

# **Autonomous Airtransportation using Vision Navigation**

Submitted to:

Prof. Basman Elhadidi  
Asst. Prof. Osama Saaid  
Asst. Prof. Mohanad

By:

July 1, 2019

---

# Acknowledgments

Before we start our Thesis we should thank and appreciate everyone helped us to reach to this moment and we admit that without their help and knowledge we couldn't reach here.

first of all we have to thank and appreciate our directly *Asst. Prof. Osama Saaid* who helped us alot and gave us from his wide knowledge we would like to thank him for his great help through this year, and we also should thank *Prof. Basman Elhadidi & Eng. Mohanad* for their time and their contribution in the design phase, and we also want to thank the all members in ASTL & UDC who helped us alot with their technical information.

---

# List of Symbols and Abbreviations

ROS	<i>Robot Operating System</i>
SLAM	<i>Simultaneous Localization And Mapping</i>
SITL	<i>Software In The Loop</i>
HITL	<i>Hardware In The Loop</i>
LIDAR	<i>Light Detection And Ranging</i>
KF	<i>Kalman Filter</i>
EKF	<i>Extended Kalman Filter</i>
IMU	<i>Inertial Measurement Unit</i>
EOM	<i>Equation Of Motion</i>
LTI	<i>Linear Time Invarient</i>
PID	<i>Proportional-Integral-Derivative Control</i>
GPU	<i>Graphical Processing Unit</i>
ODD	<i>Operational Design Domain</i>
MC	<i>Multicopter</i>
AAV	<i>Autonomous Aerial Vehicle</i>
VTOL	<i>Vertical Take-Off and Landing</i>
ATC	<i>Air Traffic Control</i>
AP	<i>Autopilot</i>

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	FLYING TAXI . . . . .	5
1.1.1	why air taxis are an attractive idea . . . . .	5
1.1.2	obstacles we need to pass . . . . .	6
1.1.3	Flying taxi categories . . . . .	7
1.1.4	Technology areas dominating air taxi technology . . . . .	8
1.2	the key players in the flying cars space . . . . .	8
1.3	Prototype . . . . .	10
<b>2</b>	<b>Aircraft Design</b>	<b>11</b>
2.1	Aerodynamic Design . . . . .	11
2.1.1	Introduction . . . . .	11
2.1.2	Wing configuration . . . . .	11
2.1.3	Design Criterion . . . . .	12
2.1.3.1	Airfoil selection . . . . .	12
2.1.4	Aerodynamic analysis: . . . . .	14
2.1.4.1	Investigating goals and requirements: . . . . .	15
2.1.4.2	Design parameters . . . . .	16
2.1.4.3	Results: . . . . .	16
2.1.5	Optimization for aerodynamic stability: . . . . .	17
2.1.6	Final results: . . . . .	18
2.1.7	Stability Analysis: . . . . .	20
2.1.7.1	For longitudinal: . . . . .	20
2.1.7.2	For lateral . . . . .	22
2.2	Mechanical Design . . . . .	25
2.2.1	Introduction . . . . .	25
2.2.1.1	Why Quad-plane . . . . .	25
2.2.1.2	Scaling . . . . .	25
2.2.1.3	Payload . . . . .	25
2.2.1.4	Endurance . . . . .	26
2.2.1.5	Range . . . . .	26
2.2.2	Sizing . . . . .	26
2.2.3	Propulsion system selection . . . . .	26
2.2.3.1	Vertical takeoff propulsion system . . . . .	26
2.2.3.2	Cruise propulsion system . . . . .	28
2.2.3.3	Power source (battery) . . . . .	29
2.2.4	Mechanical Design . . . . .	30
2.2.4.1	Building a 3D mechanical model . . . . .	30
2.2.4.2	Payload 3D models . . . . .	31
2.2.4.3	Available machining techniques . . . . .	34
2.2.4.4	Wing fixation method . . . . .	35
2.2.4.5	Fuselage . . . . .	37

<b>3 Autopilot Design</b>	<b>45</b>
3.1 Mathematical Model . . . . .	45
3.1.1 Equations of Motion(Rigid-Body dynamics) . . . . .	45
3.1.1.1 Kinetics <sup>1</sup> . . . . .	45
3.1.1.2 Kinematics <sup>2</sup> . . . . .	47
3.1.1.3 Resultant Nonlinear model . . . . .	47
3.1.2 Linearization . . . . .	49
3.1.3 Parameters . . . . .	49
3.2 Quad-copter phase Autopilot . . . . .	50
3.2.1 Autopilot design . . . . .	50
3.2.1.1 Attitude controller . . . . .	50
3.2.1.2 Altitude Controller . . . . .	55
3.2.1.3 Attitude & Altitude controllers together in action . . . . .	56
3.2.1.4 Line-of-sight (LOS) Guidance . . . . .	57
3.2.2 Simulation using LabVIEW . . . . .	63
3.2.3 Commercial Autopilot . . . . .	65
3.2.3.1 Commander . . . . .	66
3.2.3.2 Navigator . . . . .	66
3.2.3.3 Position controller . . . . .	66
3.2.3.4 Attitude & Rate Controller . . . . .	68
3.2.3.5 Estimator . . . . .	70
3.2.3.6 Output drivers . . . . .	70
3.2.4 Gains Comparison . . . . .	71
3.3 Quad-plane phase cruise controller . . . . .	71
<b>4 Vision navigation and Obstacle Avoidance</b>	<b>73</b>
4.1 Stereo Vision . . . . .	73
4.2 Obstacle Avoidance . . . . .	82
4.2.1 Local methods . . . . .	82
4.2.2 Global methods . . . . .	84
<b>5 Implementation</b>	<b>86</b>
5.1 ROS . . . . .	86
5.1.1 What is ROS . . . . .	86
5.1.1.1 History of ROS . . . . .	86
5.1.1.2 ROS Philosophy . . . . .	87
5.1.2 Pros and Cons of ROS . . . . .	89
5.1.2.1 Advantages: . . . . .	89
5.1.2.2 Disadvantages: . . . . .	89
5.1.3 Example . . . . .	89
5.1.4 Summary of some important features in ROS . . . . .	91
5.2 Proof of Concept . . . . .	97
5.2.1 State Estimation (Rover) . . . . .	97
5.2.1.1 Software . . . . .	99
5.2.1.2 Localization . . . . .	100
5.2.2 Control (half quad) . . . . .	101
5.2.2.1 Results & Conclusions . . . . .	103
5.3 px4 and Mavros . . . . .	104
5.3.1 Introduction . . . . .	104
5.3.2 User guides to Pixhawk . . . . .	105
5.4 Ground Station . . . . .	112
5.4.1 Introduction . . . . .	112
5.4.1.1 What's a Ground Station . . . . .	112
5.4.1.2 Ground Station software . . . . .	112
5.4.2 User guides to Qgroundcontrol . . . . .	112
5.4.3 User interface of Qground . . . . .	112

<sup>1</sup>the study of forces and moments on system in motion

<sup>2</sup>the study of motion without regard to forces or moments

5.4.4	User guides to Mission Planner . . . . .	113
5.4.5	User interface of Mission Planner . . . . .	115
<b>A</b>	<b>ROS Examples</b>	<b>119</b>
A.1	publisher.py . . . . .	119
A.2	subscriber.py . . . . .	119
A.3	publisher.cpp . . . . .	119
A.4	subscriber.cpp . . . . .	119
A.5	ROS services using python . . . . .	119
A.5.1	Server . . . . .	119
A.5.2	node uses server . . . . .	119
A.6	ROS actions using python . . . . .	119
A.6.1	Action server . . . . .	119
A.6.2	node uses action . . . . .	119

# List of Figures

1.1.1 the Curtiss Autoplane . . . . .	5
1.1.2 Uber's vertiport design . . . . .	6
1.1.3 EHANG184 in Dubai . . . . .	7
1.1.4 Technology Breakdown – patent filing activity . . . . .	8
1.2.1 Boeing's self-flying taxi . . . . .	9
1.2.2 Uber Air conceptual video . . . . .	9
1.2.3 Top ten companies filing patents in the flying cars space . . . . .	10
1.3.1 scaled flying taxi prototype . . . . .	10
2.1.1 Tandem wing plane . . . . .	12
2.1.2 The 2 airfoils . . . . .	12
2.1.3 Pressure distribution along NACA0015 with difference angle of attack . . . . .	13
2.1.4 Pressure distribution along Eppler423 with difference angle of attack . . . . .	13
2.1.5 Polar graph of NACA0015 . . . . .	14
2.1.6 Polar graph of Eppler423 . . . . .	14
2.1.7 Configuration . . . . .	15
2.1.8 Investigating stability . . . . .	15
2.1.9 Investigating lift . . . . .	15
2.1.10 NACA0015 Lift and Cm-alpha . . . . .	16
2.1.11 Eppler423 Lift and Cm . . . . .	16
2.1.12 Pressure distribution along NACA0015 at certain angle of attack and Reynolds number . . . . .	18
2.1.13 Polar graph of NACA0015 at a certain Reynolds number=100,000 . . . . .	19
2.1.14 Lift Vs alpha and Cm Vs alpha . . . . .	19
2.1.15 Cl Vs alpha and Cl/Cd Vs alpha . . . . .	19
2.1.16 Cl <sup>2</sup> /Cd Vs alpha . . . . .	20
2.1.17 Fz vs alpha and Cm vs alpha . . . . .	20
2.1.18 pole map for Longitudinal mode . . . . .	21
2.1.19 Short period mode . . . . .	22
2.1.20 zero at very small time. Motion around steady flight mode . . . . .	22
2.1.21 pole map for Lateral mode . . . . .	23
2.1.22 Lateral response modes . . . . .	24
2.2.1 KDE2315XF-965 . . . . .	27
2.2.2 Hover motor's performance data . . . . .	28
2.2.3 Rimfire .10 35-30-1250 Outrunner Brushless . . . . .	29
2.2.4 Lithium Polymer Battery (11.1 V, 5200 mAH- 35C) . . . . .	30
2.2.5 1/1000 Scaled Passenger . . . . .	31
2.2.6 Chair . . . . .	31
2.2.7 Pixhawk and its mount 3D model . . . . .	32
2.2.8 ZED stereo camera 3D model . . . . .	32
2.2.9 Nano Jetson KIT . . . . .	32
2.2.10 Hovering quad motor 3D model . . . . .	33
2.2.11 Pusher motor 3D model . . . . .	33
2.2.12 Battery 3D model . . . . .	33
2.2.13 Laser Cutter . . . . .	34
2.2.14 CNC foam cutter . . . . .	34
2.2.15 3D printer . . . . .	35

2.2.16variable incidence fixation assembly . . . . .	35
2.2.17variable incidence fixation Front view . . . . .	36
2.2.18variable incidence fixation at zero incidence . . . . .	36
2.2.19variable incidence fixation at -4 degrees incidence . . . . .	36
2.2.20variable incidence fixation at 4 degrees incidence . . . . .	37
2.2.21Constant incidence fixation at 2 degrees incidence . . . . .	37
2.2.221st mechanical design iteration . . . . .	38
2.2.232nd mechanical design iteration . . . . .	39
2.2.243rd mechanical design iteration . . . . .	40
2.2.254th mechanical design iteration . . . . .	41
2.2.265th mechanical design iteration . . . . .	41
2.2.276th mechanical design iteration . . . . .	42
2.2.287th mechanical design iteration . . . . .	42
2.2.298th mechanical design iteration . . . . .	43
2.2.309th mechanical design iteration . . . . .	43
2.2.3110th mechanical design iteration . . . . .	44
 3.1.1 motor configuration . . . . .	46
3.2.1 Autopilot design . . . . .	50
3.2.2 Roll Controller . . . . .	51
3.2.3 1 deg desired Roll input . . . . .	51
3.2.4 10 deg desired Roll input . . . . .	52
3.2.5 Pitch Controller . . . . .	52
3.2.6 1 deg desired pitch input . . . . .	52
3.2.7 10 deg desired pitch input . . . . .	53
3.2.8 Yaw Controller . . . . .	53
3.2.9 Yaw Step response and control action . . . . .	54
3.2.10 10 deg desired yaw input . . . . .	54
3.2.11desired roll = 5 deg , desired pitch = 0 , desired yaw = 5 . . . . .	55
3.2.12Altitude Controller . . . . .	55
3.2.13Altitude Step response and control action . . . . .	56
3.2.144m altitude input . . . . .	56
3.2.15desired roll = -5 deg , desired pitch = -5 deg , desired altitude = -4 m ( Z = 4 m) . . . . .	57
3.2.16LOS . . . . .	57
3.2.17LOS block diagram . . . . .	58
3.2.18X guidance . . . . .	58
3.2.19along-track control (trial : 1) . . . . .	59
3.2.20LabVIEW front panel . . . . .	59
3.2.21along-track control (trial : 2 increasing P) : high steady-state error . . . . .	60
3.2.22along-track control (trial : 2 decreasing D) : high steady-state error and the settling time is increased (as expected but we only wanted see the effect) . . . . .	60
3.2.23along-track control (trial : 3 adding small I term to eliminate steady-state error) : settling time is highly increased . . . . .	61
3.2.24along-track error control (final trial) : increasing P and D . . . . .	61
3.2.25Y guidance . . . . .	62
3.2.26X,Y guidance waypoint(X=0 , Y=1 , Z=-1) . . . . .	62
3.2.27X,Y Guidance waypoint(X=1 , Y=1 , Z=-1) . . . . .	63
3.2.28LabVIEW front panel . . . . .	63
3.2.29LabVIEW blockdiagram . . . . .	64
3.2.30LabVIEW Guidance blockdiagram . . . . .	64
3.2.31LabVIEW altitude BD . . . . .	64
3.2.32LabVIEW attitude controller BD . . . . .	65
3.2.33PX4 architecture from PX4 website . . . . .	66
3.2.34 Cascaded control architecture from “Nonlinear Quadrocopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello,2013” . . . . .	67
3.2.35Position Controller . . . . .	68
3.2.36Cascaded control architecture from “Nonlinear Quadrocopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello,2013” . . . . .	68

3.2.37 Attitude&rate controller . . . . .	70
3.3.1 Cruise Controller . . . . .	72
4.1.1 Ref[?] MAV uses 4 cameras to get pose estimate real-time on-board . . . . .	77
4.1.2 forward-facing stereo configuration . . . . .	78
4.1.3 final configuration . . . . .	79
5.1.1 ROS architecture . . . . .	86
5.1.2 Why ROS? . . . . .	87
5.1.3 ROS graph of Stanford's STAIR robot nodes. . . . .	88
5.1.4 ROS example . . . . .	90
5.1.5 ROS msgs . . . . .	94
5.1.6 ROS actions . . . . .	96
5.1.7 comparison between ROS parameters, Dynamic Reconfigure , Topics , Services , and Actions	96
5.2.1 Differential Drive Robot (DD robot) . . . . .	97
5.2.2 differential drive model . . . . .	97
5.2.3 differential_drive package . . . . .	98
5.2.4 1D Bi-rotor conf1 Block Diagram . . . . .	101
5.2.5 1D Bi-rotor conf2 Block Diagram . . . . .	102

# List of Tables

2.2.1 quad-plane advantages . . . . .	25
2.2.2 Non optimized payload . . . . .	25
2.2.3 Optimized payload . . . . .	26
2.2.4 Secondary payload . . . . .	26
2.2.5 Hover motor specifications . . . . .	27
2.2.6 power of motors to vehicle's weight . . . . .	29
2.2.7 Pusher motor specifications . . . . .	29
2.2.8 Battery specification . . . . .	30
4.1.1 different sensors for vision . . . . .	74
4.1.2 ZED stereo camera vs Kinect . . . . .	76

# Chapter 1

# Introduction

## 1.1 FLYING TAXI

A flying taxi is a flying vehicle with a range of 50 to 120 miles, carrying one to four passengers, and cruising at an altitude of 3,000 to 5,000 feet. Based on the current battery technology, the most common commute might be a 50-mile round trip with two short vertical takeoff and landing.

While the idea of personal flying vehicles is old, advancements in technology have reached a point to make this vision a reality. Flying cars, roadable aircrafts, VTOLs and personal air vehicles, are a few of the synonyms for air taxis.

While the race to develop flying cars has started to gain momentum in recent years, the concept has been around for decades. One of the earliest attempts at a flying car was by Glenn Curtiss who built the Curtiss Autoplane, a roadable aircraft in 1917 , that never achieved full flight. Several subsequent attempts were made during the last century to bring flying cars and autonomous aircraft to the market, but recent developments have paved the way for personal air transport to become a technological reality.

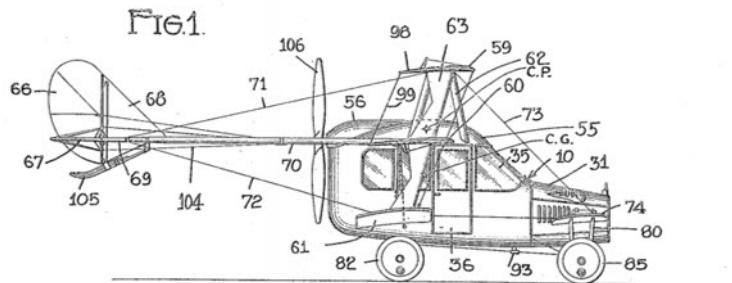


Figure 1.1.1: the Curtiss Autoplane

### 1.1.1 why air taxis are an attractive idea

- Convenience and ease of congestion

Currently, traffic congestion is a major problem in most urban areas and air travel requires planning and time wasted at airports. For commuters in busy cities, air taxis would ease traffic and reduce daily commute times. For air travel, imagine the convenience of being able to travel with your luggage from your home to your local airport to check in for a flight, instead of dealing with airport parking and shuttles. The idea is that air taxis would operate much like the Uber or Lyft app, an individual would be able to request an air taxi to pick them up from a location at their chosen time, and drop them off directly at, or close to, their chosen destination.

- Infrastructure costs

While there is skepticism, the costs associated with building the infrastructure for an air taxi network may be more cost-efficient than building infrastructure for ground transportation. Without having to build and maintain roads and the peripheral structures to support them, the infrastructure required for air taxis costs are far less than that for ground transportation.

Air taxis, in particular, those that have vertical take-off and landing capabilities, don't require runways. Existing infrastructures can be exploited for air taxi use. Uber Elevate proposes "Vertiports" that have charging facilities, hubs, and pads for take-off and landing that could be developed from existing unused land, tops of parking garages or existing helipads.



Figure 1.1.2: Uber's vertiport design

- To reduce emissions and improve safety

Advances in batteries and electric motors have changed the economics of flying car or air taxi transport. For example, today's helicopters are widely regarded as too fuel inefficient and require expensive and time-consuming maintenance, making them impractical for a mass-scale business. But battery-powered electric motors eliminate more complex transmission systems that need a lot of time and knowledge to repair. They also will be much quieter than helicopters, making them more acceptable to city dwellers.

Meanwhile, improvements in artificial intelligence mean paying a pilot to ferry passengers around won't be necessary. And that frees up another seat for a paying passenger.

Additionally, government initiatives are helping to make these technologies a potential commercial reality. For example, Britain has declared a ban on all diesel and petrol cars and vans from 2040. Dubai has outlined a self-driving strategy that aims to carry out "25% of its passenger transportation with the help of autonomous means of transport."

### 1.1.2 obstacles we need to pass

- Energy limitations

Design engineers face a big hurdle with today's all-electric aircraft in that the available batteries pack much less energy per unit of weight than jet fuel. The performance gap we need to bridge is huge about 40 times less [energy than is needed], even if we consider the best batteries available. These energy limitations become especially acute in smaller craft. Electric motors partly compensate for this disadvantage by being more efficient [than jet-fueled propulsion] in converting energy into power, but a considerable gap in performance still remains. The result is that an aircraft would need either a four- or five-time improvement in specific energy density or it would need to carry a very heavy battery pack to approach the performance of current airliners.

In order to solve this problem, the design engineers explore non-traditional approaches to address the problem. By replacing some of the conventional materials of the aircraft structure with active battery materials, we can add energy while also saving on mass and volume. We can either use the structure of the airplane to store energy or build batteries that can also serve a structural [aerodynamic] function.

- the public is ready to fly aboard an air taxi with no pilot?

Human pilots are more prone to mishaps than an artificial intelligence operator. However, in a recent poll, more than half the people polled responded that they wouldn't buy a pilotless flight

ticket even if it was cheaper than the alternative. It is important to first familiarize the public with commercial self-piloting crafts starting with autonomous cargo planes, which could demonstrate how the systems can safely fly from point A to B. The next step is to remove pilots gradually, shifting from a two-person cockpit to one person monitoring the system before phasing out humans entirely.

- our current air traffic control system doesn't accommodate air taxis

Urban airspace is open for business today, and with air traffic control systems exactly as they are, a vertical-takeoff-and-landing service could be launched and even scaled to possibly hundreds of vehicles. A successful, optimized on-demand urban VTOL operation, however, will necessitate a significantly higher frequency and density of vehicles operating over metropolitan areas simultaneously. Air traffic control is going to have to evolve, and new ATC systems will be needed to handle these extra vehicles, especially if a city were to add multiple hubs and potentially hundreds of air taxis.

