something like 50% of that number.

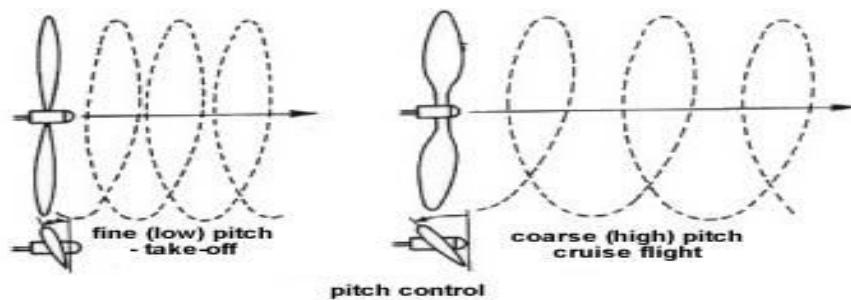


Figure 4-4 : Effect of Pitch on the Flight

Choosing a propeller with lower pitch improves the aircraft stability. A higher pitch propeller displaces a greater amount of air, creating turbulence and causes the aircraft to wobble during hovering. Choosing a lower pitched propeller reduced the amount of turbulence.

- **Number of Blades**

Adding number of blades on the propeller increases the amount of thrust it generates but generally makes it less efficient at creating that thrust. This is similar to going to a longer propeller – but in the case of using multiple blades, one doesn't have to deal with the longer blades. This is why most WWII airplanes had multi-bladed props – their props were already so massive that they were within inches of the ground when the airplane was on the runway. Miniquad pilots lately have migrated in droves to 5" tri-blade props since the operating environment of miniquads negates the inefficiencies of tri-blade propellers such that three blades are simply better than two blades. Tri blades also have a torque curve which makes the yaw axis of a quadcopter more responsive.

- **Chord Size**

The chord of the propeller is the relationship between the average width of the blade and its overall length. In airplane propeller design, the tip of the propeller blade is made smaller than the rest of the propeller to balance the thrust load across the blade (remember – the tips move faster than the rest of the propeller). Quadcopter pilots care less about the load balance of the propeller and more about the overall thrust it generates. As a result, drone pilots have found an easy way to create a propeller that generates more thrust was to buy propellers that were 1 or 2 sizes too big, then cut off the thin tip of the propeller – creating

a big, fat propeller in a smaller size. This style of propeller is called “bullnose”. Currently manufacturers have begun selling this style of propellers direct from the factory too. Bullnose propellers have a very distinct look as shown in Figure 4-5.

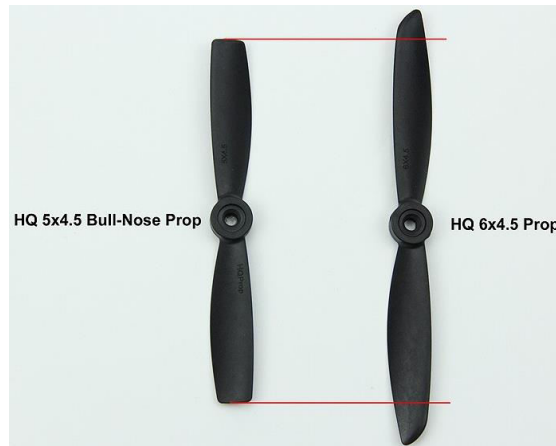


Figure 4-5 : Comparison between Bullnose and Standard propeller

They can generally create the highest thrust levels for their pitch, size and blade-count. Similar to tri-blades, the extra weight on the tips of the blades increases the torque required to spin them and improves the responsiveness of the yaw axis on the quadcopter. Unfortunately these positive aspects come with a cost – they are considerably less efficient than standard propellers and will therefore stress power system more and result in lower flight times.

Specifications of the propellers used:

- Number of Propellers: 4
- Number of blades: 2 (per Propeller)
- Type and Size: Standard Propeller and 6x4.5.

4.2.3 Electronic Speed controllers

As the brushless motors are multi-phased, normally 3 phases, one can't just apply power to it & make it spin. The motors requires a special phase-control electronics that is capable of generating three high frequency signals with different but controllable phases, but the electronics should also be able to source a lot of current as the motors can be very power-hungry.

In this case we have got the Electronic Speed Controllers, known as ESCs. The ESCs is simply a brushless motor controller board with battery input and a three phase output for the motors. For the control it is usually just a simple PPM (pulse position modulation) signal that ranges from 1 ms (min speed=turn off) to 2ms (max speed) in pulse width.

ESCs can be found in many different variants, where the source current is the most important factor. Brushless ESC systems basically drive tri-phase brushless motors by sending a sequence of signals for rotation. The correct phase varies with the motor rotation, which is to be taken into account by the ESC: Usually, back EMF from the motor is used to detect this rotation, but variations exist that use magnetic(Hall Effect) or optical detectors. Computer-programmable speed controls generally have user-specified options which allow setting low voltage cut-off limits, timing, acceleration, braking and direction of rotation. Reversing the motor's direction may also be accomplished by switching any two of the three leads from the ESC to the motor.

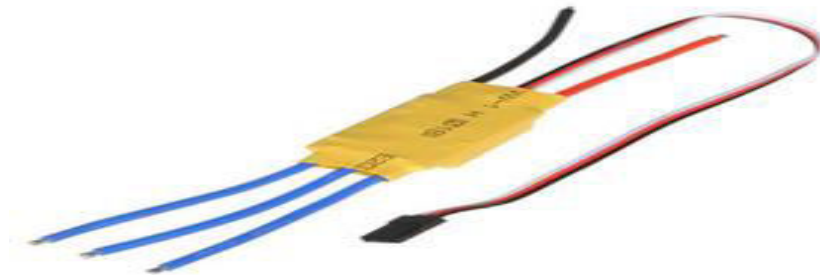


Figure 4-6: An Electronic Speed Controller

Specifications of the ESCs used:

- Input voltage: DC 6-16.8V(2-4S Lipo) ;
- Size: 26mm (L) * 23mm (W) * 11mm (H); Weight: 32g.
- Running current:30A (Output: Continuous 30A, Burst 40A up to 10 Secs.)

4.2.4 Arduino Uno:

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 Mhz ceramic resonator, USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller. We connect it to a computer with a

USB cable or power it with a AC-to-DC adapter or battery. This board features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

In this project, we use the Arduino Board to power the auadcopter. We get the readings of the current position, velocity and orientation of the quadcopter from the IMU. We fine tune the readings from the PID algorithm and send the corrected values to the rotors. This process of reading the values from the IMU and applying PID are done by the Arduino Board. Thus a **Customized Flight Controller** is implemented through **Arduino and MPU6050**.

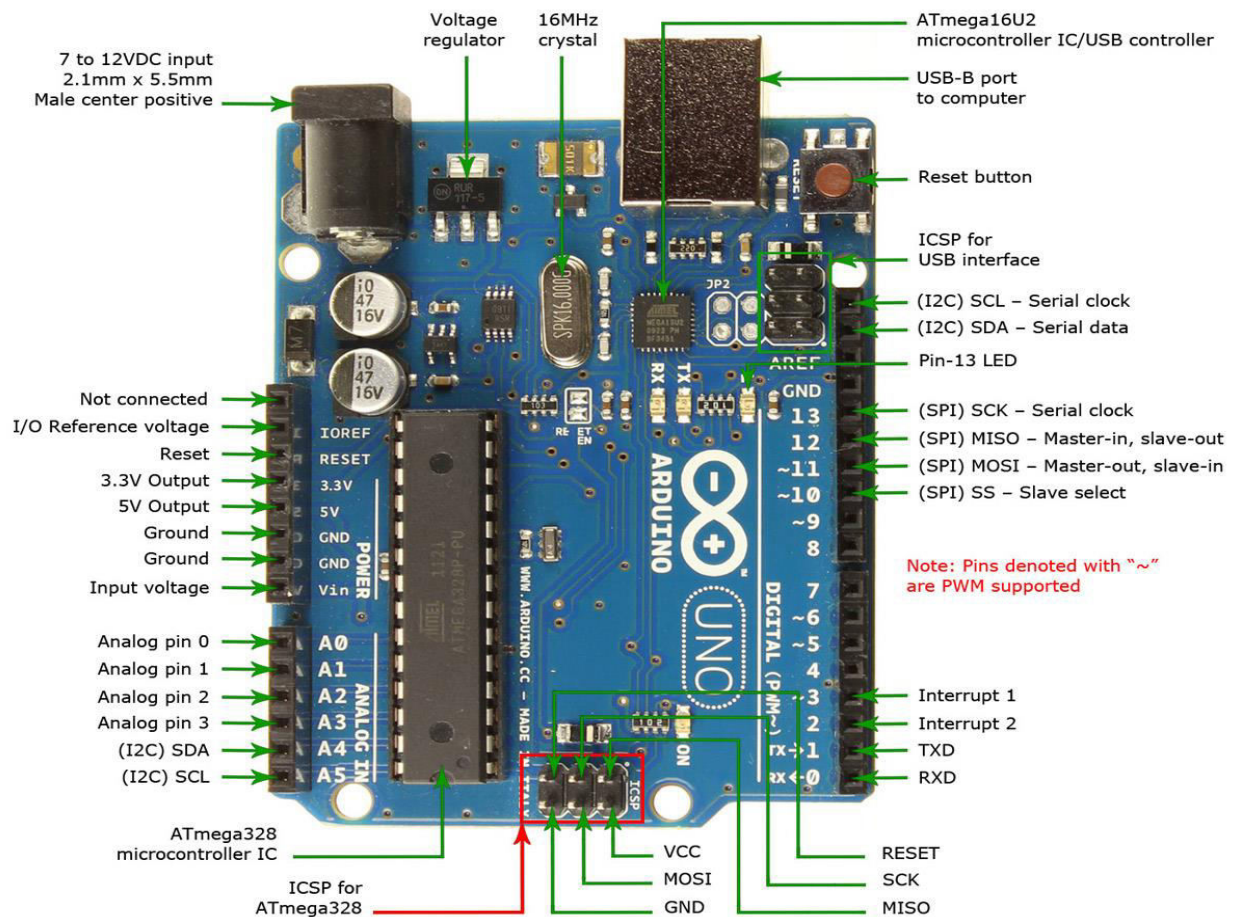


Figure 4-7: Arduino Uno Board with Detailed Pin Layout

4.2.5 Inertial Measurement Unit

An inertial measurement unit (IMU) is an electronic device that measures and reports on a craft's velocity, orientation, and gravitational forces, using a combination of accelerometers and

gyroscopes, sometimes also magnetometers. IMUs are typically used to manoeuvre aircraft, including satellites and landers.

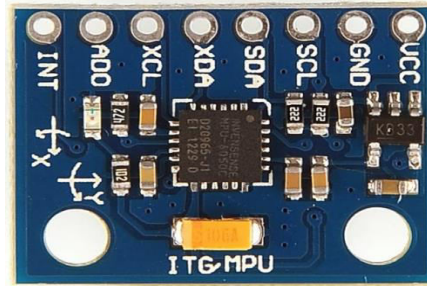


Figure 4-8: IMU- MPU6050

The IMU used for the project is the MPU6050 which contains both a 3-Axis Gyroscope and a 3-Axis accelerometer allowing measurements of both independently, but all based around the same axes, thus eliminating the problems of cross-axis errors when using separate devices.

Specifications

- Accelerometer ranges: ± 2 , ± 4 , ± 8 , $\pm 16g$
- Gyroscope ranges: ± 250 , 500 , 1000 , 2000 $^{\circ}/s$
- Voltage range: $3.3V - 5V$ (the module include a low drop-out voltage regulator)

This simple module contains everything required to interface to the Arduino and other controllers via I2C (use the Wire Arduino library) and give motion sensing information for 3 axes - X, Y and Z..

4.2.6 Dynamixel AX-12A

The AX-12A servo actuator from Robotis is the most advanced actuator in this price range and has become the defacto standard of hobby robotics. The AX-12A robot servo has the ability to track its speed, temperature, shaft position, voltage, and load. The control algorithm used to maintain shaft position on the AX-12A actuator can be adjusted individually for each servo, allowing control of the speed and strength of the motor's response. All of the sensor management and position control is handled by the servo's built-in microcontroller. This distributed approach leaves main controller free to perform other functions.

Specifications of Dynamixel AX-12A used:

- **Weight** : $54.6g$

- **Dimension** : 32mm * 50mm * 40mm.'

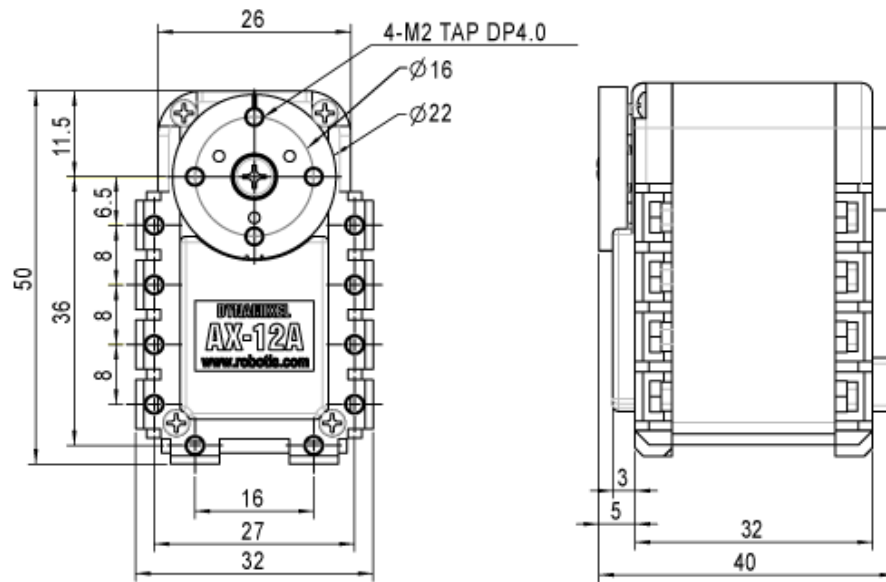


Figure 4-9 : Detailed Dimensions of Dynamixel AX-12 Actuator.

- **Resolution** : 0.29°.
- **Gear Reduction Ratio** : 254 : 1
- **Stall Torque** : 1.5N.m (at 12.0V, 1.5A), Stall torque is the maximum instantaneous and static torque, Stable motions are possible with robots designed for loads with 1/5 or less of the stall torque.
- **No load speed** : 59rpm (at 12V)Running Degree.