We do not envision air taxis flying in the same space as commercial airliners. A separate air space corridor for air taxis will probably be created. Air taxis could be managed through a server-request-like system that can de-conflict the global traffic, while allowing UAVs and VTOLs to self-separate any potential local conflicts with visual flight rules, even in inclement weather.

### 1.1.3 Flying taxi categories

Lineberger's team at Deloitte<sup>1</sup> see three flying vehicle categories in the near future, according to a report released in January 2018.

- Passenger drones: A passenger drone is expected to be an electric or hybrid-electric quadcopter (although some may have more than four rotors) that can be used to move people or cargo between both established and on-demand origination and destination points. These vehicles can be either manually piloted, remotely piloted, or fully autonomous.

example for the passenger drone is EHANG184 which is considered as The first passenger drone and started to work in Dubai since 2017, Passenger drones would cover short to medium-range distances (up to 65 miles).



Figure 1.1.3: EHANG184 in Dubai

- Traditional flying cars: A traditional flying car would be a vehicle where the driver/pilot can drive the vehicle in its car configuration to an airport, reconfigure the vehicle to an airplane mode, and then fly to a destination airport. It is designed to carry people and fly medium to long distances (50 to 200 miles). Currently, it would need to be operated by a licensed pilot, but it could be made fully autonomous and pilotless/driverless over time. In the near future, flying cars are likely to become VTOL capable.

---

<sup>1</sup>Deloitte is one of the "Big Four" accounting organizations and the largest professional services network in the world by revenue and number of professionals.

- Revolutionary vehicles: Revolutionary vehicles, which are expected to be a combination of passenger drone and traditional flying car, would be fully autonomous vehicles that can start or stop anywhere, with speed and range (distances greater than 200 miles) beyond passenger drones and the traditional flying cars. These vehicles have advanced VTOL capability and therefore can land and take off from almost anywhere because they may not require an established airport/vertiport. These would likely be piloted by a licensed pilot initially, but they could be made fully autonomous over time.

#### 1.1.4 Technology areas dominating air taxi technology

Analysis of the patents allows us to identify the technology areas that are dominating this space. Technology related to propulsion methods or systems have the most patents filed with more than double the number of patents than those relating to control systems and the design of the aircraft. Fewer patents exist on landing platforms or landing systems for aircrafts and wing design. These areas could present potential new opportunities for engineers to enter this space

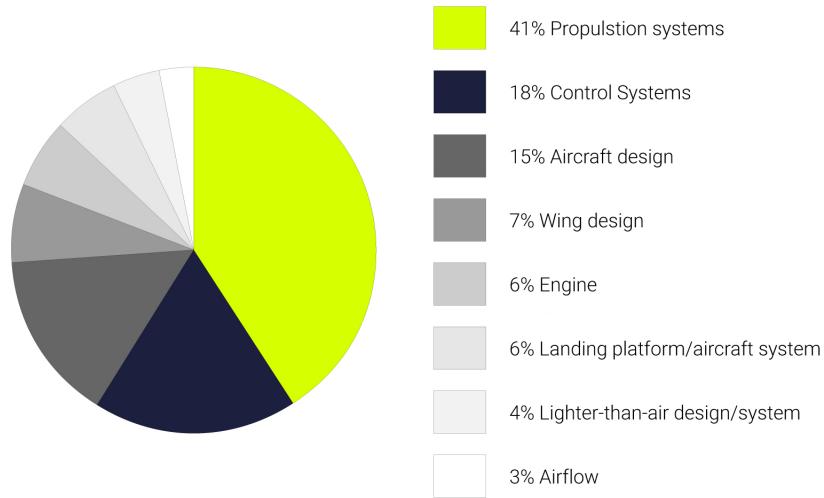


Figure 1.1.4: Technology Breakdown – patent filing activity

from the previous discussion we can conclude that the design of air taxis varies between manufacturers but there are common features. Most have been initially designed for pilot controlled use but some envisage that these aircrafts will be autonomous, thus reducing the costs associated with training and employing pilots.

Vehicles have also been designed to allow for vertical take-off and landing using tilttable electric engines that allow the engines to be rotated depending on the flight mode. Several different designs for propulsion technologies have been used, but now all engineers believe that the successful air taxi design will not use the rotary-wing design of today's helicopters. Instead, it will be a fixed-wing craft with vertical-takeoff-and-landing capabilities (quad plane)

## 1.2 the key players in the flying cars space

To capture this new market, teams across multiple transportation sectors have emerged. Not surprisingly, some of the same industries that are developing electric propulsion and autonomy on the ground are converging on air mobility. Uber has teamed up with Textron's Bell, Boeing's Aurora Flight Sciences and others for Uber Air. Airbus is working with Volkswagen's Audi to design a flying car that can drive on the road and take flight. Airbus is separately developing Vahana, an autonomous single-seat aircraft, and CityAirbus, a four-seat aircraft. Volocopter is building an air taxi with the help of Mercedes-Benz parent Daimler with plans to launch a service in three to five years. And Joby Aviation has received investments from Toyota and JetBlue Airways (JBLU).

Boeing just took an important step toward making them a practical reality. The aircraft maker has completed the first test flight of its autonomous electric VTOL aircraft, verifying that the machine can

take off, hover and land. It's a modest start, to put it mildly the taxi has yet to fly forward, let alone transition from vertical to forward flight modes. That still puts it ahead of competitors, though, and it's no mean feat when the aircraft existed as little more than a concept roughly one year ago.



Figure 1.2.1: Boeing's self-flying taxi

Uber has said it's aiming to begin service in several cities, by 2023 with early demonstrations set for 2020. An Uber Air conceptual video depicts a customer booking a flight on her smartphone, heading to an "Uber Skyport" at the top of a high-rise, and boarding an aircraft with multiple rotors.



Figure 1.2.2: Uber Air conceptual video

And if we look for the players with the strongest patent portfolios are Sikorsky Aircraft (now owned by Lockheed Martin), Boeing, Airbus and LTA. Unsurprisingly, large aircraft manufacturers such as Boeing and Airbus appear in the top ten companies. However, it is notable that smaller players such as Larry Page's start-up Zee Aero and Rafi Yoeli's Urban Aeronautics are also building strong patent portfolios. More interesting may be the appearance of Honeywell International and General Electric in the top ten. They are leading players in electric or hybrid-electric aircrafts, however, neither seem to be particularly active in manufacturing flying cars, or forming partnerships to provide systems or components for other manufacturers. Honeywell International has patented several control and propulsion systems for VTOLs or unmanned aerial vehicles, and General Electric have several patents on engines for short take-off or vertical take-off and landing vehicles. It may be interesting to see whether these experienced patent licensing companies exploit their portfolios as the industry expands and matures.



Figure 1.2.3: Top ten companies filing patents in the flying cars space

### 1.3 Prototype

This project can be assumed as a scaled quadplane prototype to a real flying taxi, many reasons can drive to build a prototype. first of all the cost of real model is very huge in addition, to make a test with real model is very difficult and risky, but this prototype has a complete software architecture like the real model. It can do obstacle avoidance and vision navigation, use EKF to fuse GPS and visual odometry to get accurate position estimations also to fuse it with INS for better attitude estimations given that the vision navigation is used in the compatible operational design domain (ODD).



Figure 1.3.1: scaled flying taxi prototype

building a prototype isn't easy process especially that you have to use the same sensors like the real model (stereo camera & lidar), so the next chapters will discuss the process of building this prototype hardware and software:

- Chapter 2: discuss the mission ,sizing and aerodynamic design.
- Chapter 3: discuss the mathematical model of quadplane and include comparison between self-built autopilot and commercial one.
- Chapter 4: discuss main concepts of vision navigation and obstacle avoidance algorithms.
- Chapter 5: discuss the used software (ROS,PX4,QGC)
- Chapter 6: discuss the flight test results.

# Chapter 2

# Aircraft Design

## 2.1 Aerodynamic Design

### 2.1.1 Introduction

Before designing the vehicle, goals, mission and constraints must be defined well to achieve the highest performance and efficiency.

- **Mission:** Flying taxi should take-off as a quad-copter. When it gains the required and safe altitude, the flying taxi starts the quad-plane phase by turning the pusher on to achieve the required cruise speed which is specified from range, endurance and available power considerations. The flying taxi continues its quad-plane phase until it's near from the landing point, it switches to the quad-copter mode and gradually lowering its altitude with visual odometry generated to aid the GPS for accurate position estimation and the obstacle avoidance algorithm is on to guide the flying taxi through a safe path to the landing point.
- **Constraints:**
  - Max. takeoff weight about 3 Kg, Where preliminary weight estimation of vehicle contains components about 2.8 Kg.
  - Max. width of vehicle 0.8 meter, This constraint has been selected to satisfy mission, be easy to manufacture and to suit the wind tunnel width for future work in testing.
  - Minimum length for easy landing at any place. XFLR5 program is chosen to design and analyse the aerodynamic parts of the flying taxi starting from airfoil design to the stability analysis of the complete plane.

### 2.1.2 Wing configuration

There are a lot of wing configuration but the suitable configuration for our vehicle constraints and mission should be chosen. Since one of the important constraint of the flying taxi is to have small size, small span constraint should be achieved with generating the required lift so tandem wing plane is preferred to be used to get greater lift than conventional wing, as it has two independent source of lift (two wings) to be able to manage the takeoff weight.

Design parameters of Tandem wing plane:

- **Stagger (St):** The distance between main wing and second wing at a position of 1/4 chord.
- **Gap (G):** The vertical distance between main and second wings.
- **Decalage ( $\delta = \alpha^{\omega} - \alpha^P$ ):** The relative angle of attack between two angles of attack for each wing.

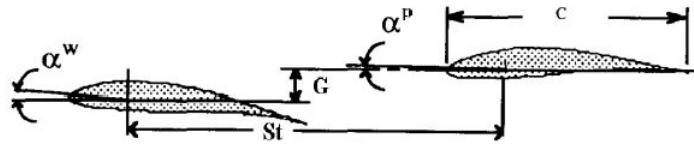


Figure 2.1.1: Tandem wing plane

### 2.1.3 Design Criterion

#### 2.1.3.1 Airfoil selection

In this section there are two constraints:

- The thickness of the standard foam sheets (in some cases it was little than the camber of the airfoil)
- Highly cambered airfoils are difficult to be manufactured using foam cutter machine (less accuracy).

The selection of the Airfoil should consider these constraints along with the requirements. xflr5 program has a built-in library of NACA series, but another airfoils can be imported.

First, airfoils that have high lift and low Reynolds number; such as (Eppler421, Eppler423, MH14, NACA0018, NACA0015, Clark(Y) and NACA632615) are analysed, then suitable airfoil is chosen according to the previously stated constraints and requirements. So the airfoils that satisfy requirements are:

- Eppler423 (higher lift)
- NACA0015 (easy to manufacture)

The two airfoils are analysed using panel method with 150 panel for smooth results.

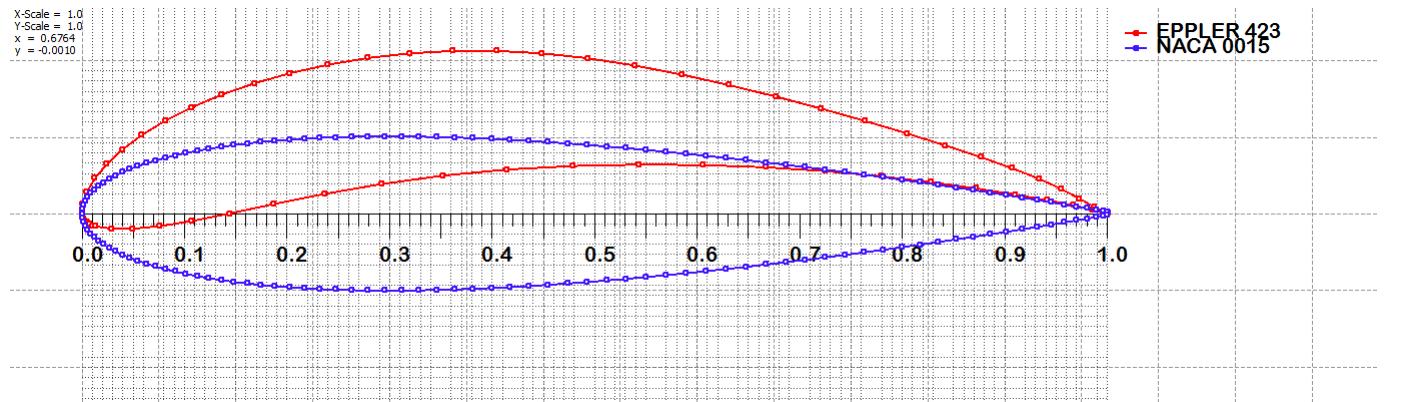


Figure 2.1.2: The 2 airfoils

Shown below the boundary layer thickness and pressure distribution along the two selected airfoils with change at angle of attack from: (0 to 6 & 0 to -4) degree :

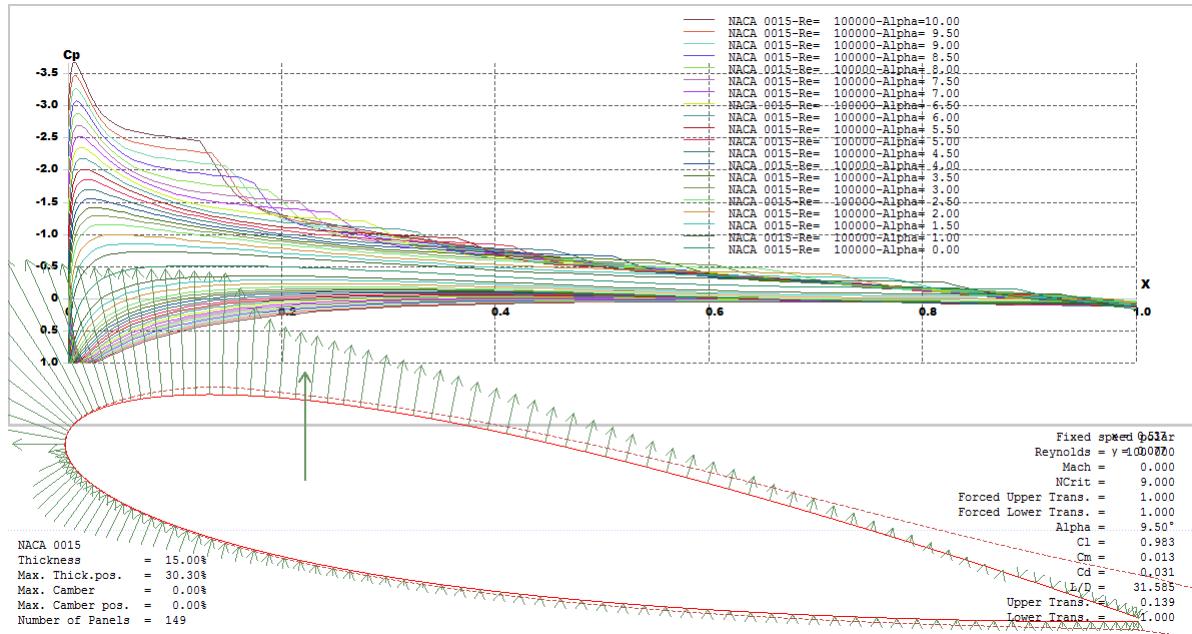


Figure 2.1.3: Pressure distribution along NACA0015 with difference angle of attack

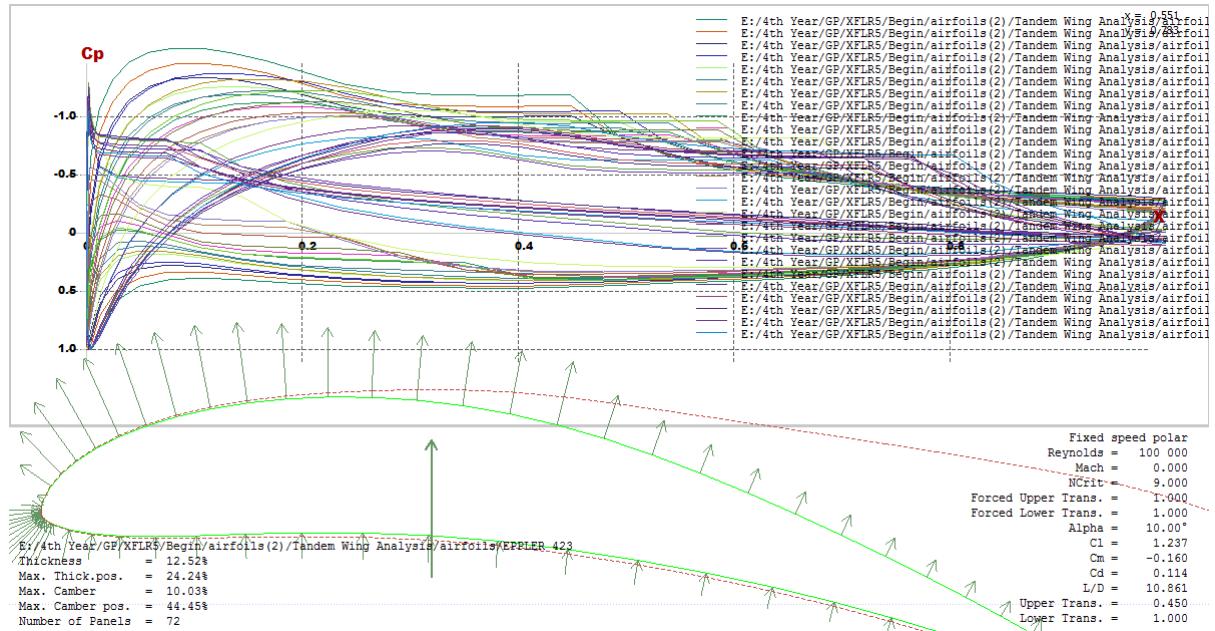


Figure 2.1.4: Pressure distribution along Eppler423 with difference angle of attack

We make analysis at different Re (from 20,000 to 1,000,000) and see plots on polar figure:

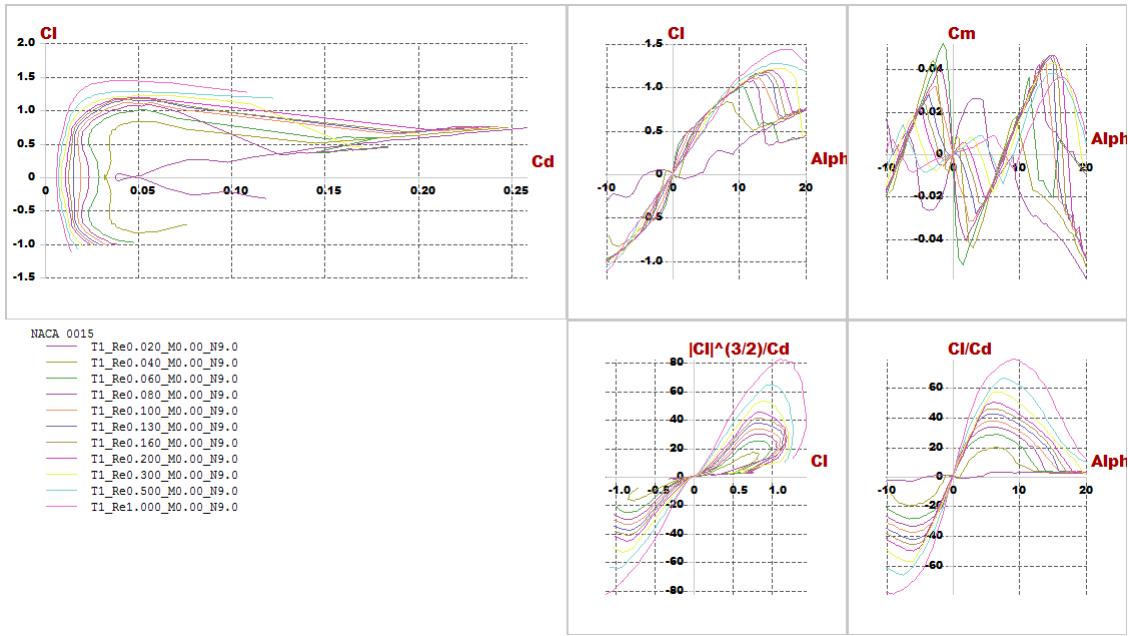


Figure 2.1.5: Polar graph of NACA0015

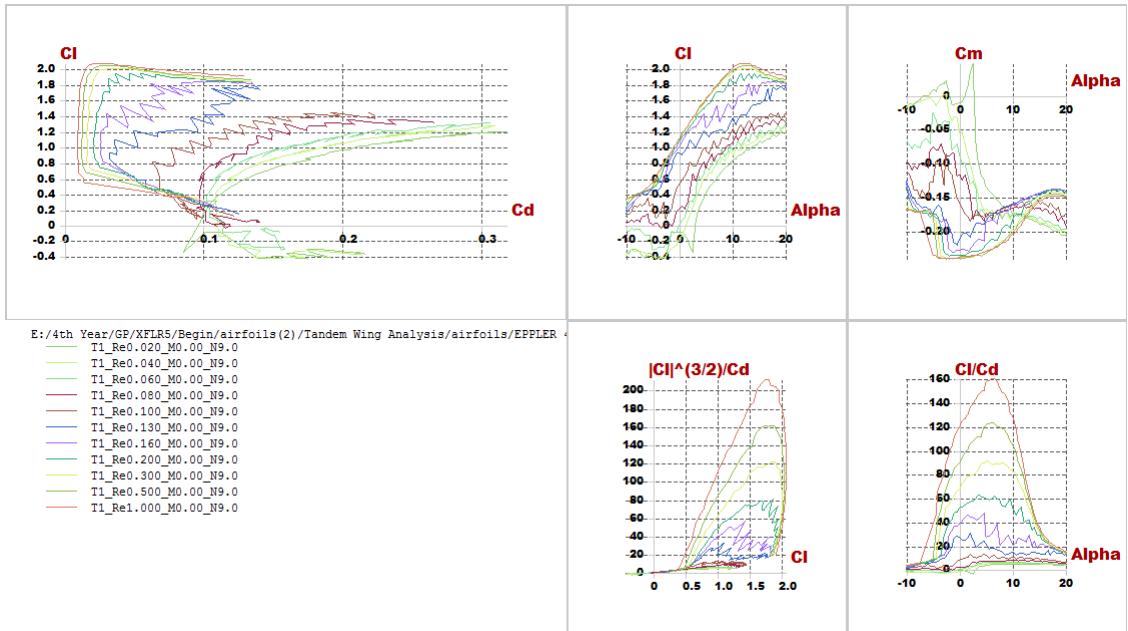


Figure 2.1.6: Polar graph of Eppler423

#### 2.1.4 Aerodynamic analysis:

Now after choosing air foils, the wing will be designed. After trying different trials considering constraints and requirements, the best configuration has been chosen then stability and lift produced is checked.