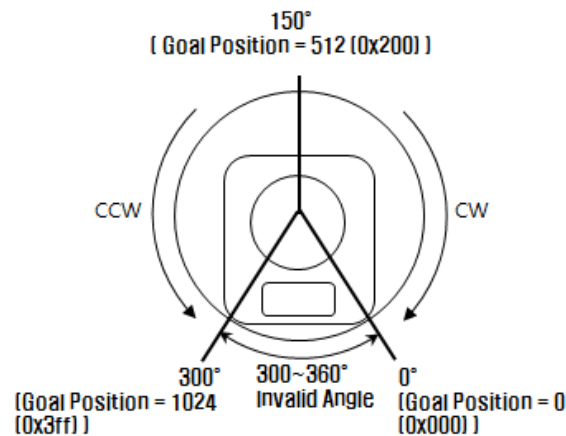


Figure 4-10 : Various Goal positions and values for Dynamixel.

Important parameters of dynamixel to keep in mind while using them:

- **ID :-** It is a unique number to identify dynamixel.
- **Max Torque -** It is the torque value of maximum output. 0 to 1023 (0x3FF) can be used.
- **Goal Position -** It is a position value of destination. 0 to 1023 (0x3FF) is available.
- **Moving Speed -** It is a moving speed to goal position.
- **Torque Limit -** It is the value of the maximum torque limit.

In the Flying Snake Robot eight Dynamixel AX-12A actuators have been equally distributed for snake motion in horizontal and vertical plane, being placed alternatively. The Actuators receive commands from OpenCM-9.04 and accordingly the snake gaits are performed as described in Chapter-2 under Section 2.2 .

4.2.7 Robotic OpenCM9.04

OpenCM9.04 is an open-source controller that runs under 32bit ARM Cortex-M3. It is possible to use the 3PIN connector to control ROBOTIS Dynamixels that supports TTL communication. The board is easily programmable with [ROBOTIS OpenCM], an Arduino-like IDE that allows the user to easily program in C/C++. Also the additional JTAG / SWD terminal can be used run commercial developmental programs.

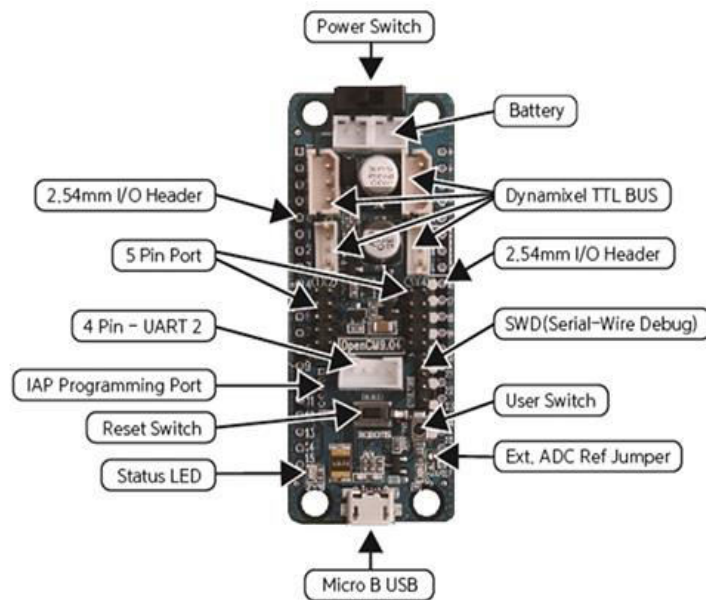


Figure 4-11: Detailed OpenCM-9.04 Board

4.2.8 LiPo battery

Lithium Polymer batteries (LiPo batteries), are a newer type of battery used in many consumer electronics devices. LiPo batteries offer a wide array of benefits. Let's first talk about the differences between LiPo batteries and their Nickel-Cadmium and Nickel-Metal Hydride counterparts.

LiPo batteries offer three main advantages over the common Nickel-Metal Hydride (NiMH) or Nickel Cadmium (NiCd) batteries:

- LiPo batteries are much lighter weight, and can be made in almost any size or shape.
- LiPo batteries offer much higher capacities, allowing them to hold much more power.
- LiPo batteries offer much higher discharge rates, meaning they pack more punch.

But, just as a coin has two sides, there are some drawbacks to LiPo batteries as well:

- LiPo batteries have a shorter lifespan than NiMH/NiCd batteries, averaging only 300–400 cycles.
- The sensitive chemistry of the batteries can lead to fire if the battery gets punctured and vents into the air.
- LiPo batteries need special care in the way they are charged, discharged, and stored. The required equipment can be expensive.



Figure 4-12: A LiPo battery with various Ratings

The way any battery is defined is through a ratings system. This allows to compare the properties of a battery and help us determine which battery pack is suitable for the need at hand. There are three main ratings that one shall be aware of on a LiPo battery:

- Voltage / Cell Count:** A LiPo cell has a nominal voltage of 3.7V. For the 7.4V battery above, that means that there are two cells in series (which means the voltage gets added together). So a two-cell (2S) pack is 7.4V, a three-cell (3S) pack is 11.1V, and so on. Nominal voltage is the default, resting voltage of a battery pack. This is how the battery industry has decided to discuss and compare batteries. It is not, however, the full charge voltage of the cell. LiPo batteries are fully charged when they reach 4.2v/cell, and their minimum safe charge, as we will discuss in detail later, is 3.0v/cell. 3.7v is pretty much in the middle, and that is the nominal charge of the cell. Voltage directly influences the RPM of the electric motor, thus it essentially determines how fast the vehicle flies.
- Capacity:** The capacity of a battery is basically a measure of how much power the battery can hold. The unit of measure here is milliamp hours (mAh). This is saying how much drain can be put on the battery to discharge it in one hour. Since, usually the drain of a motor system is discussed in amps (A), here is the conversion: **1000mAh = 1 Amp Hour (1Ah)**. The higher the number, the longer the run time, but the bigger the capacity, the bigger the physical size and weight of the battery. Another consideration to be noted is heat build up in the motor and speed control over such a long run.
- Discharge Rating ("C" Rating):**
 Voltage and Capacity had a direct impact on certain aspects of the vehicle, whether it's speed or run time. The C Rating is simply a measure of how fast the battery can be

discharged safely and without harming the battery. (e.g. $50C = 50 \times \text{Capacity in Amps}$) Thus, by the C-Rating of the battery in Figure 4-12, Maximum Discharge: $50 \times 5 = 250A$. The resulting number is the maximum sustained load that can be safely put on the battery. Going higher than that will result in, at best, the degradation of the battery at a faster than normal pace. At worst, it could burst into flames.

Most batteries today have two C Ratings: a Continuous Rating (discussed above), and a Burst Rating. The Burst rating works the same way, except it is only applicable in 10-second bursts, not continuously. The Burst Rating is almost always higher than the Continuous Rating. Batteries are usually compared using the Continuous Rating, not the Burst Rating.

Specifications of the LiPo used:

- Output voltage: DC 10-12 V(3S LiPo) ;
- Size: 23mm (L) * 33mm (W) * 105mm (H); Weight: 170g.
- Capacity: 2200mAh.
- C Rating: 35

4.2.9 Flight Controller

A flight controller (a.k.a FC) is the brain of the aircraft. It is basically a circuit board that has built-in sensors that detects orientation changes. It also receives user commands, and controls the motors in order to keep the quadcopter in the air. Nearly all flight controllers have basic sensors such as Gyro (Gyroscopes) and Acc (Accelerometer). Some FC might include more advanced sensors such as Barometer (barometric pressure sensors) and magnetometer (compass).

The Flight controller used for this project was NAZE32.

At the heart of the Naze32 is a 32bit powerful processor, with untapped memory and CPU power and a host of equally impressive sensors. The Naze is also matched up with some of the nicest GUI programs and features. The "Acro" version comes with added MS5611 barometer and a magnetometer.

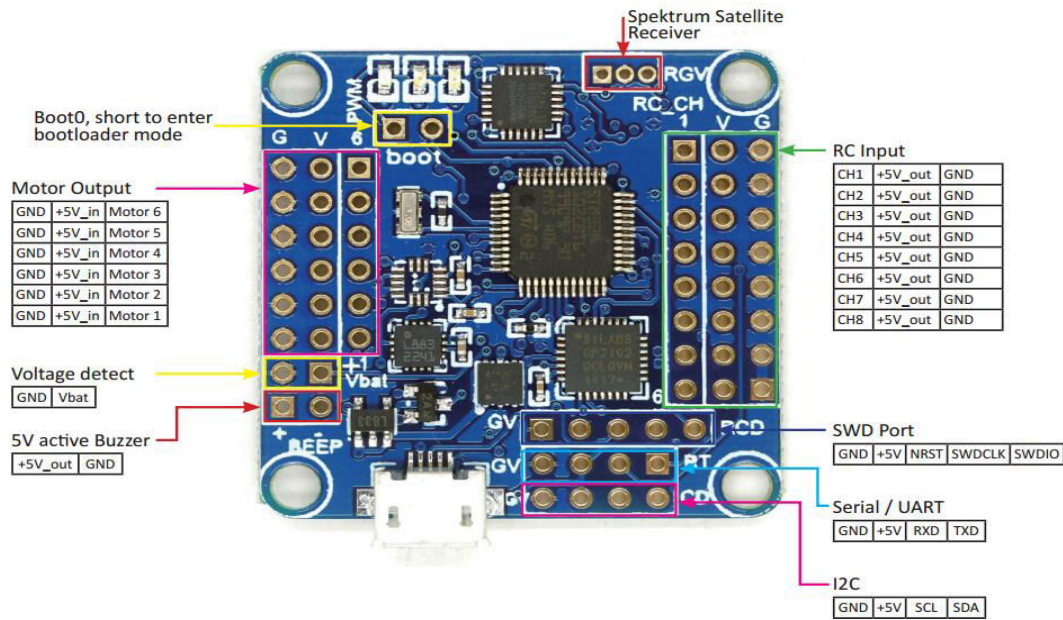


Figure 4-13: Naze32 board with detailed Pin layout

Specifications:

- STM32F103CBT6 32-bit ARM Cortex M3 processor (72MHz, 3.3V)
- Invensense MPU6500 MEMS accelerometer + gyro
- BMP280 Barometer
- Built-in Micro-USB for programming/firmware update
- 36x36mm size, ~5 gram weight w/o headers

The Flight Controller enabled initial testing of the quadcopter configuration of the Flying Snake Robot. This was also used while dealing with the challenge of unbalanced mass in above configuration which is discussed in later sections.

4.2.10 (Flysky)FS-T6 Transmitter and Receiver

This is a 2.4GHz , Digital Proportional 6 Channel Transmitter and Receiver System. It includes all of the important features and programmable parameters to support a variety of helicopters, gliders, and airplanes. It even has the ability to store 20 models allowing the user to save a whole fleet of aircrafts.

The Transmitter is used to command the Flying Snake Robot through receiver, thus leading to Semi-Autonomous nature of robot. The Arduino board receives the values directly from receiver and thus decides whether the robot would be in the snake or the quadcopter configuration and perform further actions.



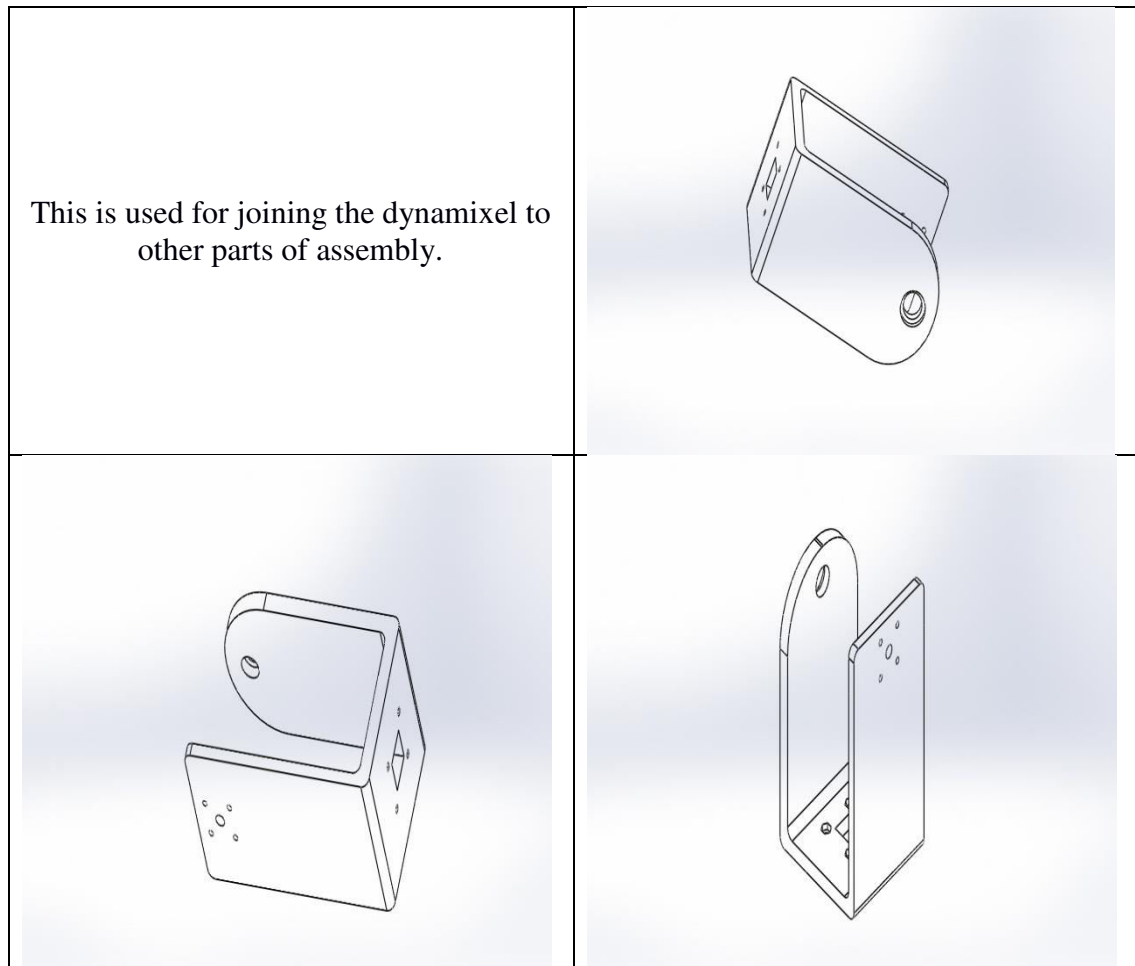
Figure 4-14 : FS-T6 6 Channel Transmitter and Receiver

4.3 Mechanical Parts and Assembly:

4.3.1 Parts: The robot uses customized parts to assemble together the Dynamixels and BLDCs. The parts are customizable, self designed via CAD and 3D printed. The basic **Parts** are:

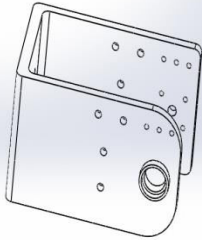
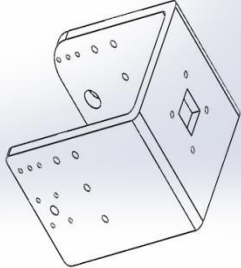
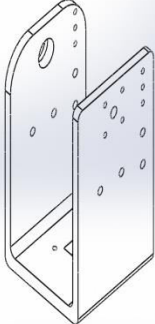
- **Simple Connector for Dynamixel :**

Table 4-3: Description and different Views of CAD model of 'Simple Connector for Dynamixel'



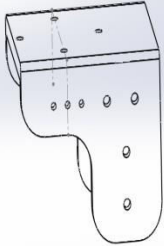
- **Special Connector for Dynamixel:**

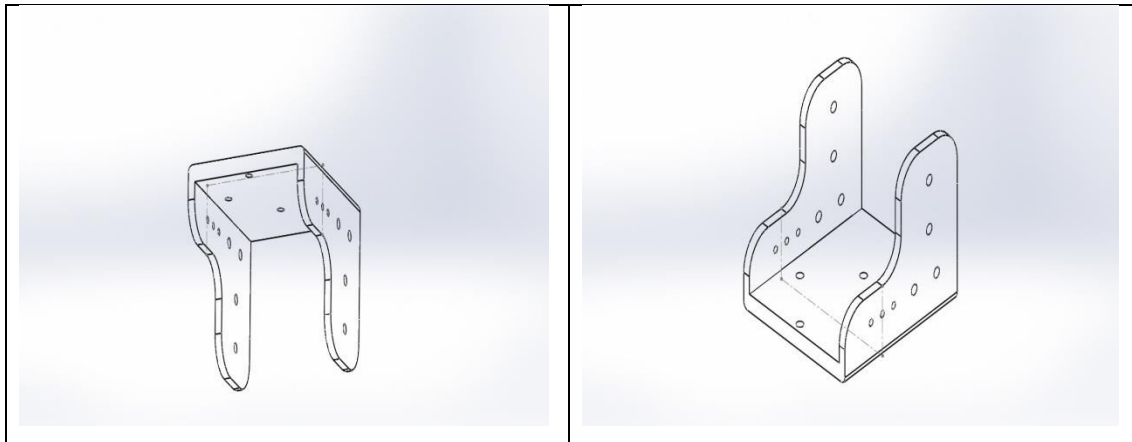
Table 4-4 : Description and different views of CAD model of 'Special Connector for Dynamixel'

<p>This is also used for joining the Dynamixel to other parts of assembly as well as mounting the <u>BLDC+BLDC connector</u> sub-assembly over itself.</p>	
	

- **BLDC Connector**

Table 4-5 : Description and different views of CAD model of 'BLDC Connector'

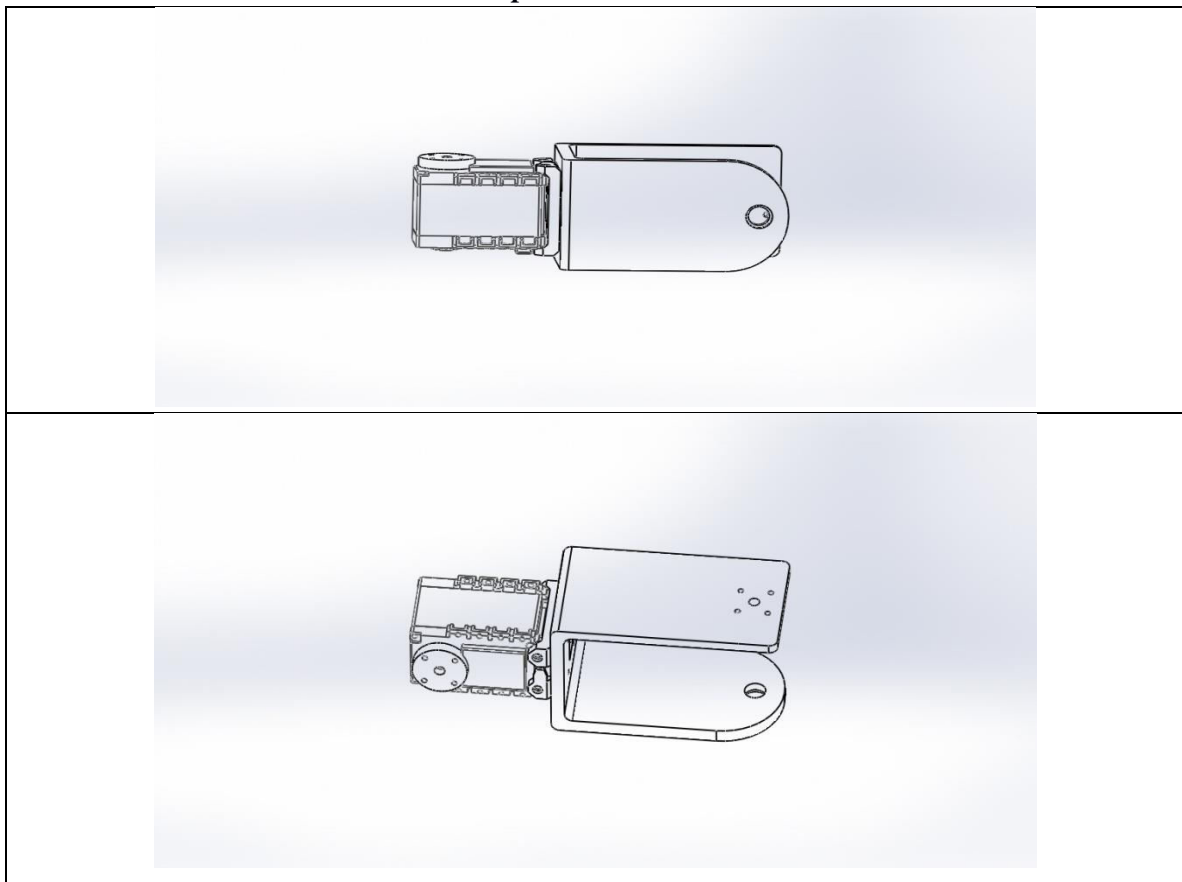
<p>This is used for connecting the BLDC to be mounted over the assembly.</p>	
--	--



4.3.2 Assemblies : Series of **Sub-assemblies** lead to the **final full assembly** of the Robot:

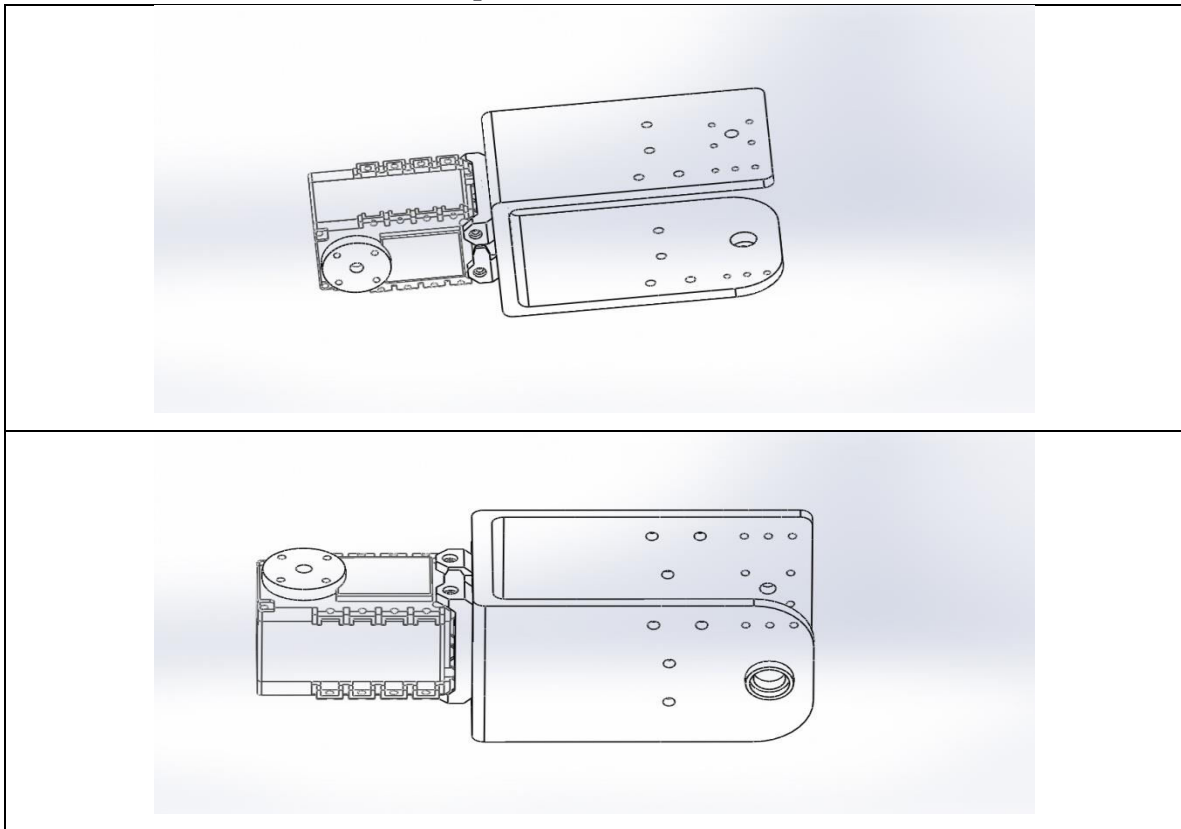
I. Dynamixel With Simple Connector.

Table 4-6 : Different views of CAD model of the Sub-Assembly 'Dynamixel with Simple Connector'



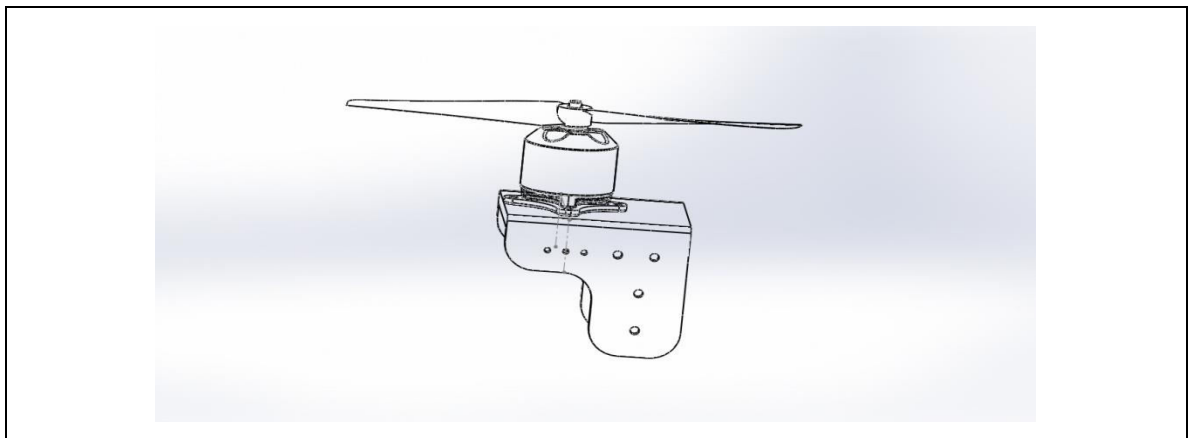
II. Dynamixel with Special Connector.

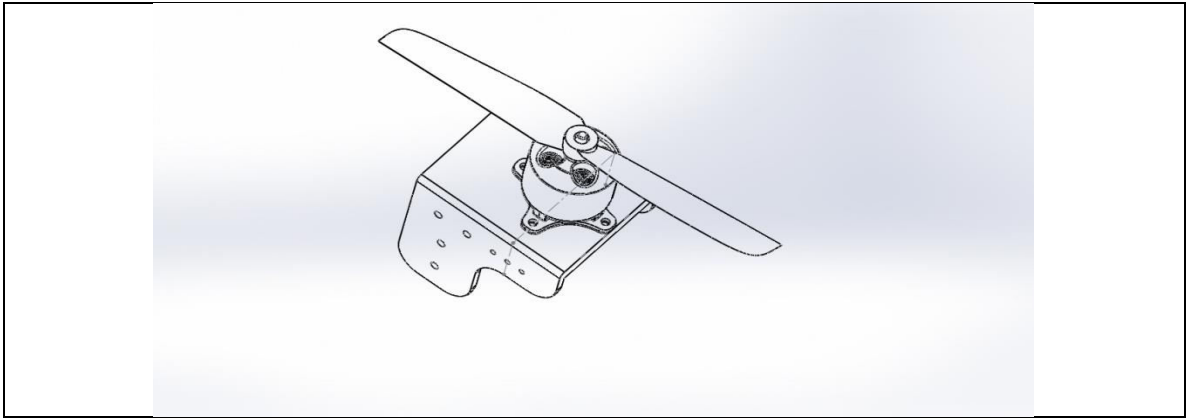
Table 4-7 : Different views of CAD model of the Sub-Assembly 'Dynamixel with Special Connector'