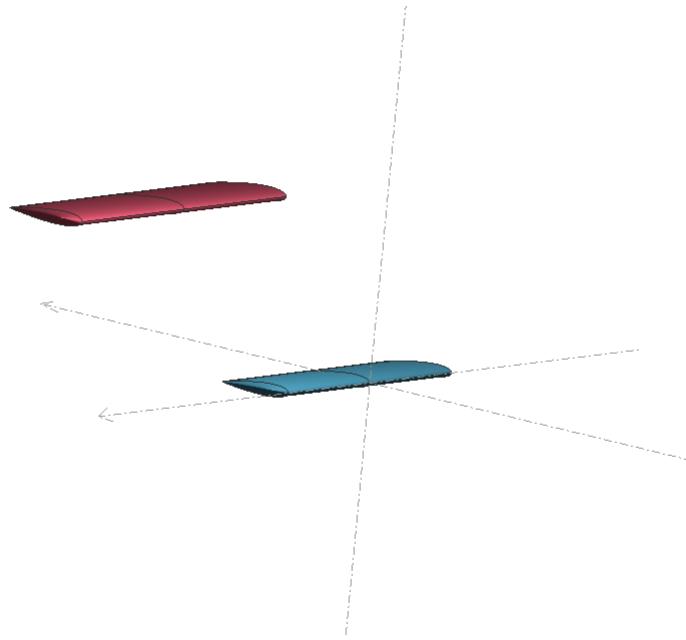


Figure 2.1.7: Configuration

#### 2.1.4.1 Investigating goals and requirements:

- For stability:  $\frac{dC_m}{d\alpha} < 0$ :

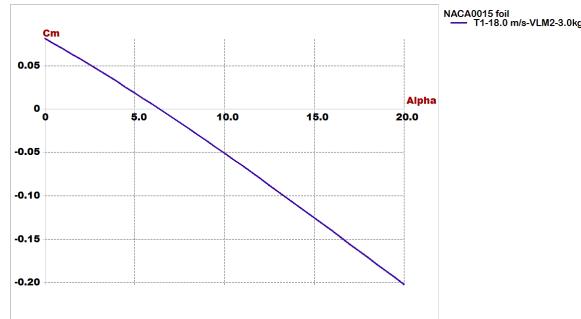


Figure 2.1.8: Investigating stability

- For lift: At trim angle the lift must carry the vehicle

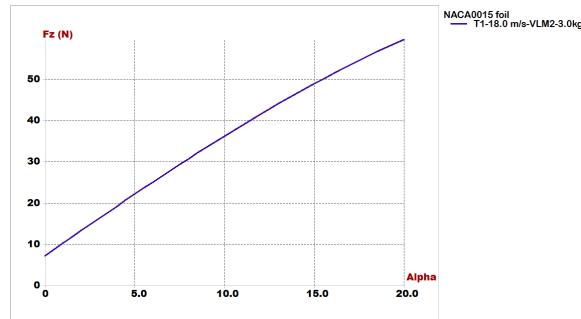


Figure 2.1.9: Investigating lift

### 2.1.4.2 Design parameters

In this part stability of quad-plane is checked and the airfoil design parameters (incidence angle, CG position, cruise speed and dimensions of wings) is chosen

Parameter	Effect on the vehicle
Span	proportional with lift
Chord	proportional with lift
Cruise speed	proportional with lift
Arm	Static stability
Incidence angle	Static stability
Position of CG	Static stability
Position of second wing	Static stability

### 2.1.4.3 Results:

- NACA0015

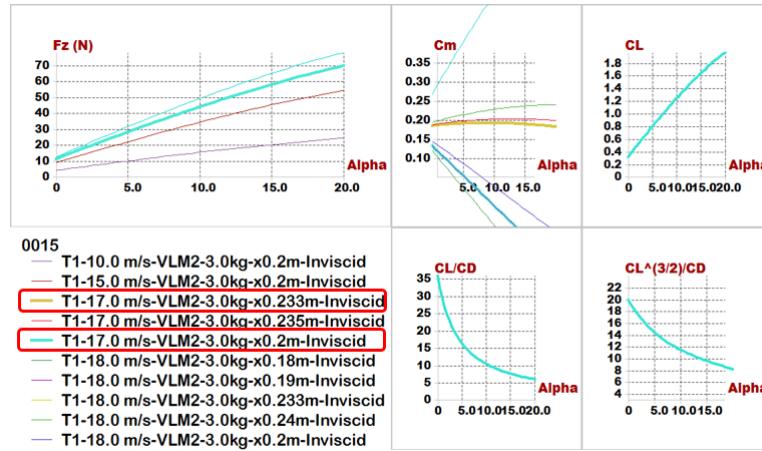


Figure 2.1.10: NACA0015 Lift and Cm-alpha

- Eppler423

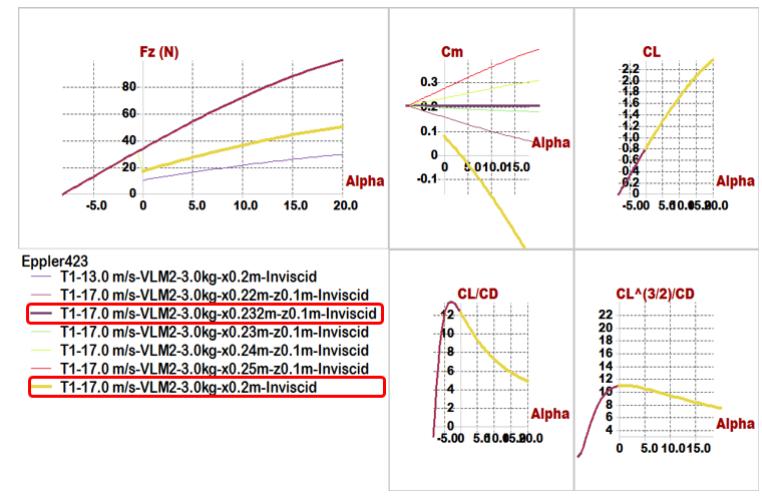


Figure 2.1.11: Eppler423 Lift and Cm

- Comparison between 2 air foils:

Comparisons	Model 1 (NACA0015)	Model 2 (Eppler423)
Position of Main wing	$x = 0 z = -0.05 \text{ m}$	$x = 0 z = -0.05 \text{ m}$
Position of Second wing	$x = 0.45 \text{ m} z = 0.1 \text{ m}$	$x = 0.45 \text{ m} z = 0.15 \text{ m}$
Span	0.8 m	0.6 m
Chord	0.25 m	0.2 m
Aspect ratio	3.2	3
Cruise speed	17 m/s	17 m/s
Incidence of Main wing	5 degree	3 degree
Incidence of Second wing	2 degree	-8 degree
Trim angle	5.1 degree	3.6 degree
$X_{cg}$	0.19 m	0.2 m
$X_{Np}$	0.233 m	0.25 m
Degree of Static Stability (S.M.)	17.2%	25%

Conclusion of results:

1. For configuration:

- First of all, there is illustration of the main reason to select tandem wing plane to get greater lift than conventional wing, because we have two independent source of lift (two wings).
- Then now we can say the reason of vertical distance (Gap) between 2 wings that can affect the resulting vortex and vortex interactions that the wing spacing has a significant effect on the resultant flow field and aerodynamic forces, finally we selected this distance to prevent wings from weak as a wing spacing is decrease, vortex structure at last wing became elongated and spread due to interactions with the front wing.
- Difference at dimension between 2 wings to equivalent to moment results from wings

So this configuration help to more stable quad-plane that increase stability due to wing spacing.

2. For incidence angle:

All models have been built it before almost of them have a negative incidence angle for tail not exceed -3 degree, So highlighting at the incidence of second wing in two models; in model (1) has a positive incidence angle, And model (2) has a large negative incidence angle.

3. For Static margin:

That known the S.M. must be (from 5 to 20 %), and the S.M. in model (1) is 17.2 that is acceptable range, but at model (2) has 25%.

4. For Lift:

We find trim angle of attack at 5.1 degree, and we go to  $F_z$  VS  $\alpha$  graph get lift that carry aircraft we get  $F_z=28.8(\text{N})$  at 5.1 degree, which carry 2.93 Kg maximum tack-off weight, all of calculations reference to 2.7 Kg maximum tack-off weight.

**So, model (1) has been selected because model (1) is more effective than model (2)**

### 2.1.5 Optimization for aerodynamic stability:

In this phase C.G. position is forced to be in the middle of the vehicle to equalize the value of moments resulting from two wings to optimize the ( $C_m - \alpha$ ) graph, PX4 requirements and control requirements, easy to manufacturing and high accuracy results during assembly. First, making several trials with constraints of stability:

- Select a set of parameters which can be optimized .
- Then set the parameters to constant values and iterate with one variable, to modify slope of stability from positive to negative slope.

- Concluded variables that affect the slope:
  - Arm
  - The height of second wing
  - Incidence of angles of 2 wings
  - Geometry of 2 wings specially wing chord
- Put position of C.G. and the arm, then change variables.

In this part the design parameters will be reduced to 6 parameters replacement 7 parameters, because of constant position of CG, **By trial and error after fail some trials we get the best performance by change parameters:**

Comparisons \ Models	Model (NACA0015)
Position of Main wing	x = 0 , z = 0
Position of Second wing	x = 0.6 m , z = 0.3 m
Span of main wing	0.6 m
Span of second wing	0.75 m
Chord of main wing	0.2 m
Chord of second wing	0.22 m
Aspect ratio	3.2
Cruise speed	18 m/s
Incidence of Main wing	3 degree
Incidence of Second wing	2 degree
Trim angle	6.7 degree
X <sub>CG</sub>	0.36 m
Z <sub>CG</sub>	0.15 m
X <sub>NP</sub>	0.383 m
Degree of Static Stability (S.M.)	11.5%
Reynolds number	About (100,000 to 200,000)

### 2.1.6 Final results:

- Foil characteristics:

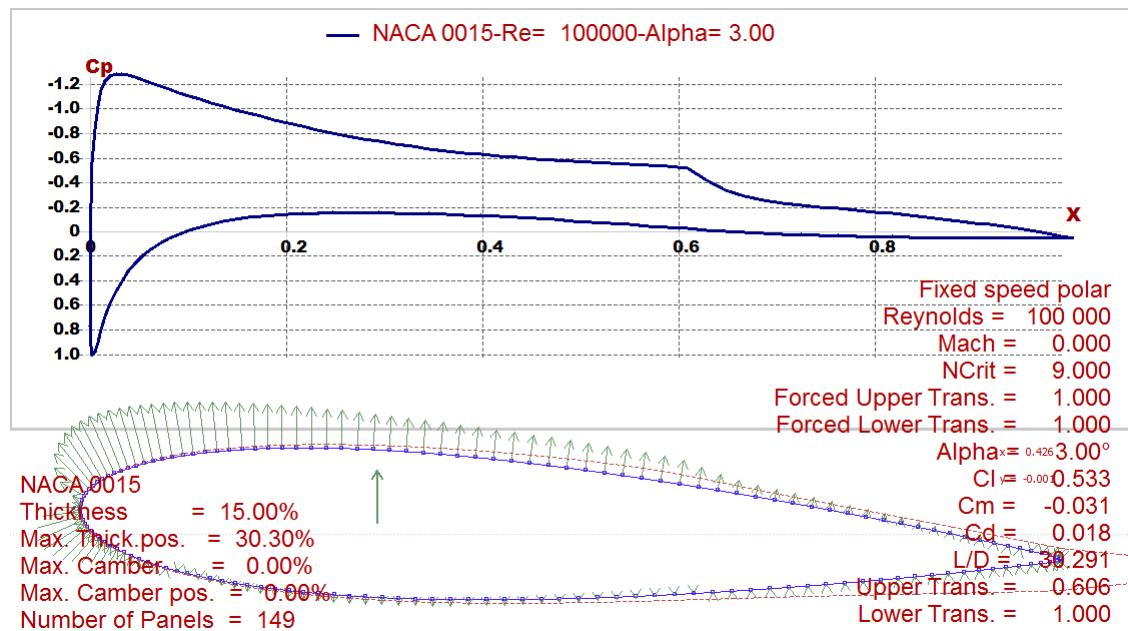


Figure 2.1.12: Pressure distribution along NACA0015 at certain angle of attack and Reynolds number

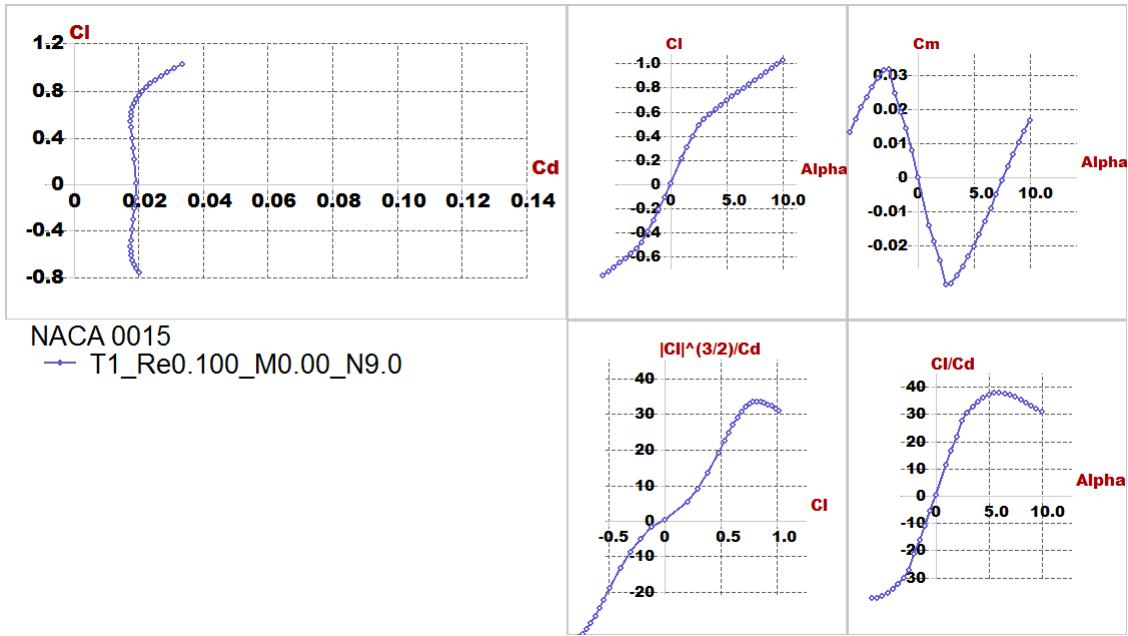


Figure 2.1.13: Polar graph of NACA0015 at a certain Reynolds number=100,000

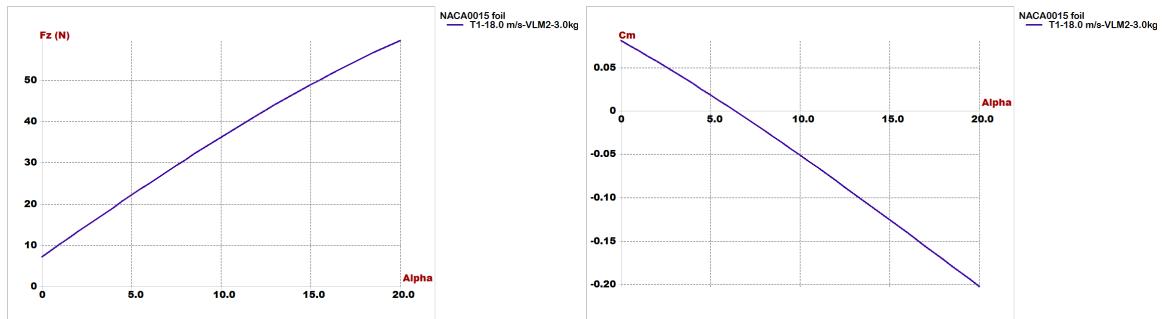


Figure 2.1.14: Lift Vs alpha and Cm Vs alpha

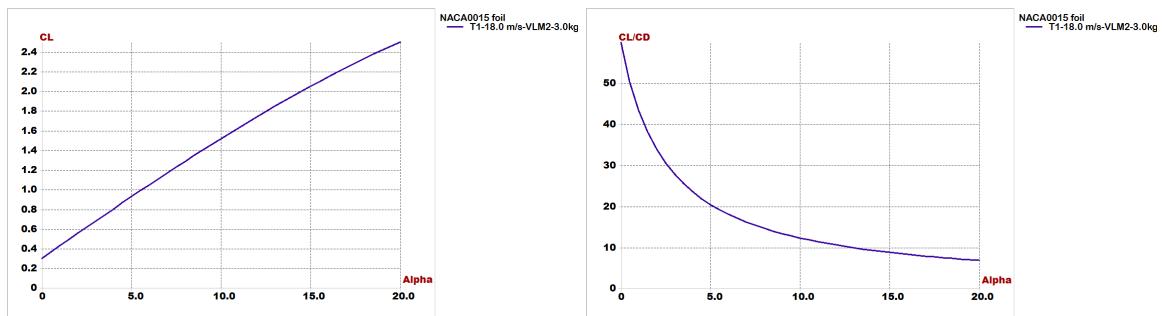
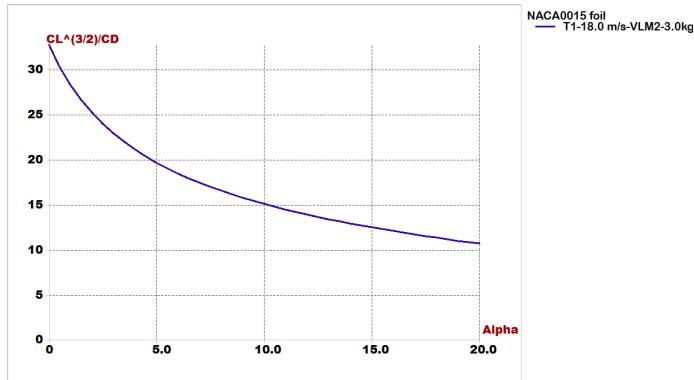
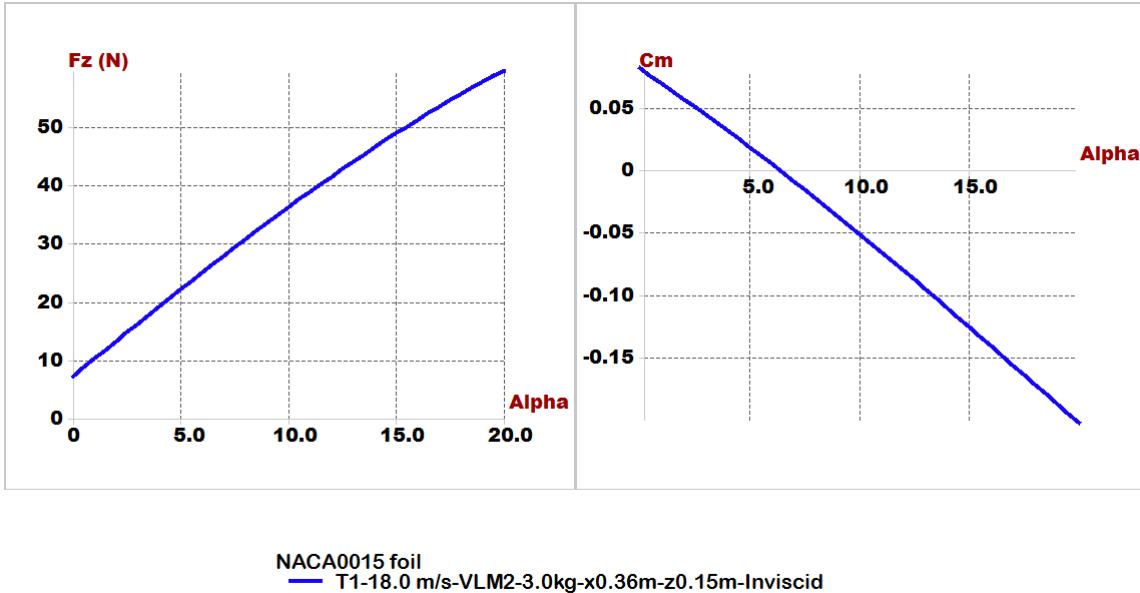