III. BLDC(With Propeller) and BLDC Connector

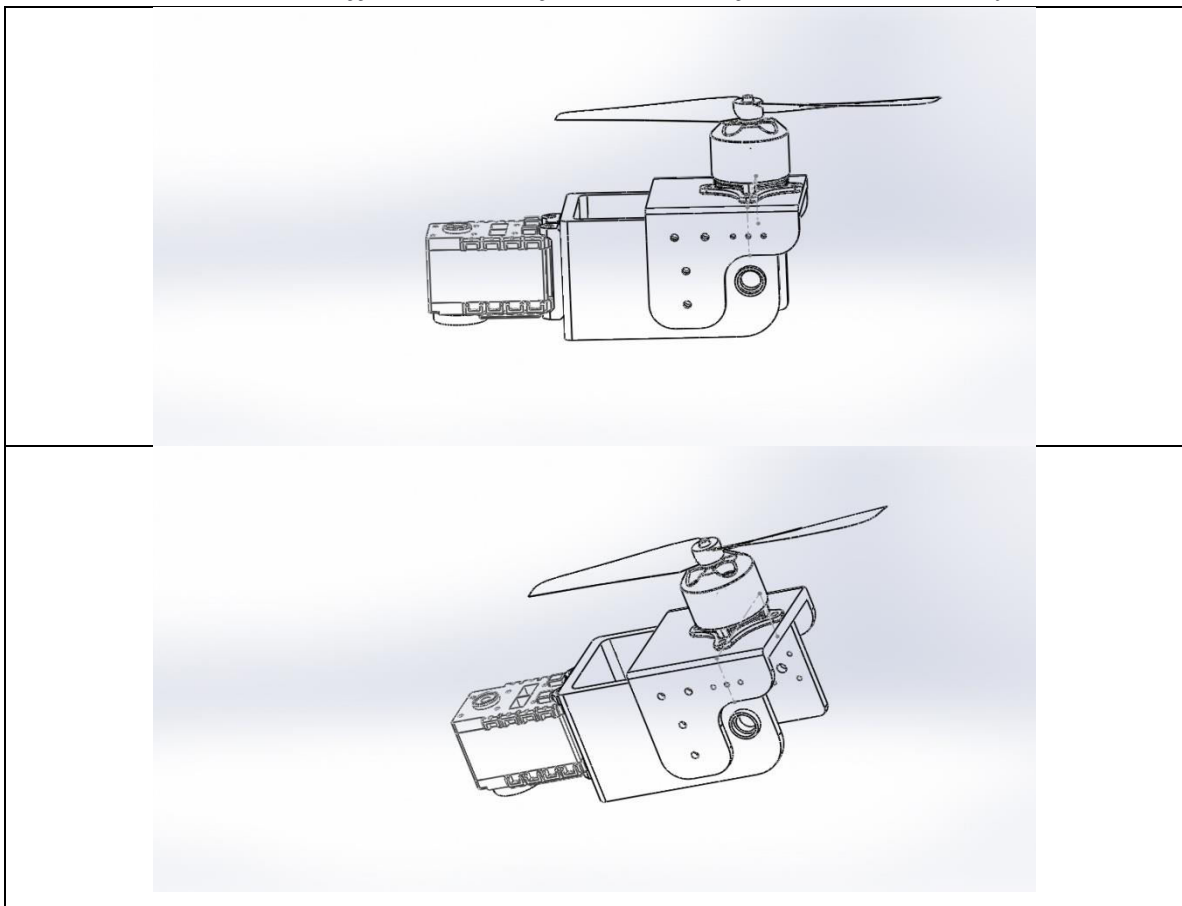
Table 4-8 : Different views of CAD model of the Sub-Assembly 'BLDC(With Propeller) and BLDC Connector'





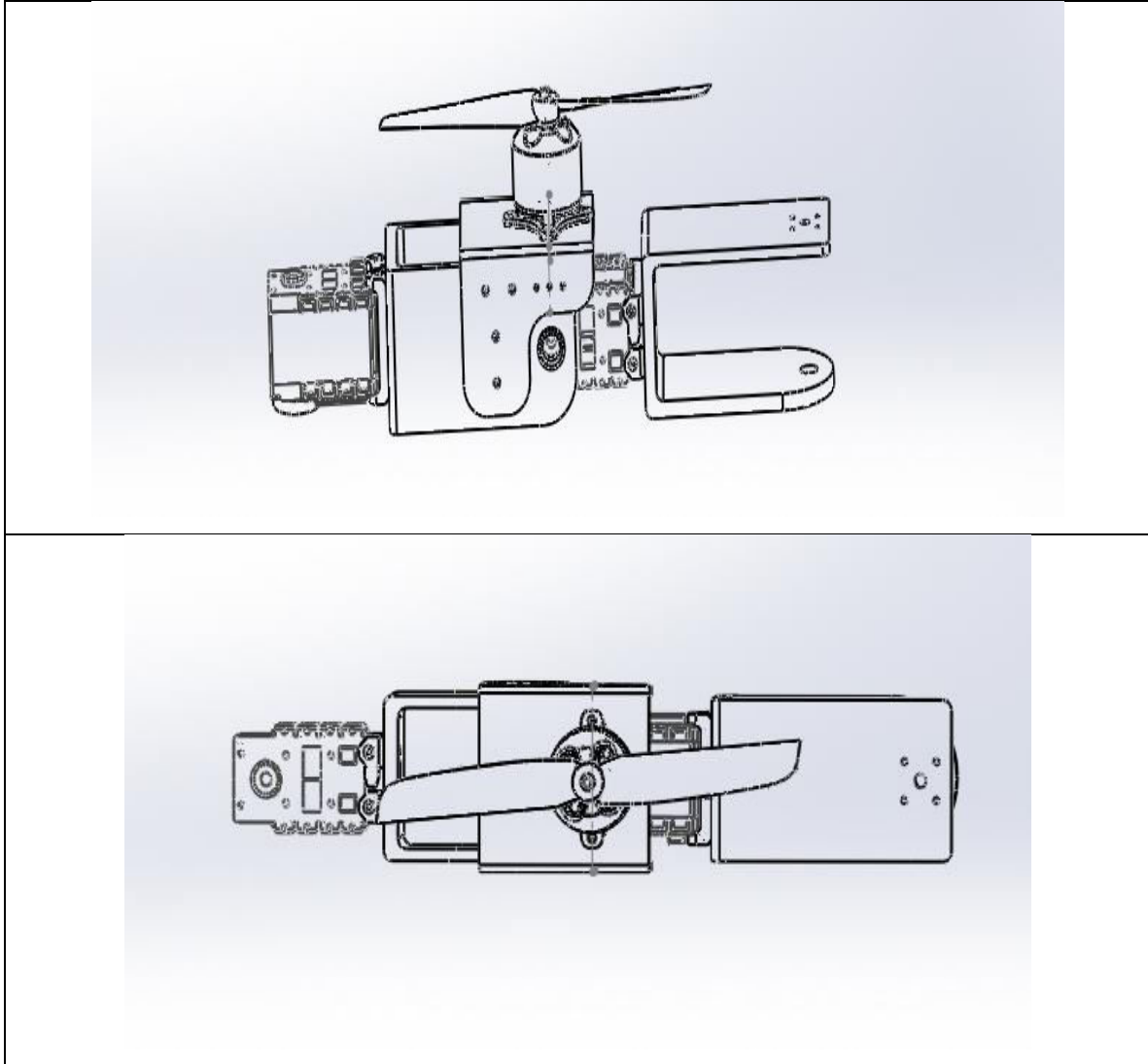
IV. SUB ASSEMBLY of Sub Assembly No. III [BLDC+BLDC Connector] with Sub Assembly No. II [Dynamixel with Special Connector] :

Table-4.9 : Different views of CAD model of the ' Sub-Assembly IV'



V. SUB ASSEMBLY of Sub Assembly No. I [Dynamixel with Simple Connector] and Sub Assembly No. IV : This Forms the basic **MODULE** of the Flying Snake Robot, which gets repeated over and over to form the whole robot.

Table 4-10 : Different views of CAD model of the Sub-Assembly V, 'The Module of the Flying Snake Robot'



In other word, for the final assembly of the robot, above Modules are placed side by side and joined together to form the whole robot. Thus, the robot has been called a Modular Robot.

4.3.3 FINAL ASSEMBLY:

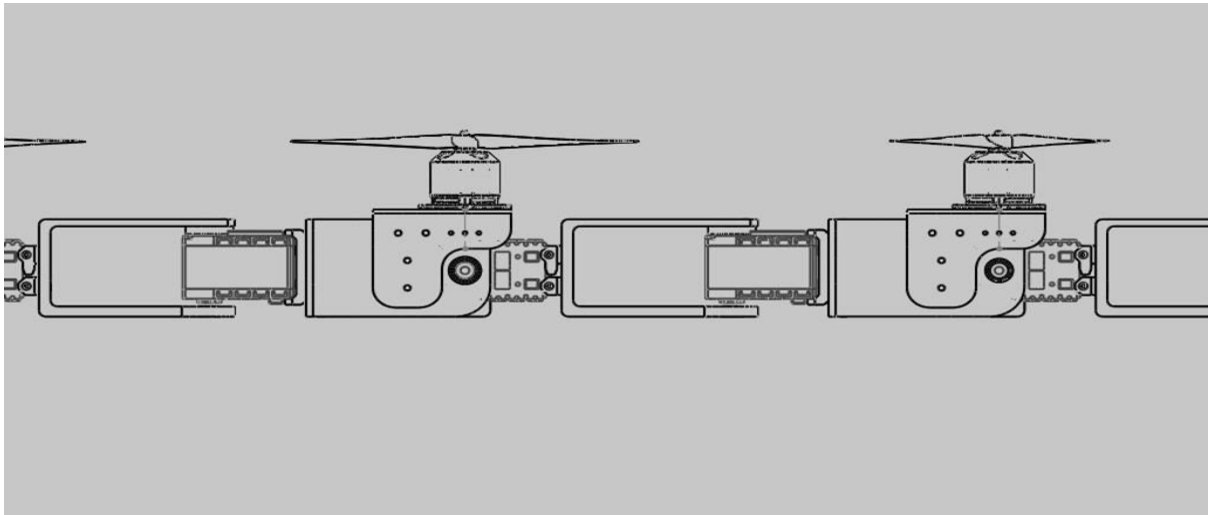


Figure 4-15 : Various Modules connected together to form Flying Snake Robot

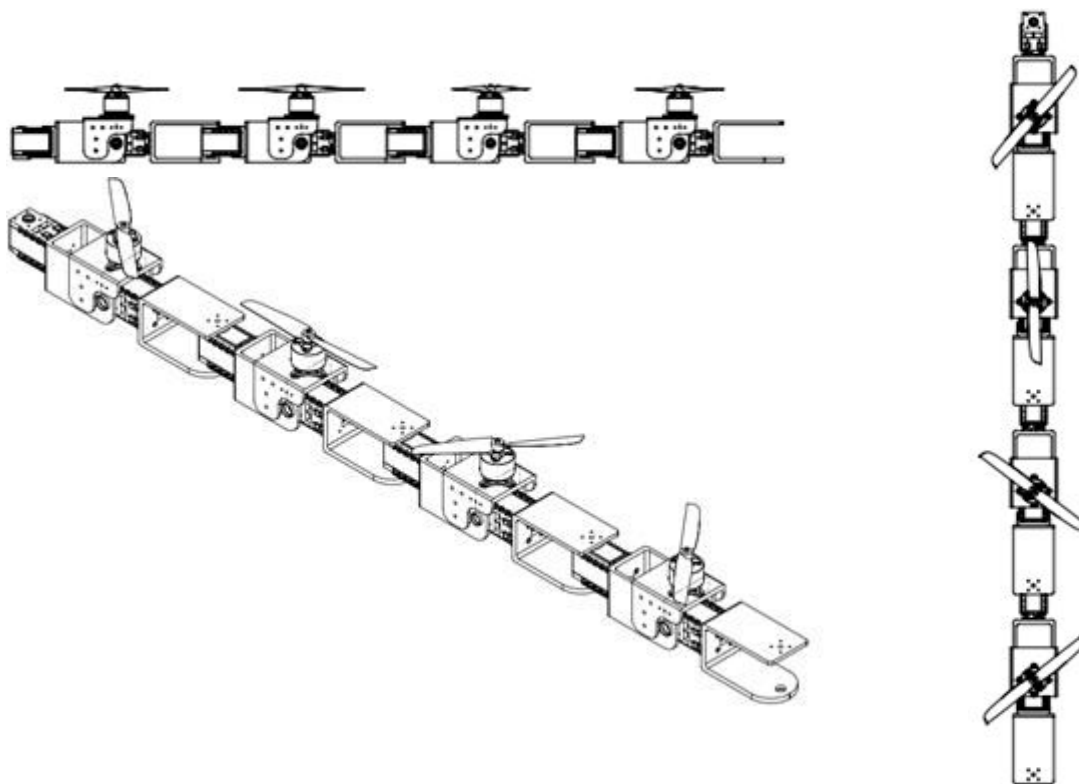


Figure 4-16 : Various Views of Flying Snake Robot in Open/Snake configuration

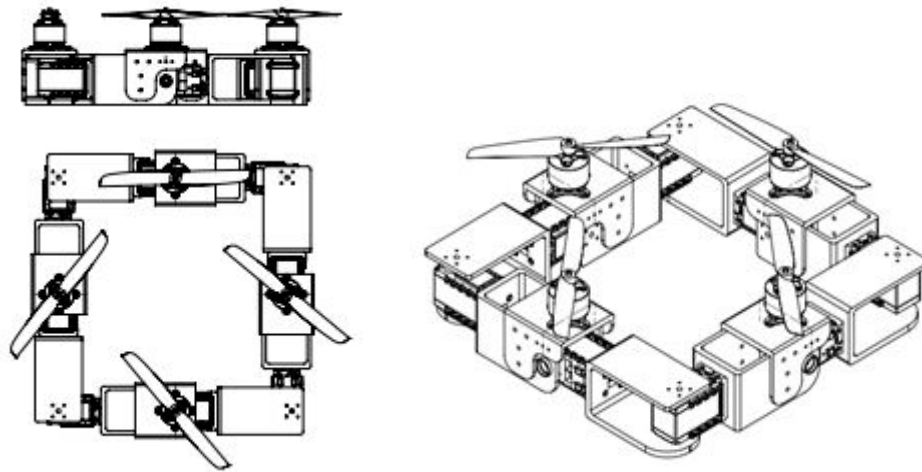


Figure 4-17 : Various Views of Flying Snake Robot in Closed/Quadcopter configuration

4.3.4 Fabricated Flying Snake Robot:

3D-printing was used to fabricate various parts involved and thus the Flying Snake Robot was assembled. The links are mounted with 8 AX-12A dynamixel actuators controlled via OpenCM processor. The BLDC motors are mounted on actuators having horizontal axis of rotation.