Figure 2.1.15: Cl Vs alpha and Cl/Cd Vs alpha

Figure 2.1.16:  $CL^{\frac{3}{2}}/Cd$  Vs alpha

Finally we must check for our results by focus on 2 important figures, (longitudinal static stability and lift generated from airfoil) , slope of  $C_m$  Vs alpha must has negative slope, and the value of lift at trim angle of attack must be carry the weight of vehicle:

Figure 2.1.17:  $F_z$  vs  $\alpha$  and  $C_m$  vs  $\alpha$ 

### 2.1.7 Stability Analysis:

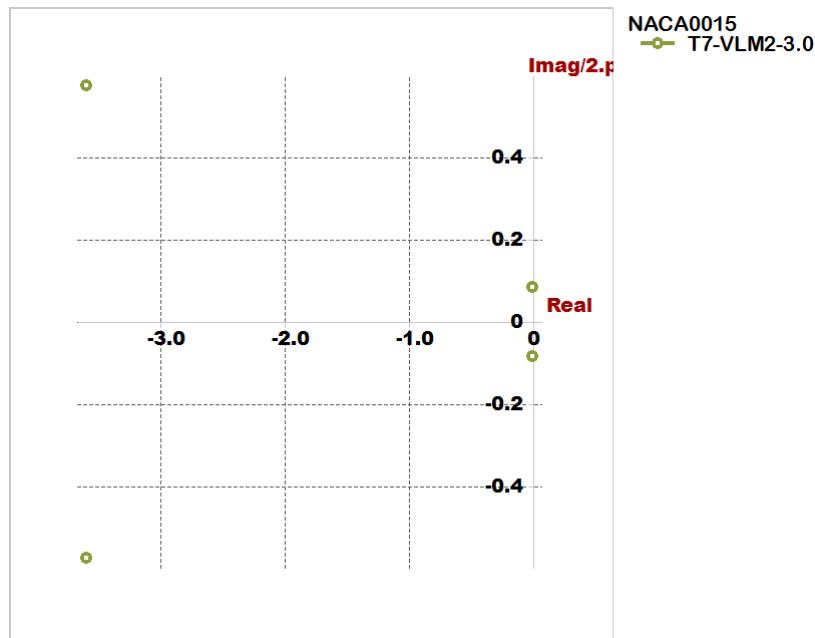
In this section longitudinal and lateral stability had been checked as the software use the stability derivatives of the model and get the TF to check the long period and short period for longitudinal motion and the different modes for lateral motion and here are the results

#### 2.1.7.1 For longitudinal:

- Type1: short period mode.
- Type2: Pitching.
- Type3,4: motion around steady state flight

- Longitudinal derivatives from Xflr5

$X_u = -0.17768$	$C_{xu} = -0.12599$
$X_w = 0.39794$	$C_{xa} = 0.28216$
$Z_u = -3.0683$	$C_{zu} = -0.00056674$
$Z_w = -9.8047$	$C_{la} = 6.9519$
$Z_q = -1.9192$	$C_{lq} = 13.608$
$M_u = 5.3903e-07$	$C_{Mu} = 1.911e-06$
$M_w = -0.22008$	$C_{Ma} = -0.78024$
$M_q = -1.20971$	$C_{Mq} = -42.886$
Neutral Point position = 0.38245 m	



#### Longitudinal modes

Eigenvalue:	-3.586+	-3.6i		-3.586+	3.6i		-0.001648+	-0.5215i		-0.001648+	0.5215i
Eigenvector:	1+	0i		1+	0i		1+	0i		1+	0i
	17.12+	-3.849i		17.12+	3.849i		-0.1554+	-0.0017i		-0.1554+	0.0017i
	-0.9854+	-3.258i		-0.9854+	3.258i		0.02773+	0.004069i		0.02773+	-0.004069i
	0.591+	0.315i		0.591+	-0.315i		-0.00797+	0.05314i		-0.00797+	-0.05314i

Figure 2.1.18: zero pole map for Longitudinal mode

- Time response characteristics:

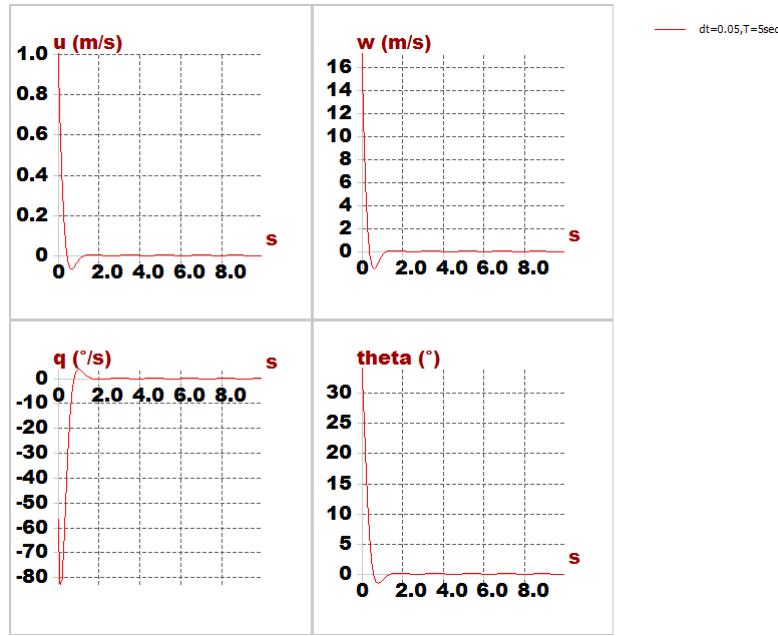


Figure 2.1.19: Short period mode

where  $u$ : horizontal speed.  $w$ : vertical speed.  $q$ : pitch rate.  $\theta$ : pitch angle, We noticed that is heavy damped mode (quick mode), all parameters reached to zero at very small time.

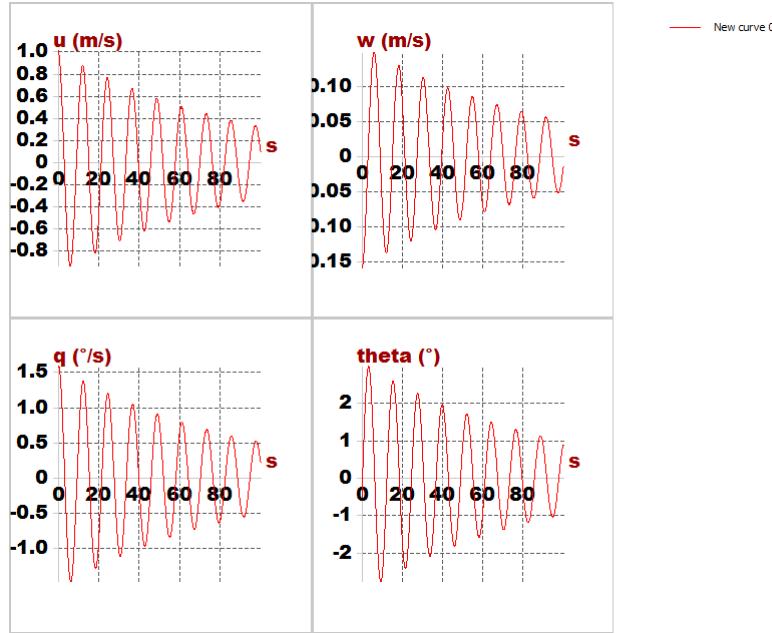


Figure 2.1.20: zero at very small time. Motion around steady flight mode

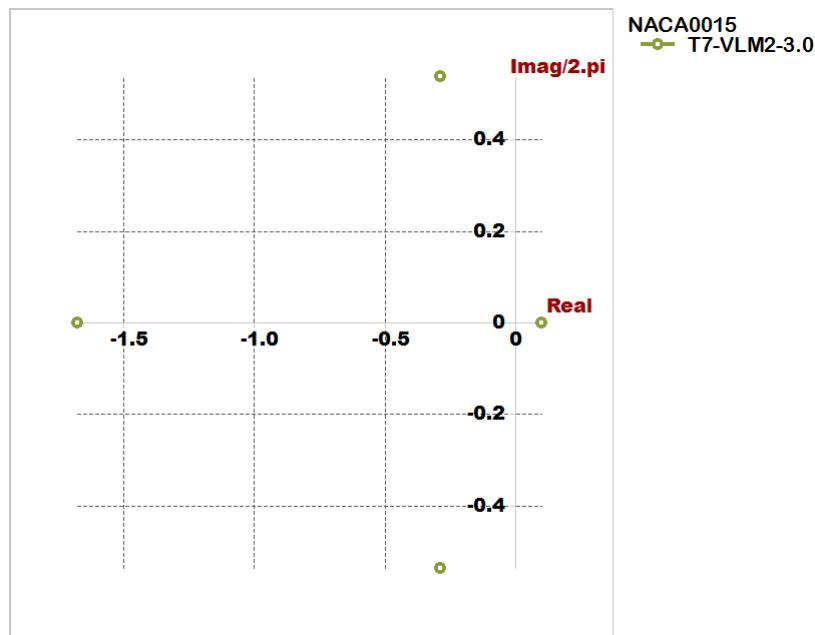
At total time=100 second, and step=0.1 second we see the signal is converge with time.

### 2.1.7.2 For lateral

- Type1: roll damping.

- Type2,3: dutch roll mode.
- Type4: spiral mode.
- **Lateral derivatives from Xflr5**

$Y_V = 1.0832e-27$	$C_{Yb} = 7.6802e-28$
$Y_P = 2.3296e-19$	$C_{Yp} = 5.506e-19$
$Y_r = -1.0915e-20$	$C_{Yr} = -2.5798e-20$
$L_V = 9.9393e-15$	$C_{lb} = 1.1746e-14$
$L_P = -0.22653$	$C_{lp} = -0.89234$
$L_r = 0.082947$	$C_{lr} = 0.32674$
$N_V = 1.1912e-14$	$C_{nb} = 1.4077e-14$
$N_P = -0.048585$	$C_{np} = -0.19138$
$N_r = -3.9184e-13$	$C_{nr} = -1.5435e-12$



#### \_\_\_\_Lateral modes\_\_\_\_\_

Eigenvalue:	-1.672+	0i		-0.2873+	-3.362i		-0.2873+	3.362i		0.1016+	0i
Eigenvector:	1+	0i		1+	0i		1+	0i		1+	0i
	-4.161+	0i		-0.0977+	-0.151i		-0.0977+	0.151i		0.2591+	0i
	1.358+	0i		0.03313+	0.1628i		0.03313+	-0.1628i		1.295+	0i
	2.488+	0i		0.04705+	-0.02504i		0.04705+	0.02504i		2.551+	0i

Figure 2.1.21: zero pole map for Lateral mode

- Time response characteristics

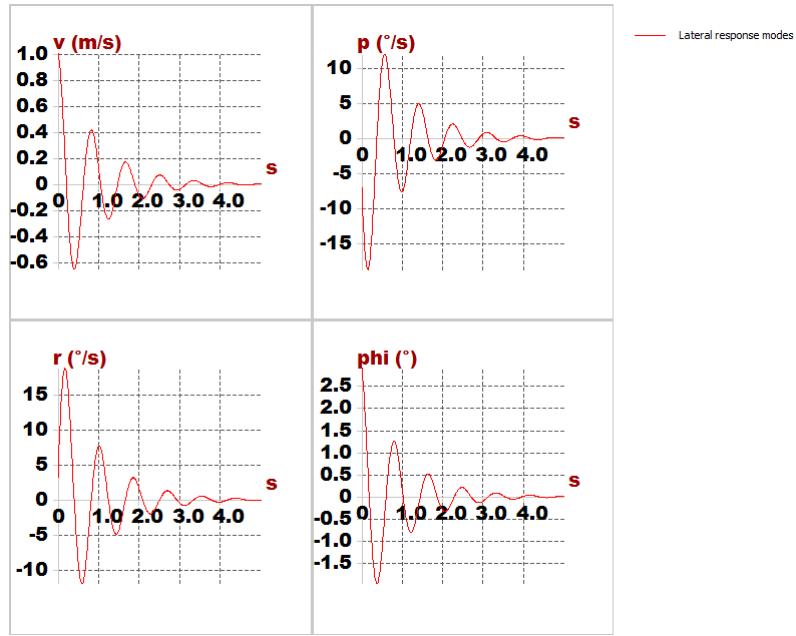


Figure 2.1.22: Lateral response modes

where  $v$ : velocity speed.  $p$ : roll rate.  $r$ : yaw rate.  $\phi$ : bank angle.

- Non-dimensional Stability Derivatives calculated by program:

$C_{Xu} = -0.12599$	$C_{Yb} = -0.24520$
$C_{Lu} = 0.00057$	$C_{Lb} = -0.05277$
$C_{Mu} = 0.00000$	$C_{Nb} = 0.12824$
$C_{Xa} = 0.28216$	$C_{Yp} = -0.06327$
$C_{La} = 6.95194$	$C_{Lp} = -0.90835$
$C_{Ma} = -0.78024$	$C_{Np} = -0.15792$
$C_{Xq} = -0.24026$	$C_{Yr} = 0.29289$
$C_{Lq} = 13.60765$	$C_{Lr} = 0.38927$
$C_{Mq} = -42.88605$	$C_{Nr} = -0.15451$

## 2.2 Mechanical Design

### 2.2.1 Introduction

#### 2.2.1.1 Why Quad-plane

A quad-plane vehicle is a mixture between fixed wing-plane and a multi-copter, it has both of their advantages

Point Of Comparisons	Fixed-wing	multi-copter	Quad-Plane
vertical takeoff and landing	yes	no	yes
comfortable	uncomfortable	comfortable	comfortable
power consumption	High	Low	Moderate
cruise speed	low	high	high
Range	short	long	long
Endurance	short	long	long

Table 2.2.1: quad-plane advantages

as a conclusion from the previous table, it's obvious that a quad-plane is the best choice to achieve this mission

#### 2.2.1.2 Scaling

scaling is based in payload, assuming a full scaled model carry a payload of 120 kg including a 80 kg passenger, 20 kg luggage and 20 kg navigation control equipment .Since vision equipment masses doesn't change from a full scaled model to a scaled model, control equipment masses are assumed to be a part of the scaled model payload

A 1/100 scaled model is to be designed so that its payload is 1.2 kg.

#### 2.2.1.3 Payload

Payload consists of the components that are loaded on the vehicle to carry out its mission so, the payload will be divided into two sections: control payload and passenger payload

- Before optimization

components	masses (gm)
Passenger payload	
Human model	200
Chair	123
Navigation payload	
XPAL power supply	517
Jetson KIT	431
RP Lidar A1	190
ZED camera	159
Rasspery pi 2	42
PIX4	38
Other electronic devices	300
Total mass	2000 gm

Table 2.2.2: Non optimized payload

- After optimization

It was found that the payload exceeded the scaling assumption, so some optimization is done in the payload.

also due to limited resources of propulsion systems and batteries, further optimization is done and the final payload is:

components	masses (gm)
Passenger payload	
Human model	200
Chair	123
Navigation payload	
nano Jetson KIT	150
ZED camera	159
PIX4	38
Other electronic devices	300
Total mass	970 gm

Table 2.2.3: Optimized payload

#### 2.2.1.4 Endurance

#### 2.2.1.5 Range

### 2.2.2 Sizing

sizing is an iterative process where the VTOL propulsion system is chosen based on the main payload and then both the main payload and the propulsion system (secondary payload) become the payload of the fixed wing. After sizing the fixed wing, the MTOM is calculated and then VTOL propulsion system is re-chosen and so-on til convergence

Due to limited resources of propulsion systems, only one VTOL propulsion system is available so, sizing process will not be iterative

- As a Multi-copter

As a multi-copter, only the propulsion system will be taken in account while the structure will be neglected as the structure will be taken in consideration in fixed wing sizing

component	mass of one piece	no of pieces	total mass
VTOL motors	71	4	284
VTOL propellers	19	4	76
VTOL ESC	25	4	100
Total mass	-	-	460 gm

Table 2.2.4: Secondary payload

- As a fixed-wing

Payload in this section will be the summation of both secondary and main payloads, so the payload is

$$\text{Payload} = \text{Main payload} + \text{Secondary payload} = 970 + 460 = 1430 \text{ gm}$$

### 2.2.3 Propulsion system selection

#### 2.2.3.1 Vertical takeoff propulsion system

- Motors : the rule of thumb in choosing a vertical takeoff propulsion system is that the total propulsive force is twice the weight of the vehicle. See eq

$$T_{motor} \leq \frac{W_{estimated}}{4}$$

**Note:** Due to limited resources of available motors, we had to estimate the maximum takeoff weight from the available set of motors

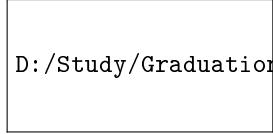


Figure 2.2.1: KDE2315XF-965

Maximum constant current	26+ A
Maximum constants watts	385+ W
No load current	0.5 A
Input Voltage	11.1 V (3S LiPo) - 17.4 V (4S LiHV)
RPM/V (Kv rating)	965
Weight	75 grams

Table 2.2.5: Hover motor specifications

MOTOR VERSION	VOLTAGE LiHV [V]	PROPELLER SIZE	THROTTLE RANGE	AMPERAGE [A] (LOWER IS BETTER)	POWER INPUT [W] [hp] (LOWER IS BETTER)		THRUST OUTPUT [g] [N] [lb] (HIGHER IS BETTER)	RPM [rev/min] (HIGHER IS BETTER)	EFFICIENCY [g/W] [lb/hp] (HIGHER IS BETTER)
KDE2315XF-965 (965kv)	11.6V (3S) 13.1V MAX	9" x 3.0	25.0%	0.6	6	0.01	100 160 270 390 520 700 850	3060 4620 6000 7260 8520 9780 10740	16.67 13.33 11.25 9.75 8.39 7.37 6.69
			37.5%	1.1	12	0.02	120	2160	17.14 28.18
			50.0%	2.1	24	0.03	220	4260	11.58 19.04
			62.5%	3.5	40	0.05	370	5520	9.49 15.60
			75.0%	5.4	62	0.08	520	6680	8.28 13.61
		9" x 4.5DJ	87.5%	8.2	95	0.13	700	7740	7.20 11.84
			100.0%	11.0	127	0.17	850	8820	6.27 10.30
			25.0%	0.7	7	0.01	110	9680	5.74 9.44
			37.5%	1.7	19	0.03	120	10940	17.14 28.18
			50.0%	3.4	39	0.05	220	12100	11.58 19.04
KDEF-UAS35 S.R. ENABLED	15.4V (4S) 17.4V MAX	10" x 3.3	62.5%	5.6	64	0.09	370	13260	18.33 30.14
			75.0%	8.7	100	0.13	530	14420	13.75 22.60
			87.5%	13.0	150	0.20	720	15580	10.97 18.03
			100.0%	16.9	195	0.26	940	16740	9.44 13.47
			25.0%	0.6	6	0.01	110	10940	9.31 15.30
		11" x 3.7	37.5%	1.4	16	0.02	120	12100	8.02 13.18
			50.0%	2.7	31	0.04	220	13260	7.26 11.93
			62.5%	4.6	53	0.07	370	14420	6.56 10.79
			75.0%	7.2	83	0.11	530	15580	5.74 9.44
			87.5%	10.7	124	0.17	720	16740	4.90 8.50
		12" x 4.0	100.0%	14.1	163	0.22	940	17900	4.66 7.65
			25.0%	0.8	9	0.01	110	18060	16.67 27.40
			37.5%	1.8	20	0.03	120	19220	15.00 24.66
			50.0%	3.7	42	0.06	220	20380	11.19 18.40
			62.5%	6.2	72	0.10	370	21540	9.31 15.30
		9" x 3.0	75.0%	9.7	111	0.15	530	22700	8.02 13.18
			87.5%	14.5	168	0.23	720	23860	6.85 11.25
			100.0%	18.5	214	0.29	940	25020	5.51 9.06
			25.0%	1.0	11	0.01	110	26180	17.27 28.40
			37.5%	2.2	26	0.03	120	27340	13.08 21.50
		9" x 4.5DJ	50.0%	4.7	54	0.07	220	28500	10.19 16.74
			62.5%	8.1	93	0.12	370	29660	8.49 13.97
			75.0%	12.4	143	0.19	530	30820	7.13 11.73
			87.5%	17.9	208	0.28	720	32980	6.06 9.96
			100.0%	22.7	263	0.35	940	34140	5.51 9.06
		11" x 3.7	25.0%	0.7	10	0.01	110	35300	13.00 21.37
			37.5%	1.8	27	0.04	120	36460	9.26 15.22
			50.0%	3.4	52	0.07	220	37620	8.27 13.59
			62.5%	5.5	84	0.11	370	38780	7.26 11.94
			75.0%	8.5	130	0.17	530	40940	6.46 10.62
		12" x 4.0	87.5%	13.0	199	0.27	720	42100	5.68 9.34
			100.0%	17.1	262	0.35	940	43260	5.19 8.53
			25.0%	1.0	15	0.02	110	44420	3840 12.00
			37.5%	2.6	40	0.05	120	45580	5520 9.25
			50.0%	4.9	75	0.10	220	46740	7.73 12.71
		9" x 3.0	62.5%	8.3	127	0.17	370	47900	8.28 10.74
			75.0%	12.7	195	0.26	530	49060	5.74 9.44
			87.5%	18.6	286	0.38	720	50220	4.90 8.05
			100.0%	24.5	376	0.50	940	51380	4.39 7.21
			25.0%	0.9	13	0.02	110	52540	14.62 24.03
		10" x 3.3	37.5%	2.1	33	0.04	120	53700	10.30 16.94
			50.0%	4.1	63	0.08	220	54860	7.280 8.57
			62.5%	6.9	106	0.14	370	56020	7.45 12.25
			75.0%	10.7	165	0.22	530	57180	6.42 10.56
			87.5%	15.8	243	0.33	720	58340	5.64 9.27
		11" x 3.7	100.0%	20.9	321	0.43	940	59500	5.05 8.30
			25.0%	1.1	16	0.02	110	60660	3720 15.63
			37.5%	2.6	40	0.05	120	61820	11.00 18.08
			50.0%	5.6	85	0.11	220	63000	8.47 13.93
			62.5%	9.3	142	0.19	370	64160	7.11 11.69
		12" x 4.0	75.0%	14.2	219	0.29	530	65320	6.07 9.98
			87.5%	20.8	319	0.43	720	66480	5.17 8.50
			100.0%	27.4	421	0.56	940	67640	4.66 7.65

Note : performance chart provided under the test conditions listed below. Measurements taken under alternate conditions will affect the final results.

Location : KOE Direct HQ Dynamometer V2 (Bend, Oregon)

Altitude : 3730 ft (1137 m)

Pressure : 30.3 inHg (1026 hPa)

Temperature : 72 °F (22 °C)

Humidity : 35% (Relative)

Figure 2.2.2: Hover motor's performance data

- ESCs

- Propellers set

### 2.2.3.2 Cruise propulsion system

- Motors : According to many references, the rule of thumb in choosing a pusher motor is according to the following table

power/weight	vehicle's category
50-70 watts/pound: 11-15 watts/100g	Minimum level of power for decent performance
70-90 watts/pound; 15-20 watts/100g	Trainer and slow flying scale models
90-110 watts/pound: 20-24 watts/100g	Sport, aerobatic and fast flying scale models
110-130 watts/pound: 24-29 watts/100g	Advanced aerobatic and high-speed models
130-150 watts/pound: 29-33 watts/100g	Lightly loaded 3D models and ducted fans

Table 2.2.6: power of motors to vehicle's weight



Figure 2.2.3: Rimfire .10 35-30-1250 Outrunner Brushless

Maximum constant current	30A
Maximum surge current	35A
Maximum constants watts	333W
Maximum brust watts	390W
No load current	1.2A
Input Voltage	7.4 - 11.1 V (2-3S LiPo)
RPM/V (Kv rating)	1250
Weight	71 grams

Table 2.2.7: Pusher motor specifications

- ESCs
- Propellers set

### 2.2.3.3 Power source (battery)

A MATLAB code is created that roughly calculates the capacity if the battery needed given the endurance time of the mission and the consumption rate of the motors or calculates the mission endurance given the battery's capacity. This code is based on some assumptions. (code is in the appendix of codes)

Assumptions:

1. The working time of the hovering motors is 20% of the mission endurance
2. The working time of pusher motor is the total mission endurance

- Battery used in vehicle

A high quality Lithium battery, low weight /size and high power make it suitable for our application



Figure 2.2.4: Lithium Polymer Battery (11.1 V, 5200 mAH- 35C)

- Battery specification

Capacity	5500 mAh
Configuration	3S1P / 11.1v / 3Cell
Peak discharge	35C
weight	380 gm
size	140x30x30 mm
connector type	T connector

Table 2.2.8: Battery specification

## 2.2.4 Mechanical Design

In this chapter, we will discuss how to build an efficient and sufficient mechanical design for an autonomous flying taxi.

Our design is both sufficient and efficient so that it can withstand all the mechanical loads and vibrations and also carry the payload with a minimum weight of its components, it's also characterized by the good and attractive appearance to attract passenger's attention.

### 2.2.4.1 Building a 3D mechanical model

Solid Works is used to create a 3D model and AutoCAD is used to edit 2D sketches for machining purposes. SolidWorks and AutoCAD are solid modeling computer-aided design (CAD) and computer-aided engineering (CAE) computer programs that runs on Microsoft Windows.

- SolidWorks outputs:

1. Files used in machining (laser cutting and 3D printing)

- 2. Accurate weight estimation
- 3. Gravity center location control
- 4. Some analysis (flow and structural)
- Steps to draw a 3D model in SolidWorks
  1. Define components of the payload, their dimensions, weight and fixation points
  2. Define important dimensions created through aerodynamic analysis process such as: arm, span, chord ...
  3. Define CG position needed for aerodynamic stability
  4. define the components that needs mounts
  5. Clearance needed for wiring and connections
- AutoCAD outputs
  1. edit 2D machining files obtained from SolidWorks
  2. Connected to a CNC Laser cutting machine and modify files into G-code for the machine

#### 2.2.4.2 Payload 3D models

- Human

Assuming a regular Human body weighs 200 kg, a scaled model is used to be the passenger of our vehicle on a scale of 1/1000

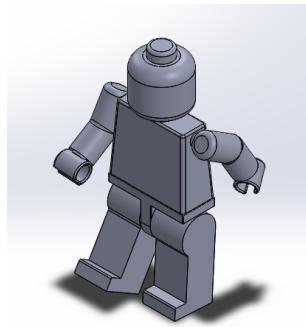


Figure 2.2.5: 1/1000 Scaled Passenger

- Chair

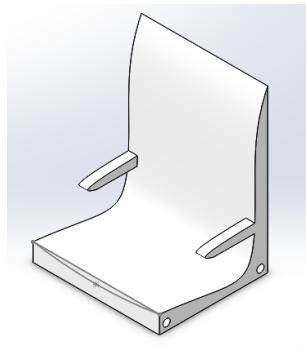


Figure 2.2.6: Chair

- Pixhawk and its mount

Pixhawk position and orientation are the constraints that must be taken in consideration during vehicle design. Pixhawk must be located at the CG of the vehicle and its axes must be the same

axes of the vehicle so, a mount must be designed and manufactured to makes sure that these constraints are satisfied

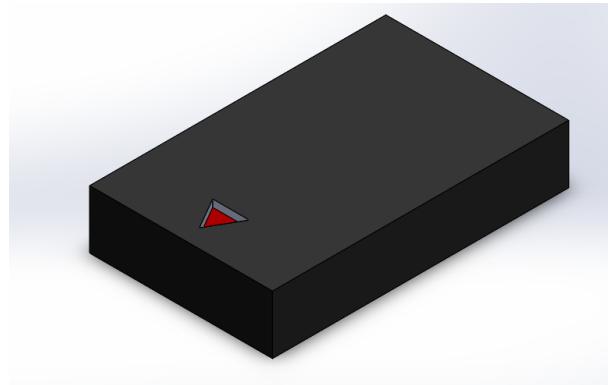


Figure 2.2.7: Pixhawk and its mount 3D model

- ZED camera

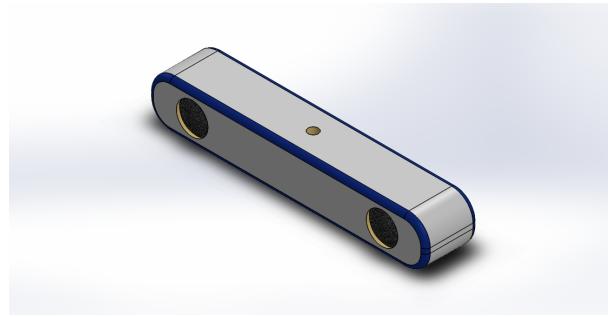


Figure 2.2.8: ZED stereo camera 3D model

- Jetson Nano KIT

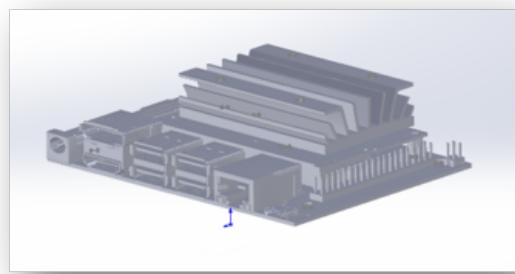


Figure 2.2.9: Nano Jetson KIT

- Quad Motor



Figure 2.2.10: Hovering quad motor 3D model

- Pusher motor

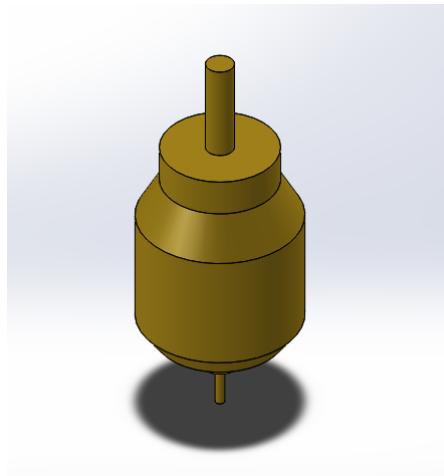


Figure 2.2.11: Pusher motor 3D model

- Battery

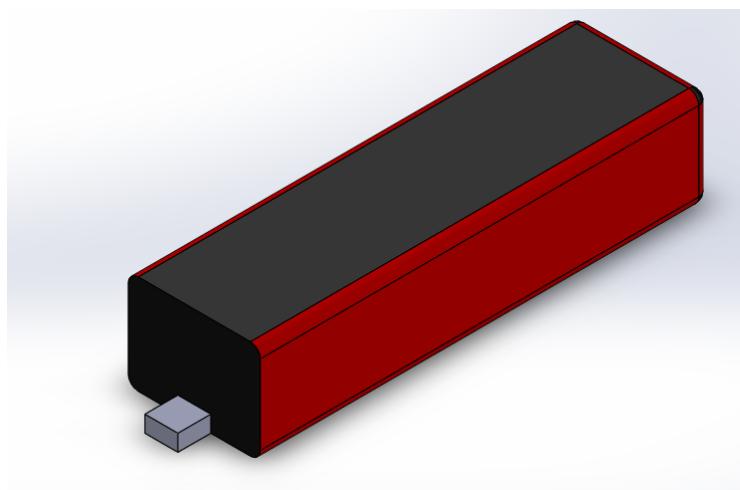


Figure 2.2.12: Battery 3D model

### 2.2.4.3 Available machining techniques

available machining techniques has put some other constrains on the design process so, available machines and their capabilities must be taken in consideration during design process.

- Laser cutter

Only cuts foam and wood sheets with maximum size of (90\*50\*1 cm)

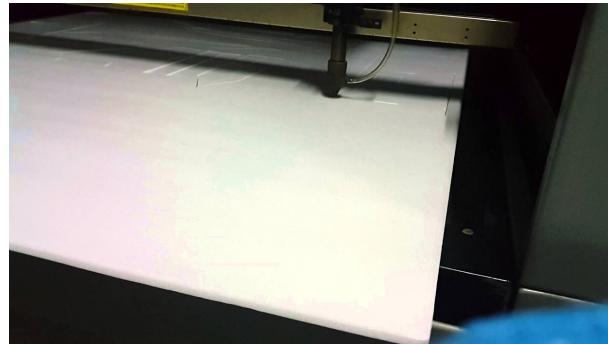


Figure 2.2.13: Laser Cutter

- CNC foam cutter

Forms wings form foam using hot wire with moderate accuracy

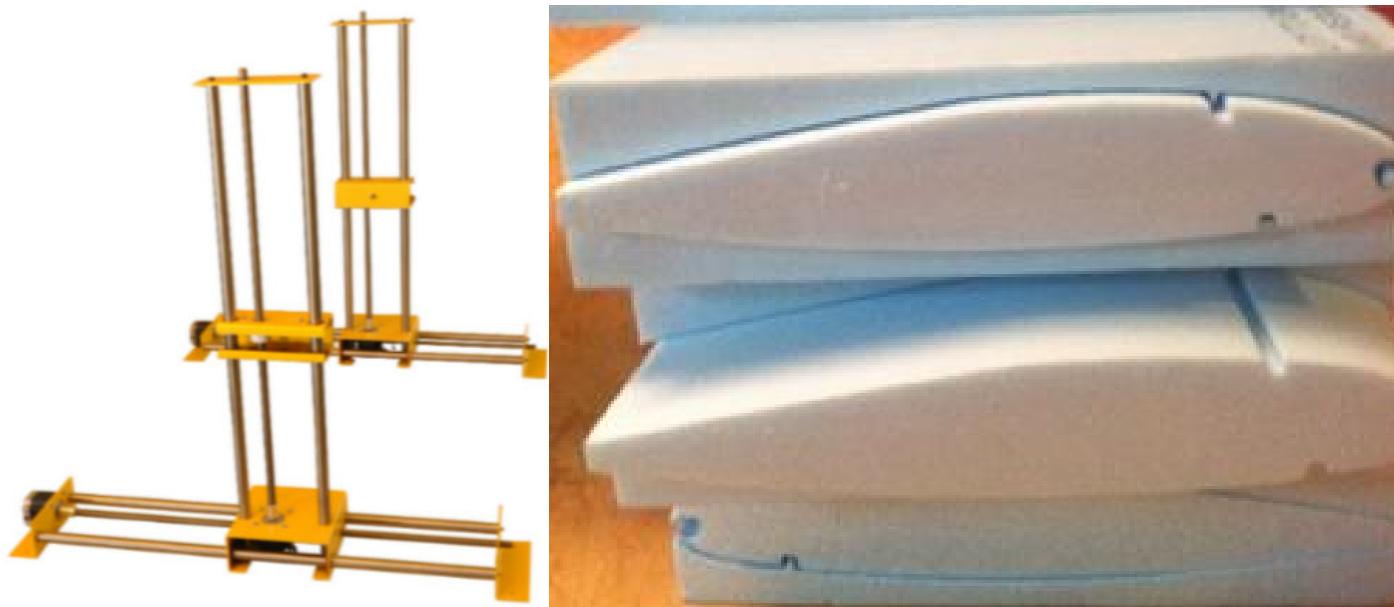


Figure 2.2.14: CNC foam cutter

- A 3D printer (20\*10\*20 cm)

the available 3D printer is with small dimensions and low both quality and accuracy

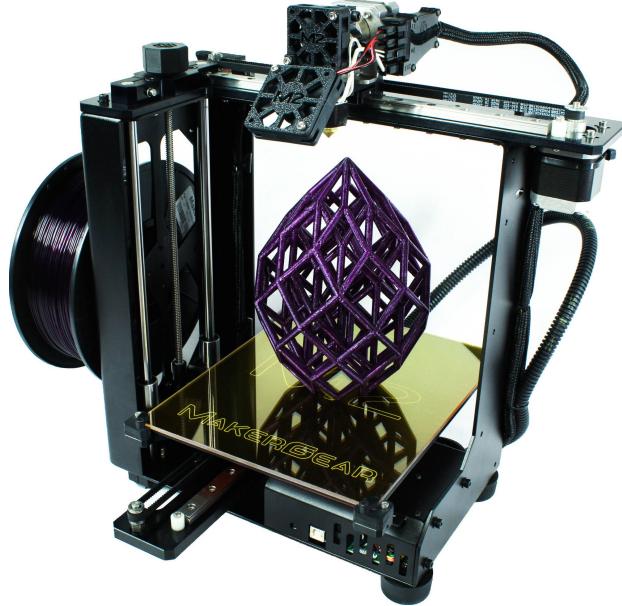


Figure 2.2.15: 3D printer

#### 2.2.4.4 Wing fixation method

- Variable incidence angle fixation method

In order to achieve the ability of tuning incidence angle during wind tunnel testing, a fixed point and a movable point must be made where the movable point is used to adjust incidence and the fixed point is used as a pinned support which prevents displacements and allow rotation.

Variable incidence fixation method is explained in the following figures:

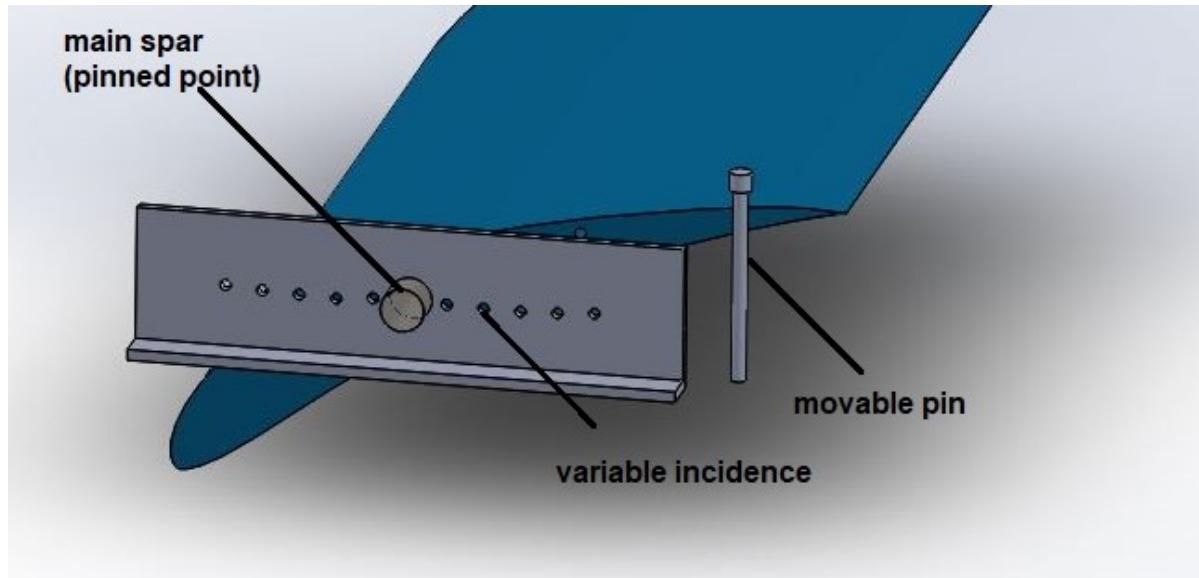


Figure 2.2.16: variable incidence fixation assembly

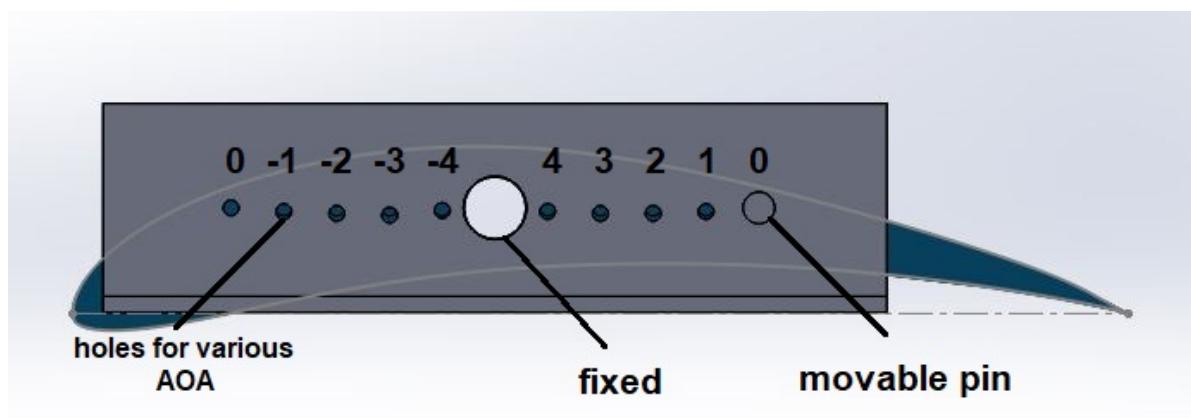


Figure 2.2.17: variable incidence fixation Front view

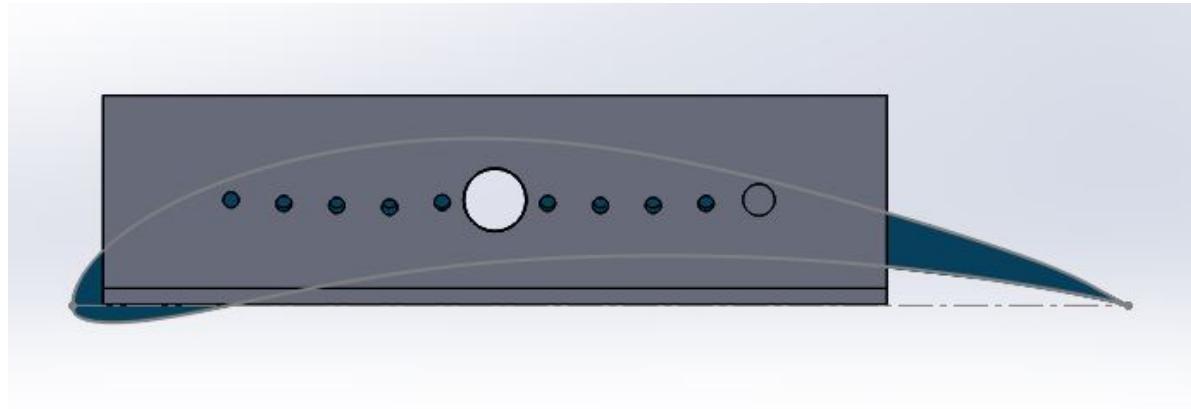


Figure 2.2.18: variable incidence fixation at zero incidence

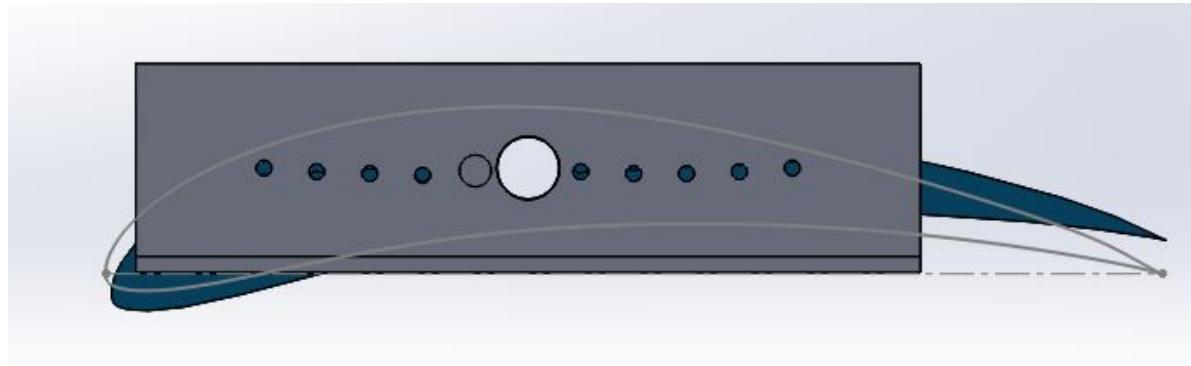


Figure 2.2.19: variable incidence fixation at -4 degrees incidence

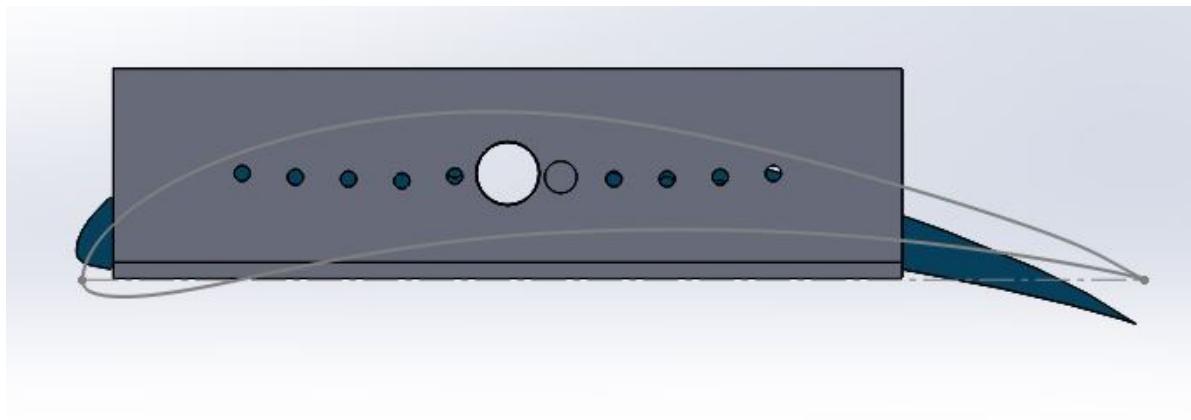


Figure 2.2.20: variable incidence fixation at 4 degrees incidence

- Constant incidence angle fixation method

Due to complicity of manufacturing and implementing the previous method, a constant incidence angle fixation method is used as explained in the following figures:

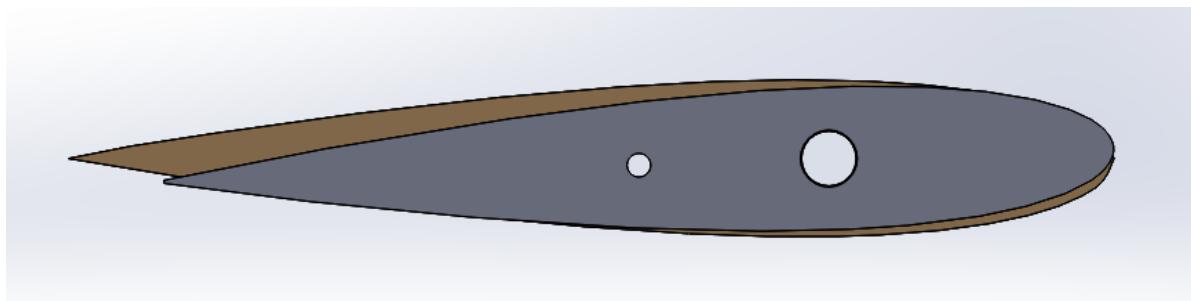


Figure 2.2.21: Constant incidence fixation at 2 degrees incidence

#### 2.2.4.5 Fuselage

Fuselage is the main structure of the aircraft which holds all other components and substructures together and also carries the payload.

After Aerodynamic analysis and bringing our CG in the middle between the four motors, a mechanical design must be done depending on Fuselage. Many iterations are designed, manufactured and tested for reliability , sufficiency and efficiency and in this part we will discuss some of these iterations and our final mechanical design

**NOTE:** after the eighth iteration, new set of motors with higher propulsive power was available increasing the maximum takeoff weight so, the ability of creating more stiff designs increased

- First iteration :This iteration was manufactures and tested and found to be very efficient but it has one fetal disadvantage, that the tractor propeller covers the vision of the ZED camera so, we had to switch to a pusher motor instead of tractor motor

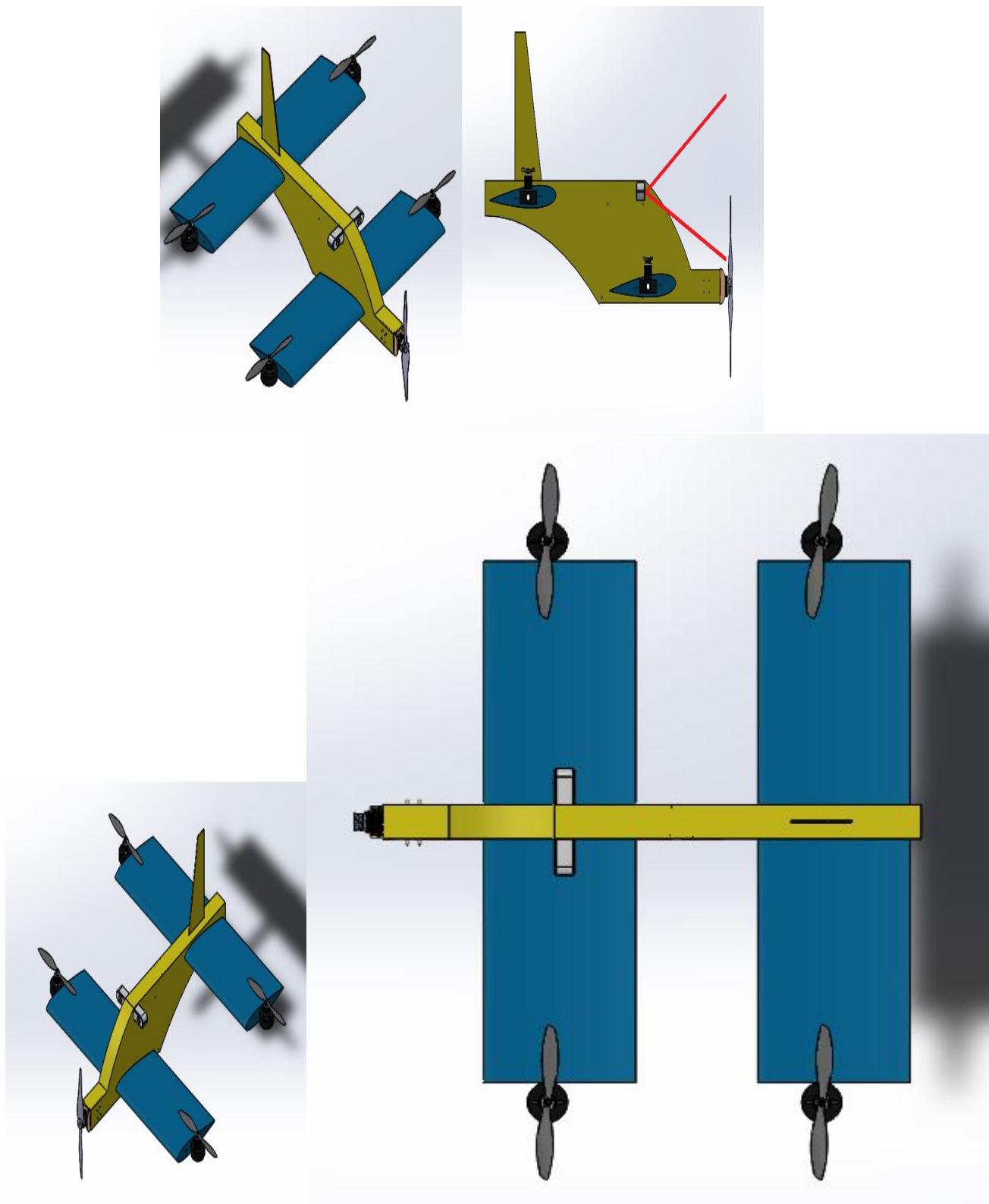


Figure 2.2.22: 1st mechanical design iteration

- Second iteration

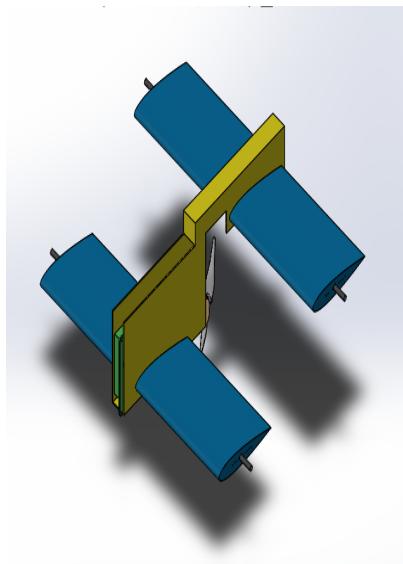


Figure 2.2.23: 2nd mechanical design iteration

- Third iteration

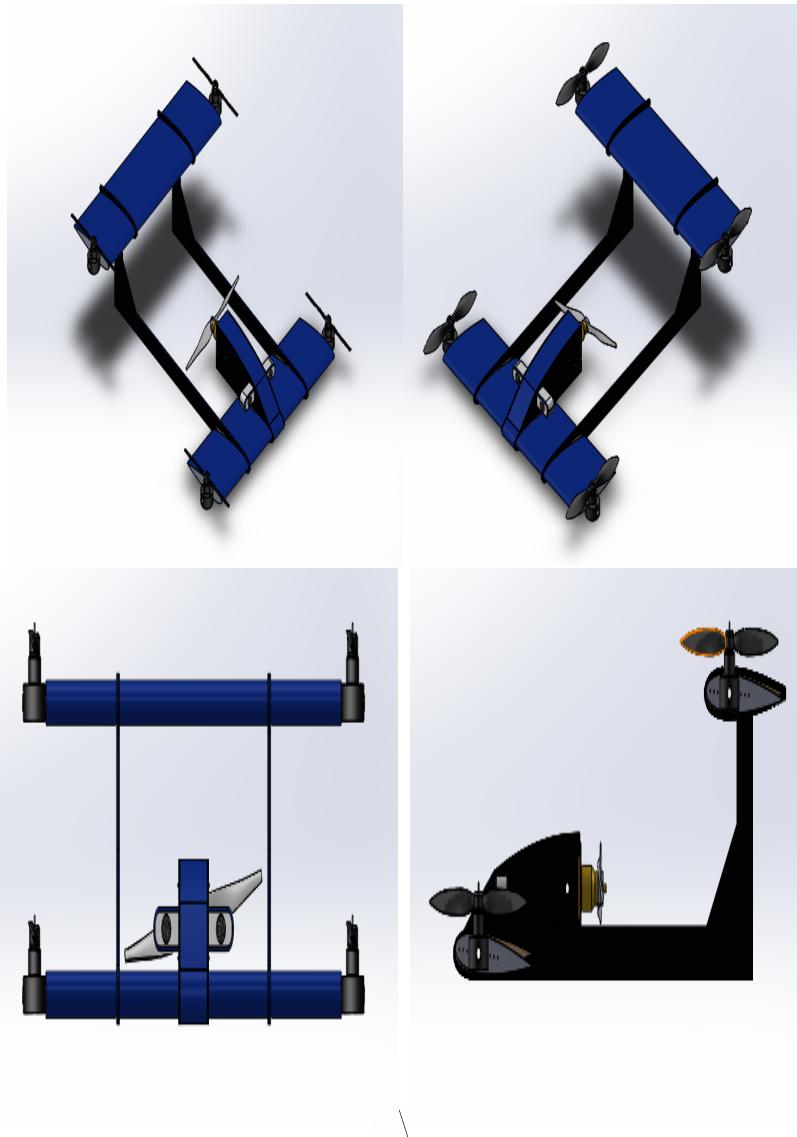


Figure 2.2.24: 3rd mechanical design iteration

- Fourth iteration

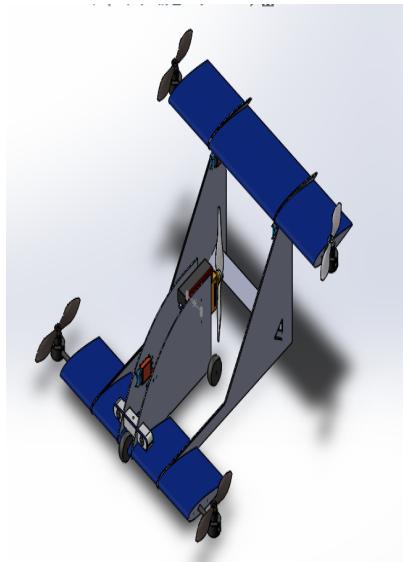


Figure 2.2.25: 4th mechanical design iteration

- Fifth iteration

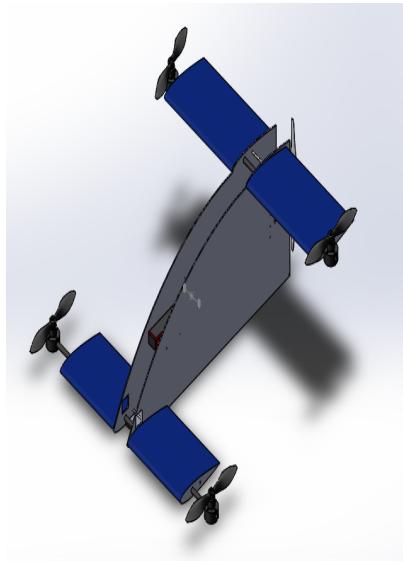


Figure 2.2.26: 5th mechanical design iteration

- Sixth iteration

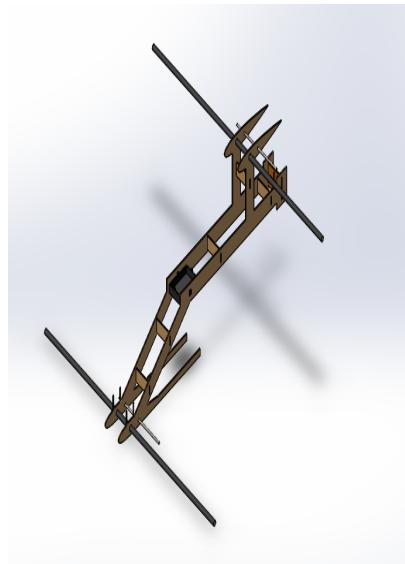


Figure 2.2.27: 6th mechanical design iteration

- Seventh iteration

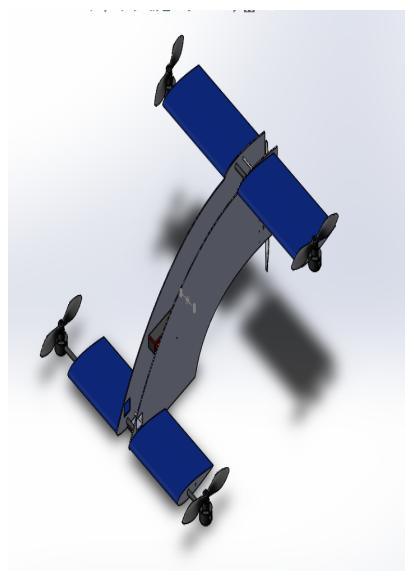


Figure 2.2.28: 7th mechanical design iteration

- Eighth iteration

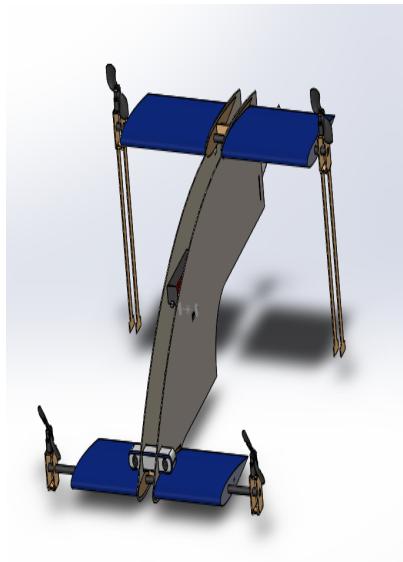


Figure 2.2.29: 8th mechanical design iteration

- Ninth iteration

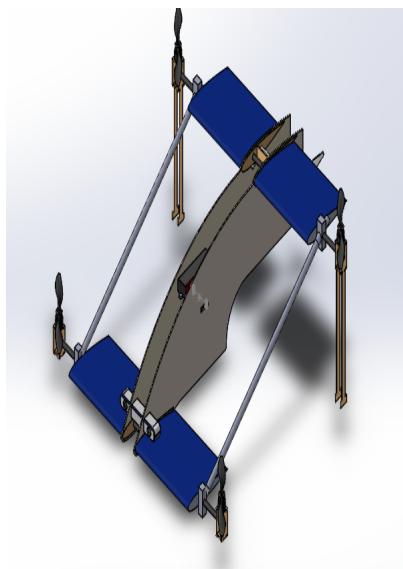


Figure 2.2.30: 9th mechanical design iteration

- Tenth iteration

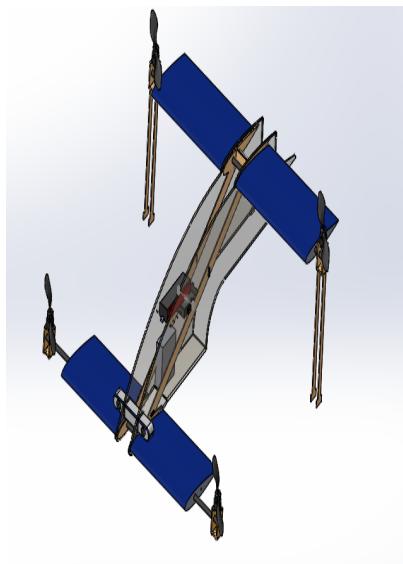


Figure 2.2.31: 10th mechanical design iteration

# Chapter 3

## Autopilot Design

### 3.1 Mathematical Model

The mathematical model for the flying taxi will be derived then the linearized model will be used to design the autopilot. The main purpose is to estimate initial gains for the PIXHAWK to make it easier to tune since our vehicle isn't conventional. The flying taxi is a quad-plane with 2 main phases, the 1st phase is the "quad-copter" phase in which the flying taxi pusher is turned off and the flying taxi acts as ordinary quad-copter during landing or takeoff which has the advantage of less distance for landing or takeoff and higher maneuverability especially for narrow places . In the 2nd phase after finishing takeoff and reaching the required altitude, the pusher is turned on and controlled via external controller code on the companion computer which sends the control action for the pusher to the PIXHAWK such that it achieves required cruise speed. It's assumed that during the 2nd phase (quad-plane phase) the flying taxi has constant heading and this is reasonable as it operates mainly between 2 waypoints at high altitude so we don't need to consider the quad-plane lateral dynamics. On the other hand, during the quad-copter phase, the flying taxi has low speed and low altitude so the position accuracy is important to be more accurate than the GPS readings. Luckily these operating conditions is suitable for generating visual odometry which can aid the GPS also the obstacle avoidance is important during this phase so it's turned on.