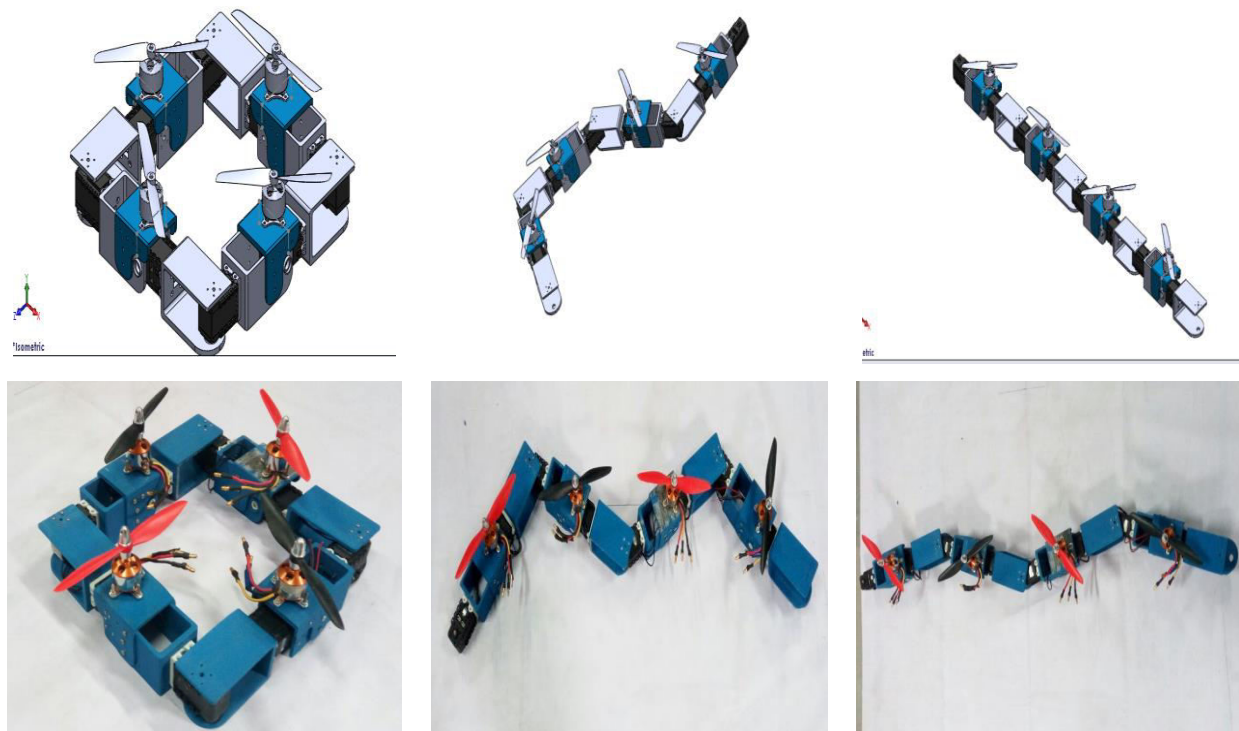


Figure 4-18: Comparison of various configurations of 3D Modelled Flying Snake Robot to Fabricated Real Robot

4.4 Design Aspects of the Flying Snake Robot:

Various Design considerations kept in mind while modelling various parts of the robot were:

- The Length of links were decided based on the quadcopter configuration of the robot. The BLDC motors have to be exactly opposite to each other as well as it had to be made sure that the propellers would not collide with each other in any case.
- It was also to be seen that propellers would not collide with any other link/part of the robot while it performed various Snake motions.
- The parts were modelled in Solidworks CAD software and using Solidworks Simulation Express, the parts were simulated for Deformation and Stresses under Load, especially to find out adequate number of holes to be made on part(used for bolts in assembling).

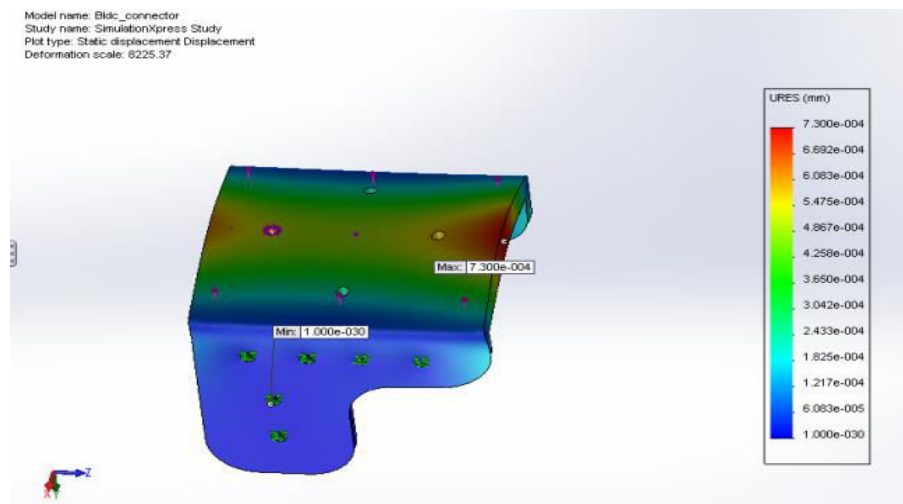


Fig 4-19 : BLDC connector under Simulation for Static Displacement under applied Load and Holes as fixtures.

- BLDC motors were placed selectively over the Dynamixels with horizontal axis of rotation such that the plane of symmetry of BLDC motor coincided with the horizontal axis of rotation. This allows no acting torque on the Dynamixels as BLDC produced upward thrust. Above can be verified by the Figure 4-20 , where Black-hashed line present axis of rotation of BLDC and Blue-hashed is axis of rotation of the Dynamixel. Thus, they shall be coinciding such that the distance between them: $R=0$; else if R is not 0, then thrust T (of BLDC) would cause $\text{Torque}=T \cdot R$ on the Horizontal Axis- motor.

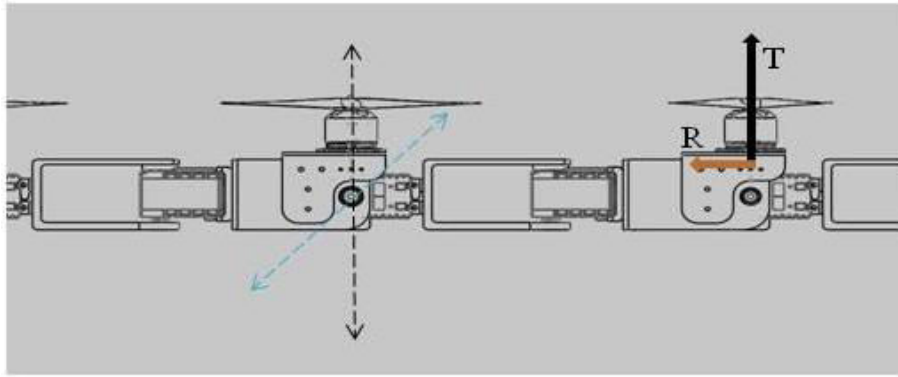


Figure 4-20 : Representation of BLDC mounted on Horizontal Axis Dynamixel.

Thus for the case of $R=0$, the only torque arising on the Dynamixels during quadcopter configuration would be due to Self-Weight of the Flying Snake Robot.

4.4.1 Major Design and Control Challenge: Asymmetry and Mass Imbalance

One of the major challenge posed is the Asymmetric nature of the Modular Snake Robot, especially when it is in the quadcopter configuration. Due to design of the modules such asymmetry is rather inevitable in case of reconfiguration into quadcopter by the Flying Snake Robot. Most of the Quadcopters' Controls Structures and Flight Controllers follow strict geometrical symmetry as well as perfect mass balance assumptions. Thus, this challenge can be said as the biggest one in the case of Flying Snake Robot.

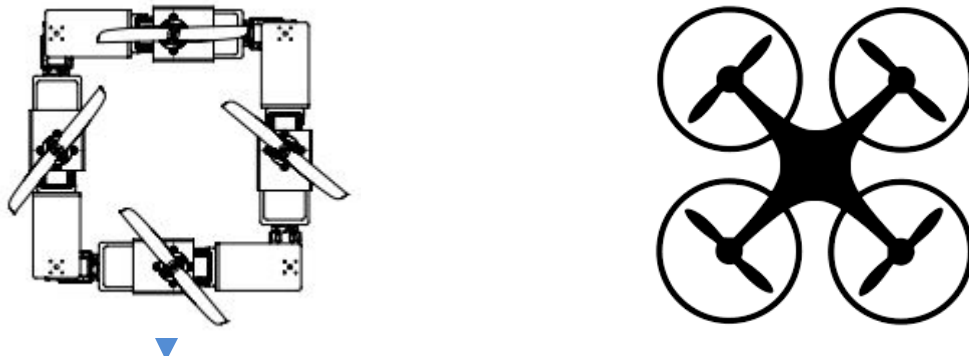


Figure 4-21 : Comparison of Symmetry between Flying Snake Robot and Standard Quadcopter

While the effect due to smaller causes of this asymmetry: The slight difference between COM position of opposite dynamixels as well as that of connector links, small components like controller boards, circuitry etc. can be neglected by applying rigorous controls, the major factor : Mass Imbalance caused by LiPo Battery(170gm, Section 4.2._) remains a problem still. This

would cause large instability with Flight Controllers or Standard Quadcopter Controls.

Solutions: Two major solutions are possible to above:

- **In the First method,** the mass imbalance is dealt in the same way as it is dealt with in industries e.g. for Vibrations, etc. i.e. by changing the mass. Since we can't decrease the mass or remove an essential component as LiPo battery, we have to add exactly equal mass opposite to it, so as to remove the Mass Imbalance due to LiPo Battery completely. This is possible via using 2 LiPo batteries connected in parallel and placed exactly opposite to each other in the quadcopter configuration or by using some other mass to balance out the single LiPo weight, though former case is much favourable and sensible. The resulting quadcopter configuration for the Flying Snake Robot can be compared to quadcopter shown in the Figure 4-22.

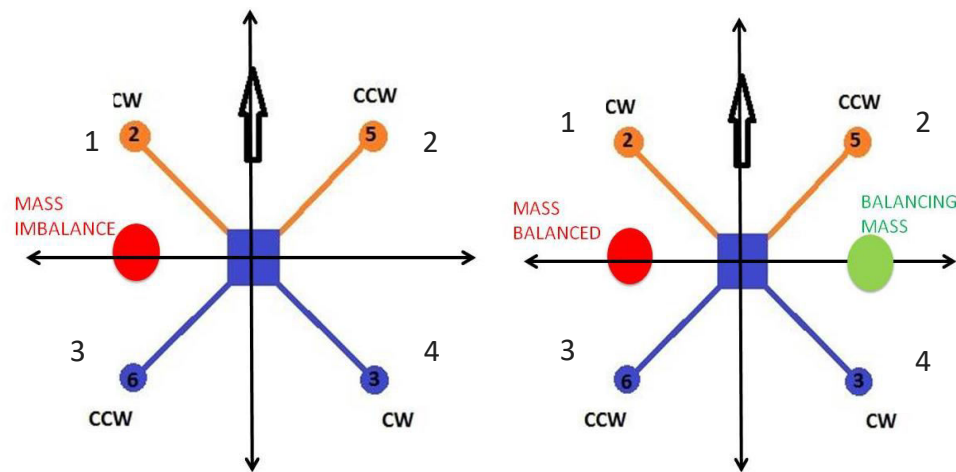


Figure 4-22 : Balancing to remove Mass Imbalance

Above method thus can be used with usual flight controllers and standard quadcopter controls(as discussed in Chapter 3, Section 4).Though it shall be kept in mind that some controllers only work perfectly when placed at COM(not in case of Naze32).The disadvantage of above method is that it causes increase in weight, thus leading to more power consumption or lesser Flight Time.

- **Second Method involves the change in Control Structure .** If the battery is placed such that a condition similar to Figure 4-22 mass imbalance arises, then it can be dealt simply by increasing the speeds of 2 motors on the side of mass imbalance by exactly the amount needed produce thrust to lift the unbalanced mass, i.e. above Motor 1 and

Motor 3 shall have higher speed than Motor 2 and Motor 4. This can be given as an offset to Motors 1 and 3 such that at minimum commanded throttle value, they lift the imbalance mass and the quadcopter stabilizes itself in horizontal plane (about Forward Direction ,hence Roll here). Thus, further the controls system remains unchanged for Yaw and Pitch (For both Motor 1 and Motor 3 act against each other to cancel out the effects of increased speed), while in Roll the unbalanced mass cancels out the net effect of increased speed of Motors 1 and 3 relative to Motors 2 and 4.

This system however can't be applied via most of market available Flight Controllers whose firmware is practically untouchable, hence this needs to be implemented on a Custom Flight Controller, which was implemented as discussed in Chapter4 Section4. Moreover this method doesn't add additional mass to system, thus no effect on flight time is seen. Hence this method proves to be better and is used further in the Thesis.

4.5 Flying Snake Robot Working:

Component Flow Diagram for the working of the Flying Snake Robot is given by Figure 4-23, while Figure 4-24 describes the Flow Chart for the working of the robot.