#### 3.1.1 Equations of Motion(Rigid-Body dynamics)

##### 3.1.1.1 Kinetics <sup>1</sup>

- Translational motion:  $\vec{F} = \frac{d}{dt}(m\vec{v}) = m(\ddot{\vec{v}} + \vec{\omega} \times \vec{v}) = \vec{F}_{Aero} + \vec{F}_{pusher} + \vec{F}_{motors} + \vec{g}$ 
  - This relation represents the absolute resultant force acting on the CG ,which is the mass multiplied by the absolute change in the velocity vector of the CG of the flying taxi  $\vec{v} = u\vec{i} + v\vec{j} + w\vec{k}$  but  $\vec{i}, \vec{j}, \vec{k}$  are unit vector in body axes directions which change direction with time so the absolute change is the sum of change in magnitudes  $\dot{\vec{v}} = \dot{u}\vec{i} + \dot{v}\vec{j} + \dot{w}\vec{k}$  and directions  $\vec{\omega} \times \vec{v}$  . Using body axes directions in translational motion is very beneficial as the inertial sensors gives its measurements in body axes so it's more suitable.
  - $\vec{F}_{Aero}$  can be ignored because during multicopter phase, the flying taxi encounters negligible aerodynamic force due to low speed but can't be ignored during the quad-plane phase as the higher velocity of this phase increases the effect of the aerodynamic forces which generates drag forces and lift force which is the main advantage for quad-plane over quad-copter since the generated lift increases range and endurance significantly with respect to the quad-copter configuration. This shows the effectiveness of the quad-plane configuration in the flying taxi application.
  - $\vec{F}_{pusher} = \begin{bmatrix} F_{pusher} \\ 0 \\ 0 \end{bmatrix}$  required to overcome the drag force to achieve required cruise speed.

<sup>1</sup>the study of forces and moments on system in motion

- $\vec{F}_{motors} = \sum_{i=1}^4 [-\frac{1}{2}\rho AC(\omega_i R_{propeller})^2] \vec{k}$  where  $b_i$  is constant can be obtained for every motor relates the square of the angular velocity of the motor  $\omega_{prop,i}^2$  to the generated thrust and the summation of the thrust of each motor holds the total thrust  $\vec{F}_{motors}$

$$\vec{g} = C_{IB} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

where  $C_{BI}$  is the rotation matrix from inertia axes to body axes according to ZYX euler angles  $(\psi, \theta, \phi)$  will be derived later in the kinematics study

$$\therefore \vec{g} = mg \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\theta) \end{bmatrix}$$

- Rotational motion:  $\vec{M} = \frac{d}{dt}(I\vec{\omega}) = I\ddot{\omega} + \vec{\omega} \times I\vec{\omega} = \vec{M}_{motors}$

—

$$\begin{aligned} \vec{M}_{motors} &= \vec{M}_{thrust\ induced} + \vec{M}_{drag\ induced} \\ &= \begin{bmatrix} l(T_3 + T_2 - T_1 - T_4) \\ l(T_1 + T_3 - T_2 - T_4) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -d_1\omega_1^2 - d_2\omega_2^2 + d_3\omega_3^2 + d_4\omega_4^2 \end{bmatrix} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} \end{aligned}$$

where  $\|M_{drag\ induced,i}\| = d_i\omega_i^2$  and  $d_i$  relates the square of the motor angular velocity and the generated torque on the motor base due to the aerodynamic resistance to the moving propeller which propagates to the flying taxi body from each motor.

motor 1: CCW drag , motor 2: CW drag , motor 3: CCW drag , motor 4: CW drag

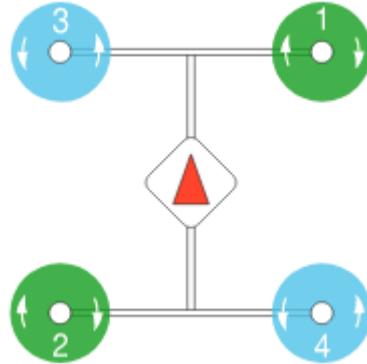


Figure 3.1.1: motor configuration

- also  $T_i = b_i\omega_i^2$   
It's better to design the controller using  $M_x, M_y, M_z$  instead of  $\omega_i^2$  to make the controller independent of the configuration so we need a control allocation (or mixing) law to map the controller output to a required angular velocity for each motor.
- This relation represents the absolute resultant moment around the CG , represented in the body axes directions which is very beneficial also in derivation because inertia is usually constant in body axes in contrast to fixed axes beside the previous reason stated in the translational motion.
- $I$  is the inertia tensor in body axes. For our flying taxi it's symmetric only about xz plane  

$$\therefore I = \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix}$$
- angular velocity of the flying taxi  $\vec{\omega} = p\vec{i} + q\vec{j} + r\vec{k}$

- absolute change of  $I\vec{\omega}$  is the sum of change in magnitudes  $I\dot{\vec{\omega}} = I(\dot{p}\vec{i} + \dot{q}\vec{j} + \dot{r}\vec{k})$  and directions  $\vec{\omega} \times I\vec{\omega}$

### 3.1.1.2 Kinematics <sup>2</sup>

Using ZYX euler angles  $(\psi, \theta, \phi)$ , the transformation matrix which can be used to transform vectors from inertial axes to body axes is:

$$C_{IB} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\cos(\phi)\sin(\psi) + \cos(\psi)\sin(\phi)\sin(\theta) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\psi)\sin(\theta) & \cos(\theta)\sin(\phi) \\ \sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta) & -\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\psi)\sin(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

$$\therefore \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = C_{BI} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad \text{where } C_{BI} = C_{IB}^T \text{ is the transform from body to inertial axes.}$$

As ZYX euler angles represents rotation from :

1. body axes around Z to intermediate axes(1) with angle  $\psi$
2. intermediate axes(1) around y1 to intermediate axes(2) with angle  $\theta$
3. intermediate axes(2) around x2 to body axes with angle  $\phi$

so  $\dot{\psi}$  is in Z direction ,  $\dot{\theta}$  is in y1 direction , and  $\dot{\phi}$  is in x2 direction

$$\therefore \vec{\omega} = p\vec{i} + q\vec{j} + r\vec{k} = \dot{\psi}\vec{K}_{inertial} + \dot{\theta}\vec{j}_1 + \dot{\phi}\vec{i}_2$$

$$\therefore \vec{K}_{inertial} = [3rd\ row\ of\ C_{BI}]. \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = [3rd\ row\ of\ C_{BI}]^T = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\theta) \end{bmatrix}$$

$$\vec{j}_1 = \vec{j}_2 = [2nd\ row\ R_x(\phi)]^T = \begin{bmatrix} 0 \\ \cos(\phi) \\ -\sin(\phi) \end{bmatrix}$$

$$\vec{i}_2 = \vec{i}$$

$$\therefore \vec{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \dot{\psi} \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\theta) \end{bmatrix} + \dot{\theta} \begin{bmatrix} 0 \\ \cos(\phi) \\ -\sin(\phi) \end{bmatrix} + \dot{\phi} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\therefore \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \cos(\theta)\sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$$\therefore \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

### 3.1.1.3 Resultant Nonlinear model

from the previous dynamics (kinetics and kinematics) study, we get the nonlinear system which is 12 equations in 12 unknowns (states) .

states: u , v , w , p ,q , r , X , Y , Z ,  $\phi$  ,  $\theta$  ,  $\psi$   
system:

<sup>2</sup>the study of motion without regard to forces or moments

$$\ddot{u} = -g\sin\theta - qw + rv + \frac{F_{pusher}}{m} + \frac{X_{aero}}{m} \quad (3.1.1)$$

$$\dot{v} = g\cos\theta\sin\phi - ru + pw \quad (3.1.2)$$

$$\dot{w} = g\cos\theta\cos\phi - pv + qu - \frac{T_{allmotors}}{m} + \frac{Z_{aero}}{m} \quad (3.1.3)$$

$$\ddot{p} = \frac{I_x I_{xz} pq - I_{xz}^2 qr - I_{xz} I_y pq + I_{xz} I_z pq + I_y I_z qr - I_z^2 qr + I_{xz} M_z + I_z M_x}{I_x I_z - I_{xz}^2} \quad (3.1.4)$$

$$\dot{q} = \frac{M_y - pr(I_x - I_z) - I_{xz}(p^2 - r^2)}{I_y} + \frac{M_{aero}}{I_y} \quad (3.1.5)$$

$$\dot{r} = \frac{I_x^2 pq - I_x I_{xz} qr - I_x I_y pq + I_{xz}^2 pq + I_{xz} I_y qr - I_{xz} I_z qr + I_x M_z + I_{xz} M_x}{I_x I_z - I_{xz}^2} \quad (3.1.6)$$

$$\dot{\phi} = p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \quad (3.1.7)$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \quad (3.1.8)$$

$$\dot{\psi} = q\sin\phi\sec\theta + r\cos\phi\sec\theta \quad (3.1.9)$$

$$\dot{X} = [\cos(\theta)\cos(\psi)]u + [\sin(\phi)\cos(\psi) - \cos(\phi)\sin(\psi)]v + [\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi)]w \quad (3.1.10)$$

$$\dot{Y} = [\cos(\theta)\sin(\psi)]u + [\sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi)]v + [\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi)]w \quad (3.1.11)$$

$$\dot{Z} = [-\sin(\theta)]u + [\sin(\phi)\cos(\theta)]v + [\cos(\phi)\cos(\theta)]w \quad (3.1.12)$$

### 3.1.2 Linearization

We linearize the dynamics of the flying taxi around equilibrium point at steady flight where :

all rates = 0 ,  $\phi_o = \theta_o = 0$ ,  $p = q = r = 0$  ,  $\alpha = 0$

$$\therefore \dot{u} = -g\theta + \frac{\Delta X_{pusher}}{m} + \frac{X_u^{aero}}{m}\Delta u + \frac{X_w^{aero}}{m}\Delta w \quad (3.1.13)$$

$$\dot{v} = g\phi - u_o r \quad (3.1.14)$$

$$\dot{w} = u_o q - \frac{\Delta T_{allmotors}}{m} + \frac{Z_u^{aero}}{m}\Delta u + \frac{Z_w^{aero}}{m}\Delta w + \frac{Z_q^{aero}}{m}\Delta q \quad (3.1.15)$$

$$\dot{p} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} + \frac{I_{xz} M_z}{I_x I_z - I_{xz}^2} \quad (3.1.16)$$

$$\dot{q} = \frac{M_y}{I_y} + \frac{M_u^{aero}}{I_y}\Delta u + \frac{M_w^{aero}}{I_y}\Delta w + \frac{M_q^{aero}}{I_y}\Delta q \quad (3.1.17)$$

$$\dot{r} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} + \frac{I_{zx} M_x}{I_x I_z - I_{xz}^2} \quad (3.1.18)$$

$$\dot{\phi} = p \quad (3.1.19)$$

$$\dot{\theta} = q \quad (3.1.20)$$

$$\dot{\psi} = r \quad (3.1.21)$$

$$\dot{Z} = w \quad (3.1.22)$$

The position linearization isn't good approximation since changes in angles especially the heading angle can't be assumed as small variations.

### 3.1.3 Parameters

from Solidworks model we can get initial estimates for the flying taxi parameters needed to design the controller in the next step:

m	1.9	kg
Ix	0.12	kg.m <sup>2</sup>
Iy	0.16	kg.m <sup>2</sup>
Iz	0.23	kg.m <sup>2</sup>
Ixz	0.05	kg.m <sup>2</sup>

## 3.2 Quad-copter phase Autopilot

The modified linearized model for the quad-copter phase:

$$\therefore \dot{\Delta u} = -g\theta \quad (3.2.1)$$

$$\dot{v} = g\phi \quad (3.2.2)$$

$$\dot{w} = -\frac{\Delta T_{allmotors}}{m} \quad (3.2.3)$$

$$\dot{p} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} + \frac{I_{xz} M_z}{I_x I_z - I_{xz}^2} \quad (3.2.4)$$

$$\dot{q} = \frac{M_y}{I_y} \quad (3.2.5)$$

$$\dot{r} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} + \frac{I_{zx} M_x}{I_x I_z - I_{xz}^2} \quad (3.2.6)$$

$$\dot{\phi} = p \quad (3.2.7)$$

$$\dot{\theta} = q \quad (3.2.8)$$

$$\dot{\psi} = r \quad (3.2.9)$$

$$\dot{Z} = w \quad (3.2.10)$$

### 3.2.1 Autopilot design

Autopilot block diagram is designed as follows:

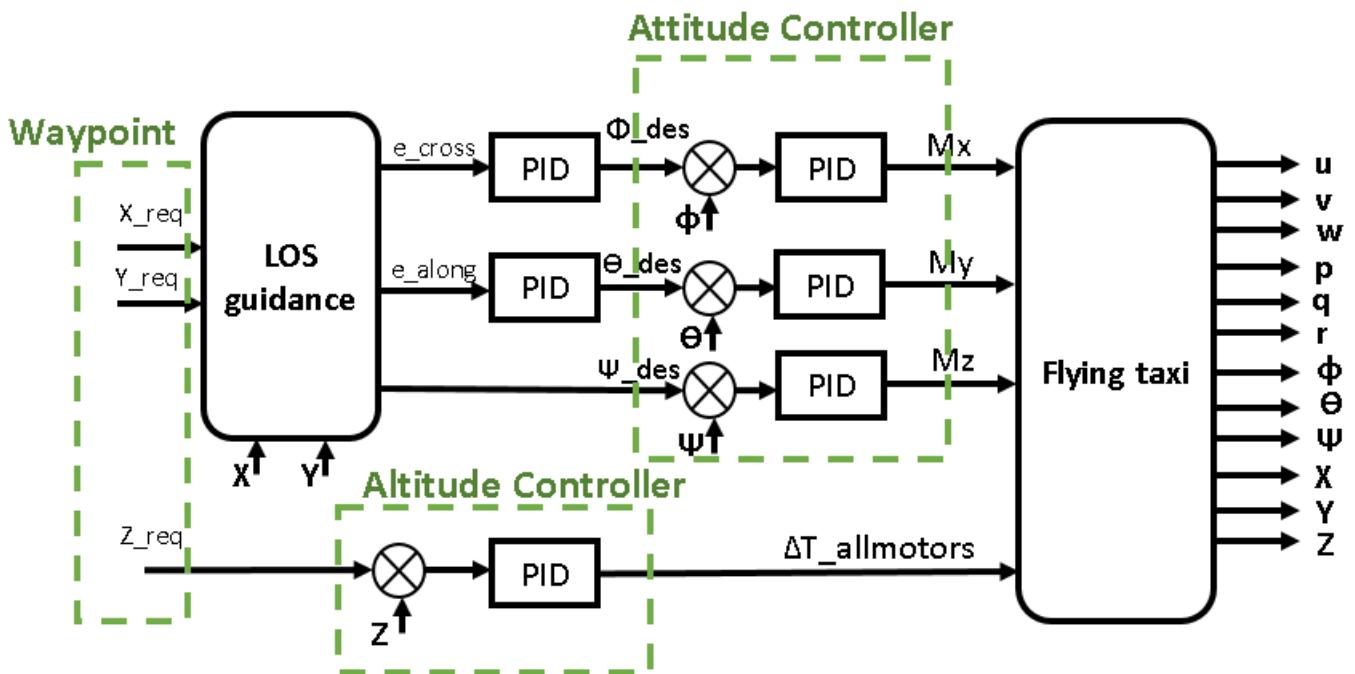


Figure 3.2.1: Autopilot design

#### 3.2.1.1 Attitude controller

##### **Roll Controller:**

As shown in parameters table,  $I_{xz}$  is very small compared to  $I_z$  so the  $M_z$  effect is less than  $M_x$  on the roll response. The accuracy of this assumption is tested by comparing Linear and Nonlinear response of the flying taxi and it turned out that it's very good assumption for our vehicle.

From Linearization section using equations( 3.2.4),(3.2.7):  $\therefore \ddot{\phi} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} \Rightarrow G_{phi} = \frac{\dot{\phi}}{M_x} = \frac{I_z}{(I_x I_z - I_{xz}^2) s^2}$   
ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

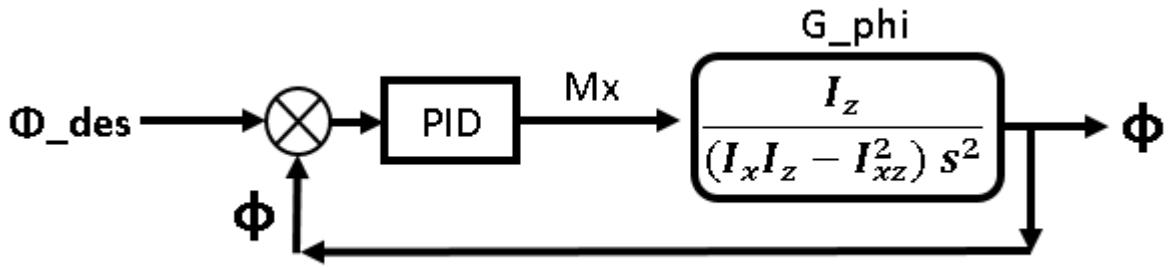


Figure 3.2.2: Roll Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.296$  ,  $D = 0$  ,  $I = 0.147533$  for inner loop and outer loop controller as  $P = 1.1681$

The roll step response and control action for linear and nonlinear models using the designed roll controller:

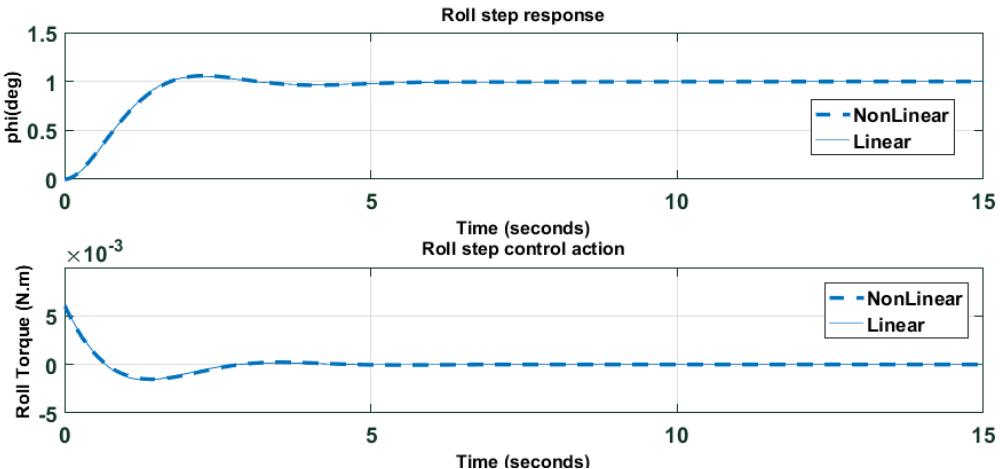


Figure 3.2.3: 1 deg desired Roll input

The response for 10 deg desired roll input and control action (to show the effect of larger input on the nonlinear model) :

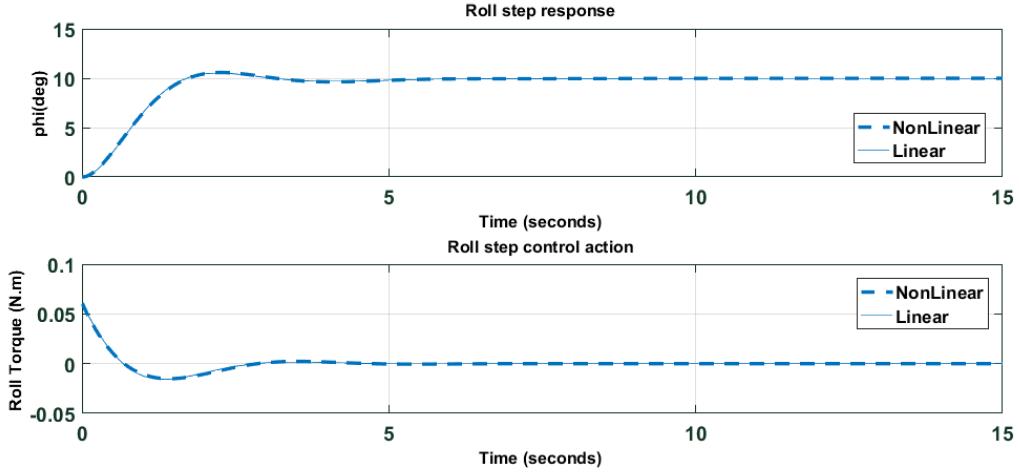


Figure 3.2.4: 10 deg desired Roll input

**Pitch Controller:**

From Linearization section using equations( 3.2.5),(3.2.8):  $\ddot{\theta} = \frac{My}{I_y} \Rightarrow G_\theta = \frac{\theta}{My} = \frac{1}{I_y s^2}$   
ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