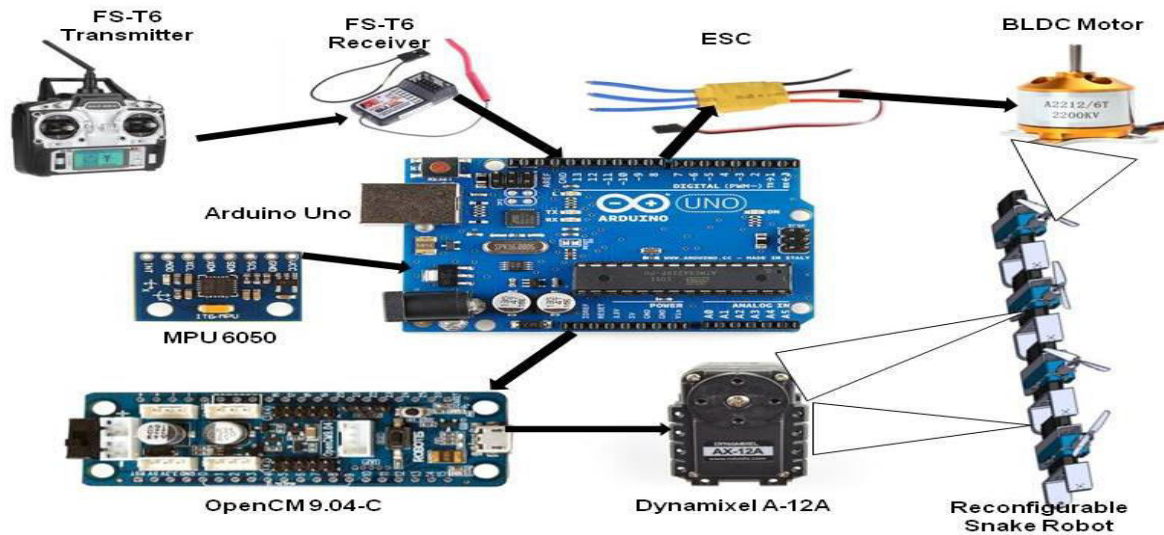


Figure 4-23: Component Flow Diagram for Flying Snake Robot

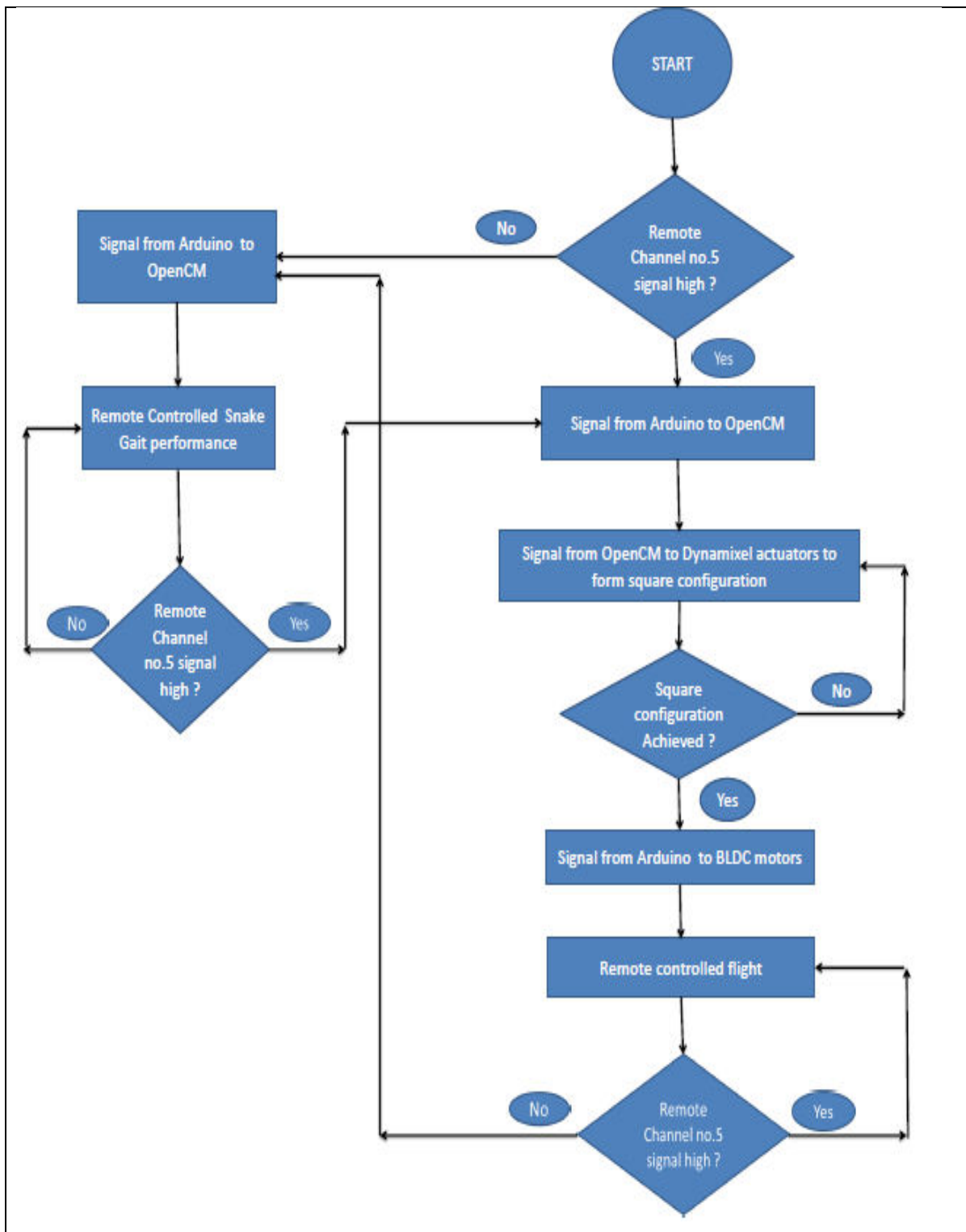


Figure 4-24 : Flow chart for Working of Flying Snake Robot

4.5.1 Working as Snake:- (Locomotion on Land)

The status of the channel 5 of the transmitter is continuously being monitored by a GPIO pin of Arduino. If it is found to be low, Arduino sends a signal to the OpenCM9.04 controller stating to perform snake locomotion. Now, three GPIO pins of Arduino are used to command OpenCM9.04 so as to perform a total of 8 snake motions on the basis of PWM signal received from different channels of the transmitter.

These different gaits of the snake being performed include: Straight line , Caterpillar forward motion, Caterpillar backward motion, Clockwise rotation , Anti-clockwise rotation, Forward Lateral shifting, Reverse Lateral shifting and Square configuration. Due to mounted propellers, this is unable to do the Rotation Motion of snake, which arises as one of the drawbacks.

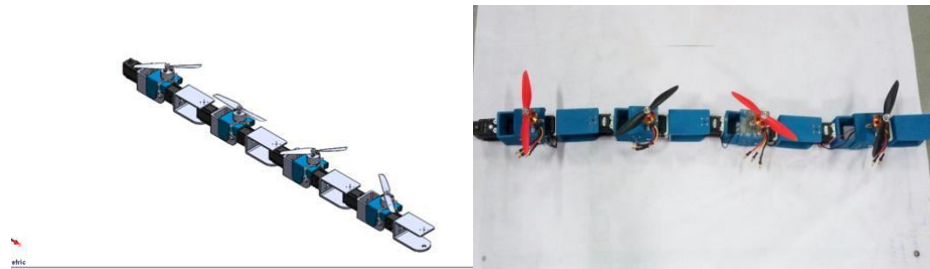


Figure 4-25 : Flying Snake Robot in Snake configuration.

Accordingly the different snake gaits are performed using the above mentioned equations. The video demonstration can be found as:- <https://www.youtube.com/watch?v=ghq8nj30-pE>

4.5.2 Working as Quadcopter: (Locomotion in air/ Flying)

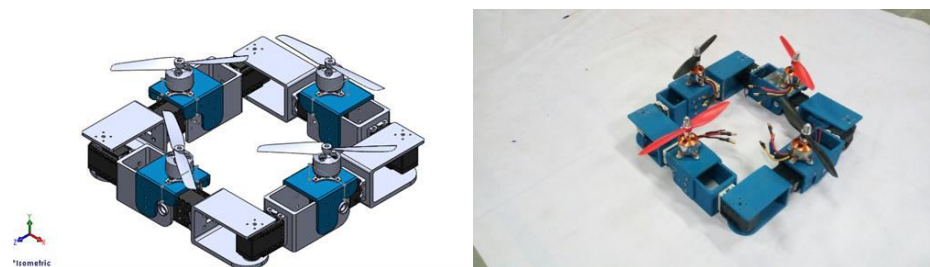


Figure 4-26 : Flying Snake Robot in Quadcopter configuration

Snake achieves an square configuration as shown in the above image. This symmetrical structure provides the ease to apply the dynamics of flight controller with brushless DC motors. The dynamixel actuators maintain their torques during the flight motion.-

During this performance, channel 5 of the transmitter is continuously monitored and if it is found to be high then, Arduino reads and commands OpenCM9.04 to form square configuration for

robust flight. Until this square formation is complete all BLDC actuators are non-active. Once the configuration is formed, signals to BLDC motors are sent through ESCs and controlled by Arduino as per the incoming commanding values on Receiver's channels and thus a stable Semi-autonomous flight is achieved.

5. RESULTS:

6.1 Flying Snake Robot was successfully designed and fabricated.

6.2 A kinematical model of Flying Snake functioning as simple snake robot was implemented.

6.3 The team was also able to demonstrate hovering of snake in square configuration.

6.4 The team was able to design Flying Snake based on the principles of modularity both in software and hardware, thus, creating a prototype that can be used as a platform for further research.

6. FUTURE WORK:

The current model is designed primarily for basic flight and easy manoeuvres in quadcopter configuration. There is much scope of research in this project. Following thing can be implemented so as to plan efficient flight control and various applications.

1. Camera can be mounted for surveillance. This type of both air and land surveillance can be implemented in –
 - Military secret operations, for spying ,survey of enemy territory etc.
 - Traffic monitoring in crowded places.
 - Remote monitoring. Surveillance of places where human access is difficult or energy consuming. Ex. Monitoring of inter-city or inter-state power transmission cables.
 - Aerial support to remote areas and victims of natural calamities.
2. Use of GPS can be done to take feedback from robot and to determine location. The robot can be controlled with wireless communication and satellite communication with the help of GPS as its location changes by giving input in terms of latitude and longitude.
3. It can be fitted with miniature cameras and sensors and can be used in geological surveys. SONAR devices can be used to analyse terrains.
4. It can be fitted with proximity sensors, camera and can be made autonomous. In this it will sense its position and alter its trajectory accordingly.
5. The major challenge would to get reconfiguration in Air, which would require extensive research in Contorls.

7. PROJECT CODES:

7.1 Codes for Arduino :

7.1.1 Interfacing MPU 6050 with Arduino

To get Quaternion, Euler, Roll-Pitch-Yaw, World_Acc, Real_Acc, etc readable values, processed via integrated DMP, and that gets simulated by Teapot application :

```
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
// #include "MPU6050.h" // not necessary if using MotionApps include file

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;

/* NOTE1: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
depends on the MPU-6050's INT pin being connected to the Arduino's
external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
digital I/O pin 2.

// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
#define OUTPUT_READABLE_QUATERNION
```

```

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal\_lock)
#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal\_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

```



```

// MPU control/status vars

bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, != 0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer


// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements
VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements
VectorFloat gravity; // [x, y, z] gravity vector
float euler[3]; // [psi, theta, phi] Euler angle container
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };


//INTERRUPT DETECTION ROUTINE
volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

//INITIAL SETUP
void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();
    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)

```

```

Serial.begin(115200);
while (!Serial); // wait for Leonardo enumeration, others continue immediately
// initialize device
Serial.println(F("Initializing I2C devices..."));
mpu.initialize();
// verify connection
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));
// wait for ready
Serial.println(F("\nSend any character to begin DMP programming and demo: "));
while (Serial.available() && Serial.read()); // empty buffer
while (!Serial.available());           // wait for data
while (Serial.available() && Serial.read()); // empty buffer again
// load and configure the DMP
Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();
// make sure it worked (returns 0 if so)
if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);
    // enable Arduino interrupt detection
    Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();
    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;
    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();

```

```

    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(""));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // you can frequently test in between other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly

```

```

mpu.resetFIFO();
Serial.println(F("FIFO overflow!"));
// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & 0x01) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;
#ifdef OUTPUT_READABLE_QUATERNION
    // display quaternion values in easy matrix form: w x y z
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    Serial.print("quat\t"); Serial.print(q.w); Serial.print("\t"); Serial.print(q.x);
    Serial.print("\t"); Serial.print(q.y); Serial.print("\t"); Serial.println(q.z);
#endif
#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q); Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI); Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI); Serial.print("\t"); Serial.println(euler[2] * 180/M_PI);
#endif
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer); mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t"); Serial.print(ypr[0] * 180/M_PI); Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI); Serial.print("\t"); Serial.println(ypr[2] * 180/M_PI);

```

```

#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);  mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q); mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t"); Serial.print(aaReal.x);  Serial.print("\t");
    Serial.print(aaReal.y); Serial.print("\t"); Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);  mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t"); Serial.print(aaWorld.x); Serial.print("\t");
    Serial.print(aaWorld.y); Serial.print("\t"); Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0]; teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4]; teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8]; teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12]; teapotPacket[9] = fifoBuffer[13];
    Serial.write(teapotPacket, 14);
    teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
#endif

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}

```

```
}
```

7.1.2 Code to run BLDC motors via ESCs through Arduino.

This also enable us to check the Minimum and Maximum PWM values/Servo angles for ESCs:

```
#include <Servo.h>

Servo FL_ESC, FR_ESC, BL_ESC, BR_ESC;

//Create as much as Servo objects=ESCs as wanted. Can control 2 or more ESC at the same time.
void setup() {
  //below code is essential to Setup the ESCs and Motors to run.
  FL_ESC.attach(3);  FL_ESC.write(30);
  BL_ESC.attach(5);  BL_ESC.write(30);
  FR_ESC.attach(6);  FR_ESC.write(30);
  BR_ESC.attach(9);  BR_ESC.write(30);
  delay(5000);
}

void loop() {
  FL_ESC.write(52); BL_ESC.write(52); FR_ESC.write(52); BR_ESC.write(52);
  // Various Speeds acan be set as servo angles 0 to 180, ESC would start run ning Motors after 40
  //and up to 160 (in general)
  //delay(500);
  // FL_ESC.write(160);  //delay(500);
  // BL_ESC.write(60);   // delay(500);
  // FR_ESC.write(100);  // delay(500);
  // BR_ESC.write(150);  // delay(500);
}
```

7.1.3 Code to receive values from different channels(5) used of Receiver of FS-T6

Transmitter Used:

```
#include "Wire.h"
```

```

#include <Servo.h>

byte PWM_PIN = 4; byte PWM_PIN2 = 7; byte PWM_PIN3 = 8; byte PWM_PIN4 = 12;
byte PWM_PIN5 = 13; //pins defined for incoming PWM values from Receiver
int pwm_value; int pwm_value2; int pwm_value3; int pwm_value4; int pwm_value5;
long rcthr, rcyaw, rcpit, rcroll;

long map(long x, long in_min, long in_max, long out_min, long out_max)
{ return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

//It is used as: result = map(VALUE, FROM_MIN, FROM_MAX, TO_MIN, TO_MAX)

void setup() {
    // put the setup code here, to run once:
    Serial.begin(9600);

    pinMode(PWM_PIN1, INPUT) ; pinMode(PWM_PIN2, INPUT) ;
    pinMode(PWM_PIN3, INPUT) ; pinMode(PWM_PIN4, INPUT) ;
    pinMode(PWM_PIN5, INPUT);
}

void loop() {
    pwm_value = pulseIn(PWM_PIN1, HIGH); // Function to read value of 1st channel
    Serial.println("\t 01 = "); Serial.println(pwm_value);
    pwm_value2 = pulseIn(PWM_PIN2, HIGH); // Function to read value of 2nd channel
    Serial.println("\t 02 = "); Serial.println(pwm_value2);
    pwm_value3 = pulseIn(PWM_PIN3, HIGH); // Function to read value of 3rd channel
    Serial.println("\t 03 = "); Serial.println(pwm_value3);
    pwm_value4 = pulseIn(PWM_PIN4, HIGH); // Function to read value of 4th channel
    Serial.println("\t 04 = "); Serial.println(pwm_value4);
    pwm_value5 = pulseIn(PWM_PIN5, HIGH); // Function to read value of 5th channel
    Serial.println("\t 05 = "); Serial.println(pwm_value5);

    //below is mapping done to convert values into usable angles etc.
    rcthr = map(pwm_value3, 970, 1954, 0, 1000); Serial.println("\t rcthr = ");
    rcyaw = map(pwm_value4, 920, 1980, -150, +150); Serial.println("\t rcyaw = ");
    rcpit = map(pwm_value, 980, 2066, 45, -45); Serial.println("\t rcpit = ");
}

```

```

rcroll = map(pwm_value2, 984, 2038, 45, -45); Serial.println("\t rcroll = ");
rcaux = map(pwm_value5, 980, 2066, 1000, 2000); Serial.println("\t rcpit = ");
delay(333); //needed to get printable output on Serial Monitor, else can remove this.
}

```

7.1.4 Final Arduino Code to control the "Flying Snake Bot":

It integrates the MPU, reads the Reciever values and accordingly directs the OpenCM board to either perform the Snake gaits or get the Bot in "Square-Config" and then controls BLDCs via ESCs for Quadcopter Motion:

```

#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
//include "MPU6050.h" // not necessary if using MotionApps include file
#include <Servo.h>
// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
// class default I2C address is 0x68 // specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high
#define OUTPUT_READABLE_YAWPITCHROLL
//input pins for I/p PWM form receiver:
byte PWM_PIN = 4; byte PWM_PIN2 = 7; byte PWM_PIN3 = 8; byte PWM_PIN4 = 12;
byte PWM_PIN5 = 13;
//Input variables to store PWM values:
int pwm_value;int pwm_value2; int pwm_value3;int pwm_value4; int pwm_value5;
long rcthr, rcyaw, rcpit, rcroll; // Variables to store rc input
Servo FL_ESC, FR_ESC, BL_ESC, BR_ESC;//servo objects for ESCs

```



```

int motorFL; int motorFR; int motorBL; int motorBR; int motorFR_act;//Angle-Values for ESCs
#define LED_PIN 1 // if really required,
bool blinkState = false;
// MPU control/status vars (See Previous Code for MPU for details):
bool dmpReady = false; uint8_t mpuIntStatus; uint8_t devStatus;          uint16_t packetSize;
uint16_t fifoCount; uint8_t fifoBuffer[64];
// orientation/motion vars
Quaternion q; VectorInt16 aa; VectorInt16 aaReal; VectorInt16 aaWorld; VectorFloat gravity;
float euler[3]; float ypr[3];
// variable needed for RPY values and Related Controls:
int prev_roll=0;int new_roll=0; int rate_mpu_roll=0; int zero_for_roll=0; int new_yaw=0;
int prev_yaw=0; int rate_mpu_yaw=0; int rate_counter=0; int zero_for_yaw=0;int new_pitch=0;
int prev_pitch=0; int rate_mpu_pitch=0; int zero_for_pitch=0;int mpu_pitch=0;int mpu_roll=0;
int mpu_yaw=0; float pitch_output=0; float roll_output=0; float yaw_output=0;
float pitch_error=0; float prev_pitch_error=0; float delta_pitch_error=0; float roll_error=0;
float prev_roll_error=0; float delta_roll_error=0; float yaw_error=0; float prev_yaw_error=0;
float delta_yaw_error=0; float net_pitch_error=0; float net_roll_error=0; float net_yaw_error=0;
int QUAD_FLAG=0;//Used to determine whether Bot is in Quad State or Snake State
//Mapping F'n, USE:- result = map(VALUE, FROM_MIN, FROM_MAX, TO_MIN, TO_MAX)
long map(long x, long in_min, long in_max, long out_min, long out_max){
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;}
// INTERRUPT DETECTION ROUTINE
volatile bool mpuInterrupt = false;  // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {    mpuInterrupt = true;    }
// INITIAL SETUP
void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE

```

```

    Fastwire::setup(400, true);
#endif

//Pins for pwm input from receiver
pinMode(PWM_PIN, INPUT);  pinMode(PWM_PIN2, INPUT);
pinMode(PWM_PIN3, INPUT);  pinMode(PWM_PIN4, INPUT);
pinMode(PWM_PIN5, INPUT);

//Pins for digital Output for commanding the OpenCM
pinMode(A0, OUTPUT); digitalWrite(A0, HIGH); //given By Default HIGH values at all
pinMode(A1, OUTPUT); digitalWrite(A1, HIGH);
pinMode(A2, OUTPUT); digitalWrite(A2, HIGH);

// initialize serial communication
    Serial.begin(9600);

    //Important Setup for Initializing ESCs
FL_ESC.attach(3); FL_ESC.write(30);  BL_ESC.attach(5);BL_ESC.write(30);
FR_ESC.attach(6); FR_ESC.write(30);  BR_ESC.attach(9); BR_ESC.write(30);delay(5000);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately
    Serial.println(F("Initializing I2C devices...")); mpu.initialize();
    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));

    //  // wait for ready :: USE BELOW FOR INITIAL TESTING ONLY!
    //  Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    //  while (Serial.available() && Serial.read()); // empty buffer
    //  while (!Serial.available());                // wait for data
    //  while (Serial.available() && Serial.read()); // empty buffer again
    // load and configure the DMP
    Serial.println(F("Initializing DMP...")); devStatus = mpu.dmpInitialize();

// supply your own gyro offsets here, scaled for min sensitivity
mpu.setXGyroOffset(220); mpu.setYGyroOffset(76); mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788); // 1688 factory default for test chip

```

```

// make sure it worked (returns 0 if so)
    if (devStatus == 0) { // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP...")); mpu.setDMPEnabled(true);
        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(0, dmpDataReady, RISING); mpuIntStatus = mpu.getIntStatus();
        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;
        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    }
else {
    // ERROR! // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
}

    pinMode(LED_PIN, OUTPUT); // configure LED for output
}

//MAIN PROGRAM LOOP
void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here(if any)    }
    // reset interrupt flag and get INT_STATUS byte

```

```

mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();
// get current FIFO count
fifoCount = mpu.getFIFOCount();
// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));
// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();
    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);
    // track FIFO count here in case there is > 1 packet available
    // (this lets us immediately read more without waiting for an interrupt)
    fifoCount -= packetSize;
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    mpu.dmpGetQuaternion(&q, fifoBuffer); // display Euler angles in degrees
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("yprimu\t");
//below yaw is very unstable at start and gets stable at some value after some time, so what we
//do is get the rate_yaw first. Then once rate gets stable, then set the coming yaw value as the
//"zero for yaw", and hence compute yaw _error and yaw_output for controller..
    // Do same for roll and Pitch too!
    new_yaw=(ypr[0] * 180/M_PI); //current yaw value
    rate_mpu_yaw=new_yaw-prev_yaw; prev_yaw=new_yaw; Serial.print("yprimu\t");
    Serial.print(rate_mpu_yaw); Serial.print("\t"); Serial.print(new_yaw); Serial.print("\t");
    new_pitch=ypr[1] * 180/M_PI-3; //offset seen in practical

```

```

rate_mpu_pitch=new_pitch-prev_pitch; prev_pitch=new_pitch;
Serial.print(mpu_pitch);    Serial.print("\t");
new_roll=ypr[2] * 180/M_PI+2; //offset seen in practical
rate_mpu_roll=new_roll-prev_roll; prev_roll=new_roll; Serial.println(mpu_roll);
if(rate_mpu_yaw==0 && rate_counter<150)
    {rate_counter=rate_counter+1; }
Serial.print("rate_counter "); Serial.print(rate_counter); Serial.print("\t");
if(rate_counter==140)
    {zero_for_yaw=new_yaw; zero_for_roll=new_roll;    zero_for_pitch=new_pitch; }
Serial.print("zero_for_yaw "); Serial.print(zero_for_yaw); Serial.print("\t");
Serial.print("zero_for_roll "); Serial.print(zero_for_roll); Serial.print("\t");
Serial.print("zero_for_pitch "); Serial.print(zero_for_pitch); Serial.print("\t");
mpu_yaw=new_yaw-zero_for_yaw;
Serial.print(" mpu_yaw_org "); Serial.print(mpu_yaw);
mpu_roll=new_roll-zero_for_roll;
Serial.print(" mpu_roll_org "); Serial.print(mpu_roll);
mpu_pitch=new_pitch-zero_for_pitch;
Serial.print(" mpu_pitch_org "); Serial.println(mpu_pitch);
//below is to map mpu_yaw value back to {-180,180} range
// mpu_yaw = map(mpu_yaw,(-180-zero_for_yaw),(+180-zero_for_yaw),-180,180);
//Serial.print("mpu_yaw_mapped ");Serial.println(mpu_yaw);

```

```

//below add up for Receiving PWM values:
pwm_value = pulseIn(PWM_PIN, HIGH);
//Serial.println("\t 01 = "); //Serial.println(pwm_value);
pwm_value2 = pulseIn(PWM_PIN2, HIGH);
//Serial.println("\t 02 = "); //Serial.println(pwm_value2);
pwm_value3 = pulseIn(PWM_PIN3, HIGH);
//Serial.println("\t 03 = "); //Serial.println(pwm_value3);
pwm_value4 = pulseIn(PWM_PIN4, HIGH);
//Serial.println("\t 04 = "); //Serial.println(pwm_value4);

```

```

    pwm_value5 = pulseIn(PWM_PIN5, HIGH); // value btw 984 and 1965
//Map above into usable values:
    rcthr = map(pwm_value3,970,1954,0,1000);
    rcyaw = map(pwm_value4, 920, 1980, -180, +180);
    // defined offsets as per receiver values for hovering:
    if(rcyaw < 22 && rcyaw > 0){ rcyaw=0;}
    rcpit = map(pwm_value, 980, 2066, 45, -45);
    if(rcpit < 8 && rcpit > 1){ rcpit=0;}
    rcroll = map(pwm_value2, 984, 2038, 45, -45);
    if(rcroll < 6 && rcroll > 1){ rcroll=0;} // See all above values on serial Monitor to verify
    Serial.print(rcthr); Serial.print("\t"); Serial.print(rcyaw); Serial.print("\t"); Serial.print(rcpit);
    Serial.print("\t"); Serial.println(rcroll); Serial.print("pwm5\t"); Serial.println(pwm_value5);
//PID CONTROLS:
    pitch_error=rcpit-mpu_pitch; delta_pitch_error=pitch_error-prev_pitch_error;
    if(rate_counter>142 && rcthr > 200)
    {net_pitch_error=net_pitch_error+pitch_error;}
    else
    { net_pitch_error=0;}
    //pitch_output=0.4*pitch_error+0.4*delta_pitch_error;//set kp and kd here
    pitch_output=0.25*pitch_error+0.0*delta_pitch_error+0.025*net_pitch_error;
    prev_pitch_error=pitch_error; //set kp ,kd,ki above
    roll_error=rcroll-mpu_roll;
    delta_roll_error=roll_error-prev_roll_error;
    if(rate_counter>142 && rcthr > 200)
    {net_roll_error=net_roll_error+roll_error;//introducing term for I n Ki}
    else
    { net_roll_error=0;}
    //roll_output=0.05*roll_error+0.05*delta_roll_error;//set kp and kd here
    roll_output=0.05*roll_error+0.00*delta_roll_error+0.0*net_roll_error;//set kp ,kd,ki here
    prev_roll_error=roll_error;
    yaw_error=rcyaw-mpu_yaw;

```

```

    delta_yaw_error=yaw_error-prev_yaw_error;
    if(rate_counter>142 && rcthr > 200)
        { net_yaw_error=net_yaw_error+yaw_error;}
    else{ net_yaw_error=0;}
    //yaw_output=0.12*yaw_error+0.17*delta_yaw_error;//set kp and kd here
    yaw_output=0.05*yaw_error+0.0*delta_yaw_error+0.0*net_yaw_error;
    prev_yaw_error=yaw_error; // set Kp Kd Ki above
    Serial.print(net_yaw_error); Serial.print("\t"); Serial.print(net_pitch_error);
    Serial.print("\t");Serial.print(net_roll_error); Serial.print("\t"); Serial.print(yaw_output);
    Serial.print("\t");Serial.print(pitch_output); Serial.print("\t"); Serial.println(roll_output);

if(rate_counter>142){ //Will do ANYTHING only after stable MPU read
    if(pwm_value5>1500 && QUAD_FLAG==0 && rcthr < 100){
        digitalWrite(A0, LOW); Serial.print("A0=low ");
        digitalWrite(A1, LOW);Serial.print("A1=low ");
        digitalWrite(A2, LOW);Serial.print("A2=low ");
        QUAD_FLAG=1;
        delay(5000); //give 5 seconds for quad config
    }
    if(pwm_value5>1500 && QUAD_FLAG==1 && rcthr > 200) {
        Serial.print("MOTOR_FL\t");
        //Serial.print((rcthr + roll_output + pitch_output - yaw_output));
        motorFL = map((rcthr + roll_output + pitch_output - yaw_output), 185, 1080, 51, 180);
        FL_ESC.write(motorFL);    Serial.print(motorFL);  Serial.print("\t");
        Serial.print("MOTOR_BL,\t");
        //Serial.print(rcthr + roll_output - pitch_output + yaw_output);
        motorBL = map((rcthr + roll_output - pitch_output + yaw_output), 185, 1080, 51, 180);
        BL_ESC.write(motorBL); Serial.print(motorBL); Serial.print("\t");
        Serial.print("MOTOR_FR,\t");
        //Serial.print(rcthr - roll_output + pitch_output + yaw_output);
        motorFR = map((rcthr - roll_output + pitch_output + yaw_output), 185, 1080, 51, 180);
    }
}

```

```

//motorFR_act = map(motorFR, 51, 180, 39, 142); if ESC start and end values r different
FR_ESC.write(motorFR);    Serial.print(motorFR);    Serial.print("\t");
//FR_ESC.write(motorFR_act); //Serial.print("MOTOR_FR_act,\t");
//Serial.print(motorFR); Serial.print("\t"); Serial.print("MOTOR_BR,\t");
//Serial.print(rcthr - roll_output - pitch_output - yaw_output);
motorBR = map((rcthr - roll_output - pitch_output - yaw_output), 185, 1080, 51, 180);
BR_ESC.write(motorBR);    Serial.print(motorBR);        Serial.println("\t");
QUAD_FLAG=1;
}

if(pwm_value5>1500 && QUAD_FLAG==1 && rcthr < 200) {
    Serial.print("MOTOR_FL\t"); Serial.print(40); FL_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_BL,\t"); Serial.print(40); BL_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_FR,\t"); Serial.print(40); FR_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_BR,\t");    Serial.println(40);    BR_ESC.write(40);
    QUAD_FLAG=1;
}

if(pwm_value5<1200 && QUAD_FLAG==1 && rcthr < 100){
    Serial.print("MOTOR_FL\t"); Serial.print(40); FL_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_BL,\t"); Serial.print(40); BL_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_FR,\t"); Serial.print(40);FR_ESC.write(40); Serial.print("\t");
    Serial.print("MOTOR_BR,\t"); Serial.println(40); BR_ESC.write(40);
    digitalWrite(A0, HIGH); Serial.print("A0=high ");
    digitalWrite(A1, HIGH); Serial.println("A1=high ");
    digitalWrite(A2, HIGH); Serial.println("A2=high ");
    QUAD_FLAG=0;
    delay(5000); //give 5 seconds for SNAKE RE-config
}

if(pwm_value5<1200 && QUAD_FLAG==0) {
    digitalWrite(A0, HIGH); digitalWrite(A1, HIGH); digitalWrite(A2, HIGH);
    QUAD_FLAG=0;
    Serial.print("MOTOR_FL\t"); Serial.print(40); FL_ESC.write(40); Serial.print("\t");

```



```

Serial.print("MOTOR_BL,\t"); Serial.print(40); BL_ESC.write(40); Serial.print("\t");
Serial.print("MOTOR_FR,\t"); Serial.print(40); FR_ESC.write(40); Serial.print("\t");
Serial.print("MOTOR_BR,\t"); Serial.println(40); BR_ESC.write(40);

    if(rcthr<200) {
        digitalWrite(A0, HIGH); digitalWrite(A1, HIGH); digitalWrite(A2, HIGH);
        Serial.print("A0=high ");Serial.print("A1=high ");Serial.println("A2=high ");
    }
    else if(rcthr>500 && rcthr<750) {
        if(rcroll>20){
            digitalWrite(A0, LOW);      digitalWrite(A1, HIGH);
            digitalWrite(A2, LOW);      Serial.print("A0=low ");
            Serial.print("A1=high ");    Serial.println("A2=low ");
        }
        if(rcroll<-20) {
            digitalWrite(A0, HIGH);      digitalWrite(A1, LOW);
            digitalWrite(A2, LOW);      Serial.print("A0=high ");
            Serial.print("A1=low ");    Serial.println("A2=low ");
        }
    }
    else if(rcthr>750){
        if(rcroll>20) {
            digitalWrite(A0, LOW);      digitalWrite(A1, HIGH);
            digitalWrite(A2, HIGH);      Serial.print("A0=low ");
            Serial.print("A1=high ");    Serial.println("A2=high ");
        }
        if(rcroll<-20){
            digitalWrite(A0, HIGH);      digitalWrite(A1, LOW);
            digitalWrite(A2, HIGH);      Serial.print("A0=high ");
            Serial.print("A1=low ");    Serial.println("A2=high ");
        }
    }
}

```

```

else if(rcthr>200 && rcthr<500) {
    if(rcroll>20) {
        digitalWrite(A0, HIGH);    digitalWrite(A1, HIGH);
        digitalWrite(A2, LOW);      Serial.print("A0=high ");
        Serial.print("A1=high ");   Serial.println("A2=low ");
    }
    if(rcroll<-20){
        digitalWrite(A0, LOW);     digitalWrite(A1, LOW);
        digitalWrite(A2, HIGH);     Serial.print("A0=low ");
        Serial.print("A1=low ");    Serial.println("A2=high ");
    }
    }
}

else{ // motors off and do nothing
    Serial.print("MOTOR_FL\t"); Serial.print(40); FL_ESC.write(40);
    Serial.print("\t"); Serial.print("MOTOR_BL,\t"); Serial.print(40); BL_ESC.write(40);
    Serial.print("\t"); Serial.print("MOTOR_FR,\t"); Serial.print(40); FR_ESC.write(40);
    Serial.print("\t"); Serial.print("MOTOR_BR,\t"); Serial.println(40); BR_ESC.write(40);
}

//delay(300);//use for serial printing only
#endif

// blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}
}

```

7.2 Code for OpenCM9.04

Code for controlling various gaits of snake robot after receiving signal from Arduino and Receiver :

```
/* OpenCM 485 EXP board is needed to use 4 pin Dynamixel and Pro Series */
/* Serial device defines for dxl bus */
#define DXL_BUS_SERIAL1 1 //Dynamixel on Serial1(USART1) <-OpenCM9.04
#define DXL_BUS_SERIAL2 2 //Dynamixel on Serial2(USART2) <-LN101,BT210
#define DXL_BUS_SERIAL3 3 //Dynamixel on Serial3(USART3) <-OpenCM
485EXP
/* Dynamixel ID defines */
#define ID_NUM1 1 #define ID_NUM2 2 #define ID_NUM3 3 #define ID_NUM4 4
#define ID_NUM5 5 #define ID_NUM6 6 #define ID_NUM7 7 #define ID_NUM8 8
#define speed 180
volatile float volt10=0; volatile float volt11=0; volatile float volt12=0;
Dynamixel Dxl(DXL_BUS_SERIAL1);
void setup() {
    /* Dynamixel 2.0 Baudrate -> 0: 9600, 1: 57600, 2: 115200, 3: 1Mbps */
    Dxl.begin(3);
    /*Defining the digital pins as Input, to be read from Arduino */
    pinMode(10, INPUT); pinMode(11, INPUT);pinMode(12, INPUT);

    Dxl.goalSpeed(ID_NUM1, speed); Dxl.jointMode(ID_NUM1);
    Dxl.goalSpeed(ID_NUM2, speed); Dxl.jointMode(ID_NUM2);
    Dxl.goalSpeed(ID_NUM3, speed); Dxl.jointMode(ID_NUM3);
    Dxl.goalSpeed(ID_NUM4, speed); Dxl.jointMode(ID_NUM4);
    Dxl.goalSpeed(ID_NUM5, speed); Dxl.jointMode(ID_NUM5);
    Dxl.goalSpeed(ID_NUM6, speed); Dxl.jointMode(ID_NUM6);
    Dxl.goalSpeed(ID_NUM7, speed); Dxl.jointMode(ID_NUM7);
    Dxl.goalSpeed(ID_NUM8, speed); Dxl.jointMode(ID_NUM8);
}

    int a=1; float t=0;

void loop() {
```

```

/* Reading digital pins and displaying on Serial Monitor */
    int Data10 = digitalRead(10); volt10 = Data10;
    int Data11 = digitalRead(11); volt11 = Data11;
    int Data12 = digitalRead(12); volt12 = Data12;
    SerialUSB.print(" volt10 = "); SerialUSB.print(volt10 );
    SerialUSB.print(" volt11 = "); SerialUSB.print(volt11 );
    SerialUSB.print(" volt12 = "); SerialUSB.print(volt12 );
    SerialUSB.println(" [V]");    delay(25);

if ( (volt12== 1) && (volt11== 1) && (volt10== 1))
{
    //----- STRAIGHT LINE FORMATION -----//
    Dxl.goalPosition(ID_NUM1,355);  Dxl.goalPosition(ID_NUM3,360);
    Dxl.goalPosition(ID_NUM5,360);  Dxl.goalPosition(ID_NUM7,360);
    Dxl.goalPosition(ID_NUM2,350);  Dxl.goalPosition(ID_NUM4,350);
    Dxl.goalPosition(ID_NUM6,350);  Dxl.goalPosition(ID_NUM8,350);
}

else if ( (volt12== 1) && (volt11== 1) && (volt10== 0) )
{
    //-----FORWARD CATERPILLAR MOTION -----//
    Dxl.setPosition(ID_NUM1,355,80);  Dxl.setPosition(ID_NUM3,360,80);
    Dxl.setPosition(ID_NUM5,360,80);  Dxl.setPosition(ID_NUM7,360,80);
    Dxl.setPosition(ID_NUM2,360+60*sin(((9*3.14*t)/6)+((2*3.14))/3),600);
    Dxl.setPosition(ID_NUM4,360+60*sin(((9*3.14*t)/6)+((4*3.14))/3),600);
    Dxl.setPosition(ID_NUM6,360+60*sin(((9*3.14*t)/6)+((6*3.14))/3),600);
    Dxl.setPosition(ID_NUM8,360+60*sin(((9*3.14*t)/6)+((8*3.14))/3),600);
}

else if ( (volt12== 1) && (volt11== 0) && (volt10== 1) )
{
    //-----BACKWARD CATERPILLAR MOTION -----//
    Dxl.setPosition(ID_NUM1,355,80);  Dxl.setPosition(ID_NUM3,360,80);
    Dxl.setPosition(ID_NUM5,360,80);  Dxl.setPosition(ID_NUM7,360,80);
    Dxl.setPosition(ID_NUM2,360+60*sin(((9*3.14*t)/6)-((2*3.14))/3),600);
    Dxl.setPosition(ID_NUM4,360+60*sin(((9*3.14*t)/6)-((4*3.14))/3),600);
}

```

```

        Dxl.setPosition(ID_NUM6,360+60*sin(((9*3.14*t)/6)-((6*3.14)/3),600);
        Dxl.setPosition(ID_NUM8,360+60*sin(((9*3.14*t)/6)-((8*3.14)/3),600);
    }
else if ( (volt12== 0) && (volt11== 0) && (volt10== 1) )
    {
        //-----LATERAL SHIFTING MOTION -----//
        Dxl.setPosition(ID_NUM1,360+30*sin(((5*3.14*t)/6)+((0*3.14)/3),50);
        Dxl.goalPosition(ID_NUM3,360+100*sin(((13*3.14*t)/6)-((4*3.14)/3));
        Dxl.goalPosition(ID_NUM5,360+100*sin(((13*3.14*t)/6)-((8*3.14)/3));
        Dxl.goalPosition(ID_NUM7,360+100*sin(((13*3.14*t)/6)-((12*3.14)/3));
        Dxl.goalPosition(ID_NUM2,360+30*sin(((13*3.14*t)/6)-((2*3.14)/3));
        Dxl.goalPosition(ID_NUM4,360+30*sin(((13*3.14*t)/6)-((6*3.14)/3));
        Dxl.goalPosition(ID_NUM6,360+30*sin(((13*3.14*t)/6)-((10*3.14)/3));
        Dxl.goalPosition(ID_NUM8,360+30*sin(((13*3.14*t)/6)-((14*3.14)/3));
    }
else if ( (volt12== 0) && (volt11== 1) && (volt10== 1) )
    {
        //-----CLOCKWISE ROTATING MOTION-----//
        Dxl.goalPosition(ID_NUM2,360+40*sin(((3*3.14*t)/6)+(2*3.14/3));
        Dxl.goalPosition(ID_NUM3,360+150*sin(((3*3.14*t)/6)+(4*3.14/3));
        Dxl.goalPosition(ID_NUM4,360+40*sin(((3*3.14*t)/6)+(6*3.14/3));
        Dxl.goalPosition(ID_NUM5,360+150*sin(((3*3.14*t)/6)+(8*3.14/3));
        Dxl.goalPosition(ID_NUM6,360+40*sin(((3*3.14*t)/6)+(10*3.14/3));
        Dxl.goalPosition(ID_NUM7,360+150*sin(((3*3.14*t)/6)+(12*3.14/3));
        Dxl.goalPosition(ID_NUM8,360+40*sin(((3*3.14*t)/6)+(14*3.14/3));
        delay(9);
    }
else if ( (volt12== 1) && (volt11== 0) && (volt10== 0) )
    {
        //-----COUNTERCLOCKWISE ROTATING MOTION-----//
        Dxl.goalPosition(ID_NUM2,360+40*sin(((3*3.14*t)/6)-(2*3.14/3));
        Dxl.goalPosition(ID_NUM3,360+150*sin(((3*3.14*t)/6)-(4*3.14/3));
        Dxl.goalPosition(ID_NUM4,360+40*sin(((3*3.14*t)/6)-(6*3.14/3));
        Dxl.goalPosition(ID_NUM5,360+150*sin(((3*3.14*t)/6)-(8*3.14/3));
    }

```

```

        Dxl.goalPosition(ID_NUM6,360+40*sin(((3*3.14*t)/6)-(10*3.14/3)));
        Dxl.goalPosition(ID_NUM7,360+150*sin(((3*3.14*t)/6)-(12*3.14/3)));
        Dxl.goalPosition(ID_NUM8,360+40*sin(((3*3.14*t)/6)-(14*3.14/3)));
        delay(9);
    }
else
    {
        //----- SNAKE SQUARE CONFIGURATION-----//
        Dxl.setPosition(ID_NUM2,360,80); Dxl.setPosition(ID_NUM4,360,80);
        Dxl.setPosition(ID_NUM6,360,80); Dxl.setPosition(ID_NUM8,360,80);
        Dxl.setPosition(ID_NUM3,45,80); Dxl.setPosition(ID_NUM5,45,80);
        Dxl.setPosition(ID_NUM7,45,80); delay(5);
    }
if(a==1) { t+=0.09; }
}

```

8. REFERENCES:

1. Mobile Robots :- https://en.wikipedia.org/wiki/Mobile_robot
2. Side winding motion – <https://www.youtube.com/watch?v=JyXNBaHu32o>
3. Lateral Undulation – <https://www.youtube.com/watch?v=ZKaYbMZqTkY>
4. Kinematical Equations – <http://www.inobotics.com/2012/09/project-snake-robot.html>
5. Snake Applications:- <http://www.inobotics.com/2012/09/applications-of-snake-robot.html>
6. Snake robot as a Spy:- <https://www.newscientist.com/article/dn4075-robot-spy-can-survive-battlefield-damage/>
7. “Bio-inspired locomotion for a modular snake robot” by Shubo Zhang and Yi Guo
8. “Generating gaits for snake robots: annealed chain fitting and keyframe wave extraction” by Ross L. Hatton · Howie Choset
9. Drone applications:- <https://dronebuff.com/uses-for-drones/>
10. BLDC-Purchase-reference:-<http://www.amazon.in/REES52-2200-MOTOR-BULLET-Connector/dp/B01MEERG6O>
11. Propellers-Purchase-reference:-<http://www.amazon.in/APC-Landing-Products-LP06040E-Electric-Propeller/dp/B0006O34O4>
12. ESC Purchase reference:- <http://www.amazon.in/Robomart-30A-BLDC-ESC/dp/B00RFEMV5G>
13. Arduino Uno R3 Purchase reference:- <https://www.robomart.com/arduino-uno-online-india>
14. IMU Purchase reference:- <http://www.amazon.in/GY-521-Mpu6050-Accelerometer-Arduino-REES52/dp/B008BOPN40>
15. IMU Specification manual:- <http://www.hotmcu.com/gy521-mpu6050-3axis-acceleration-gyroscope-6dof-module-p-83.html>

16. OpenCM9.04 Purchase reference and manual downloads:-
<https://www.mgsuperlabs.co.in/estore/OpenCM9-04-A>
17. OpenCM9.04 Software installation:-
http://support.robotis.com/en/software/robotis_opencm/robotis_opencm.htm
18. Dynamixel-Purchaselink:-<https://www.mgsuperlabs.co.in/estore/DYNAMIXEL-AX-12A>
19. Lipo Battery Purchase reference:- <http://robokits.co.in/batteries-chargers>
20. Lipo Battery Terminologies:- <https://rogershobbycenter.com/lipoguide/>
21. Flysky FS-T6 Purchase reference:- <http://www.amazon.in/FlySky-FS-T6-Proportional-Transmitter-Receiver/dp/B016D91CN0>
22. Flysky FS-T6 User manual :-
<https://hobbyking.com/media/file/692940060X652998X22.pdf>