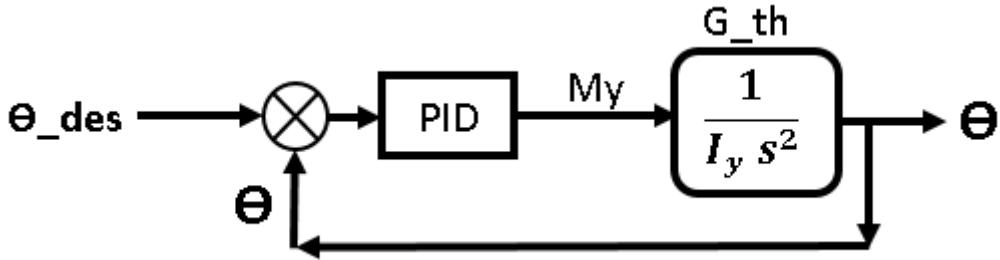


Figure 3.2.5: Pitch Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.32$  ,  $D = 0$  ,  $I = 0.156$  for inner loop  
,and outer loop controller as  $P = 0.89671$

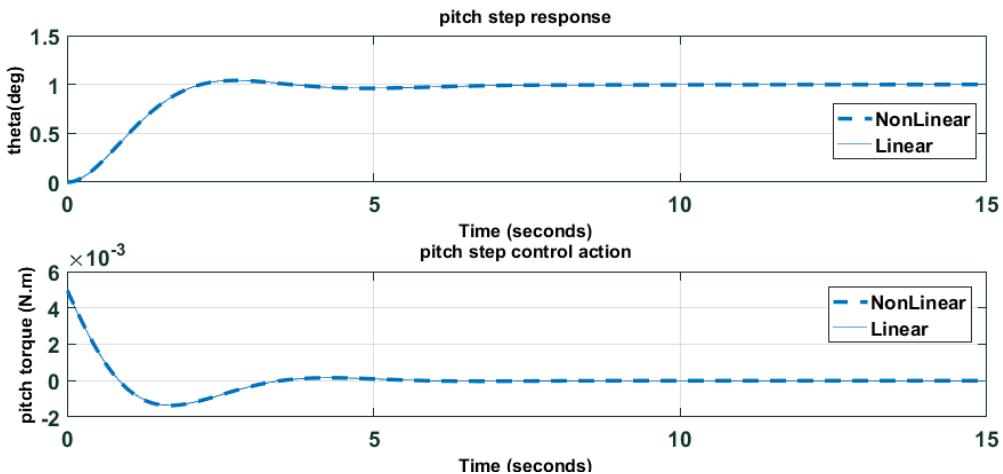


Figure 3.2.6: 1 deg desired pitch input

The response for 10 deg desired pitch input and control action (to show the effect of larger input on

the nonlinear model) :

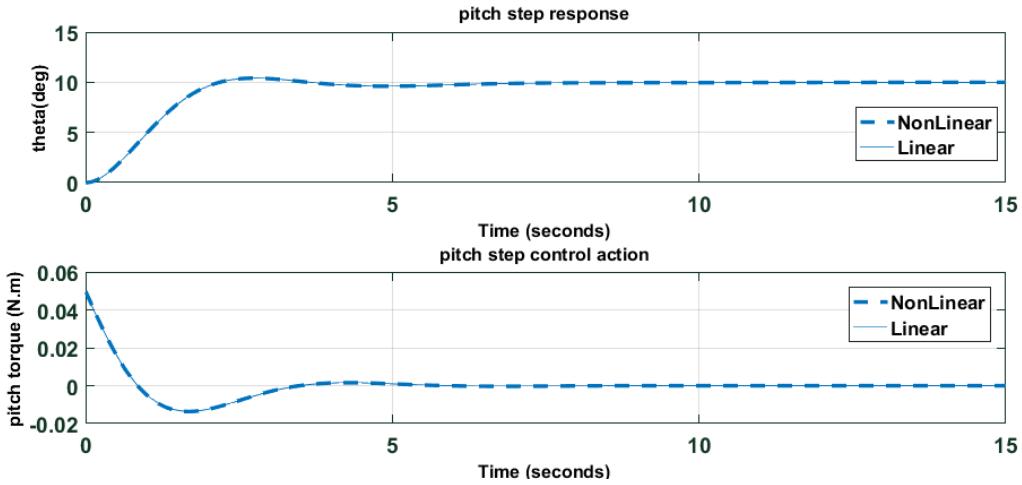


Figure 3.2.7: 10 deg desired pitch input

#### Yaw Controller:

As shown in parameters table,  $I_{xz}$  is also very small compared to  $I_x$  so the  $M_x$  effect is less than  $M_z$  on the roll response.. The accuracy of this assumption is tested by comparing Linear and Nonlinear response of the flying taxi and it turned out that it's very good assumption for our vehicle.

From Linearization section using equations( 3.2.6),(3.2.9):  $\therefore \ddot{\psi} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} \Rightarrow G_\psi = \frac{\dot{\psi}}{M_z} = \frac{I_x}{(I_x I_z - I_{xz}^2) s^2}$  ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

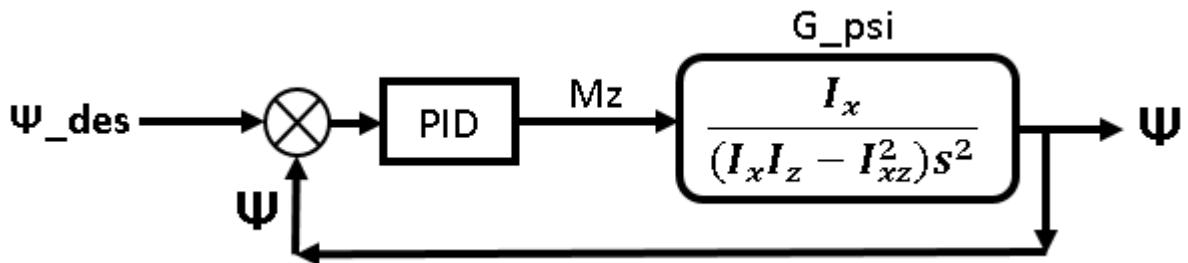


Figure 3.2.8: Yaw Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.72$  ,  $D = 0$  ,  $I = 0.5634$  for inner loop ,and outer loop controller as  $P = 1.8681$

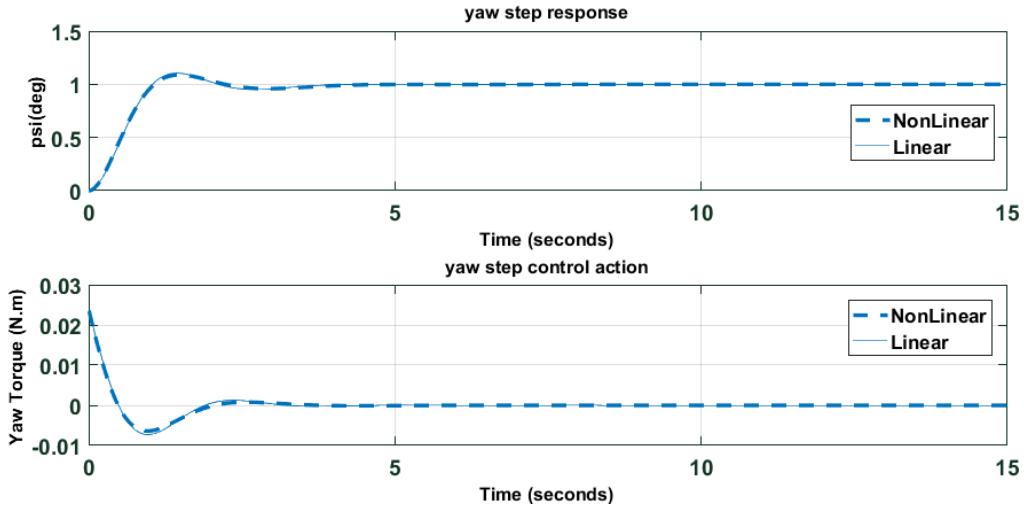


Figure 3.2.9: Yaw Step response and control action

The response for 10deg desired Yaw input and control action (to show the effect of larger input on the nonlinear model) :

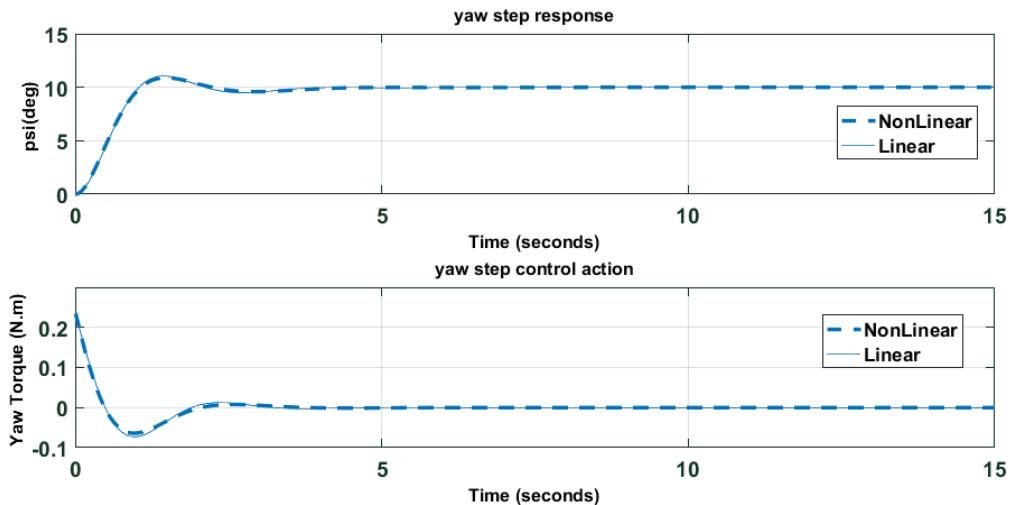


Figure 3.2.10: 10 deg desired yaw input

#### Roll,Pitch and Yaw Controllers together in action:

As we see from the nonlinear model all attitude states affects each others (Coupled) so we need to verify that there disturbance on each other won't lead to bad performance or even instability. As shown the linear model is decoupled which isn't the case in reality. The disturbance on pitch from changing other states is handled well by the pitch controller which enforce the yaw angle back to zero. There is small effects on the roll and yaw responses and control actions.

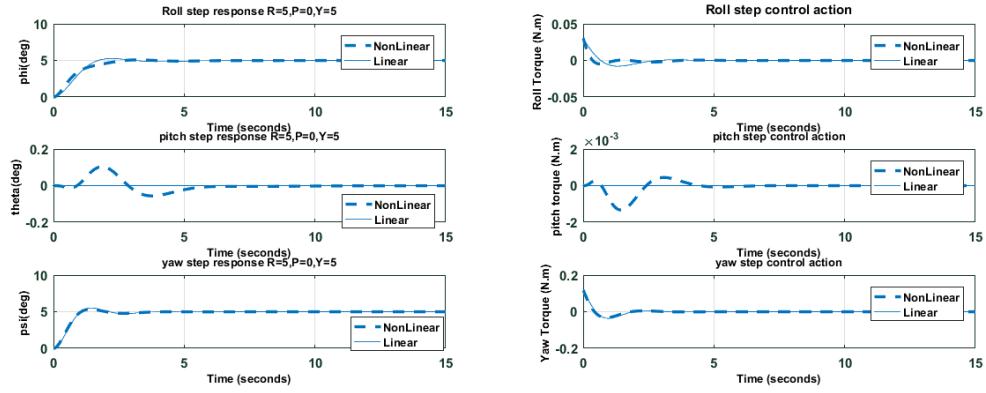


Figure 3.2.11: desired roll = 5 deg , desired pitch = 0 , desired yaw = 5

### 3.2.1.2 Altitude Controller

From Linearization section using equations( 3.1.15),(3.1.22):

$$\therefore \ddot{Z} = -\frac{\Delta T_{allmotors}}{m} \Rightarrow G_Z = \frac{Z}{\Delta T_{allmotors}} = \frac{1}{m s^2}$$

ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

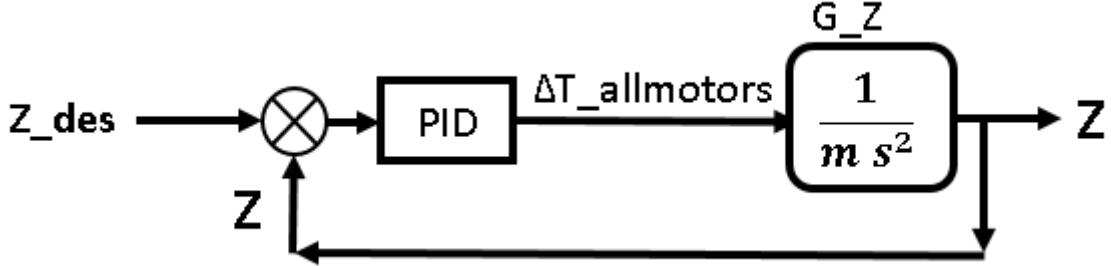


Figure 3.2.12: Altitude Controller

Using Matlab SISO tool to obtain robust controller :  $P = 3.753$  ,  $D = 0$  ,  $I = 1.853$  for inner loop ,and outer loop controller as  $P = .86913$

The altitude step response ( $Z = -1$  m) and control action for linear and nonlinear models using the designed yaw controller:

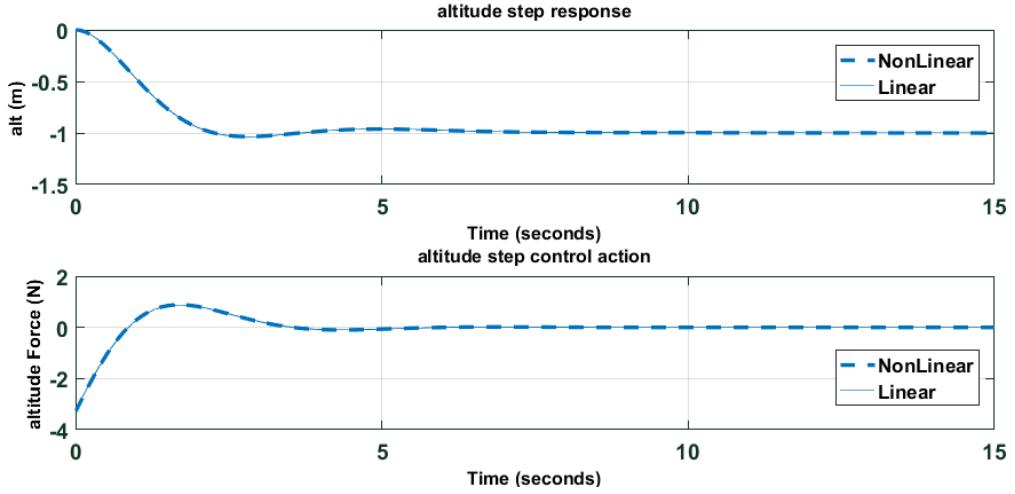


Figure 3.2.13: Altitude Step response and control action

The response for -10 m ( $Z = 10$  m) desired altitude input and control action (to show the effect of larger input on the nonlinear model) :

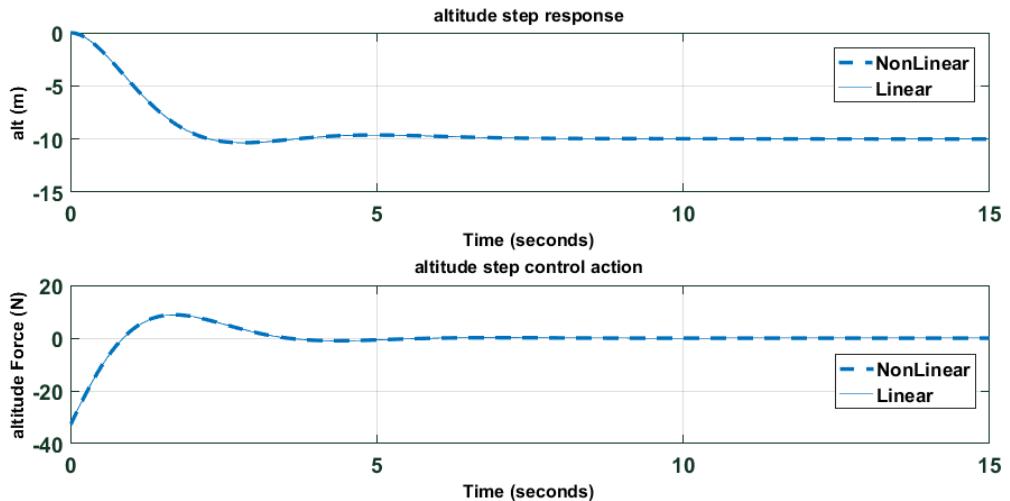


Figure 3.2.14: -4m altitude input

Note : Negative  $\Delta T_{allmotors}$  doesn't mean reversed motors thrust but means that the thrust will decrease below nominal thrust.

### 3.2.1.3 Attitude & Altitude controllers together in action

The effect of changing both attitude and altitude is studied because as appearing in the nonlinear model the attitude is affecting the altitude so we should verify that the altitude controller can handle this effect which considered relative to the altitude controller as external disturbance

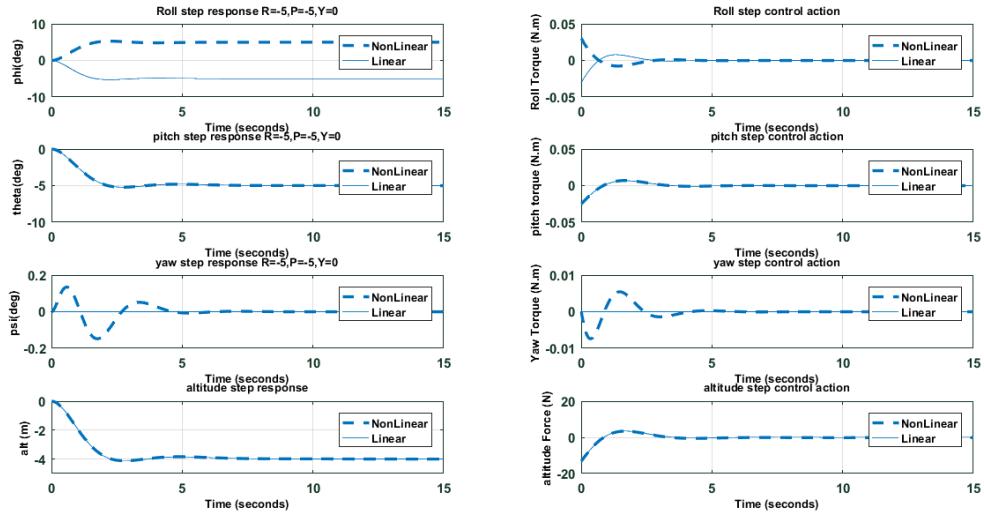


Figure 3.2.15: desired roll = -5 deg , desired pitch = -5 deg , desired altitude = -4 m ( Z = 4 m)

### 3.2.1.4 Line-of-sight (LOS) Guidance

LOS guidance is a method used to guide a vehicle between 2 waypoints using 2 types of errors : along-track error and cross-track error by calculating both of them we can then choose pitching to eliminate along-track error and rolling to eliminate cross-track error. From the previous waypoint and current one we can get the desired yaw angle. LOS can be more complicated especially for fixed wing as we can't control rolling without changing yaw angle which isn't the case for us during multicopter flight phase of the flying taxi.

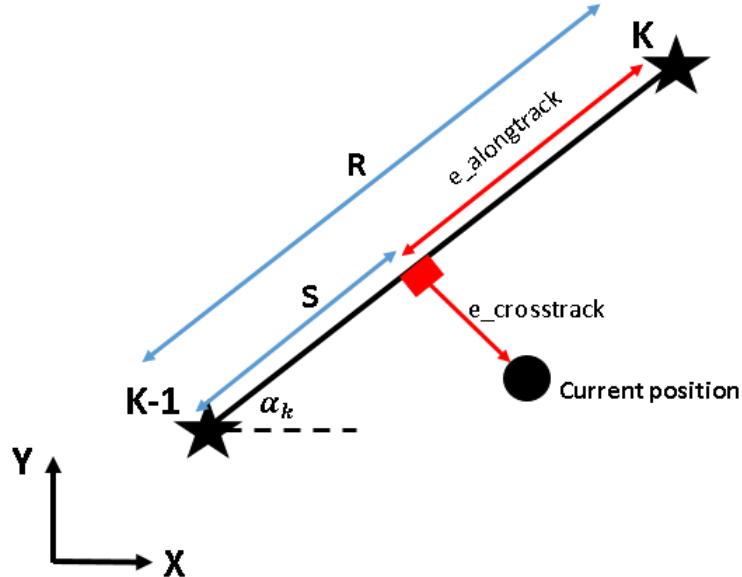


Figure 3.2.16: LOS

$$\alpha_k = \tan^{-1} \left( \frac{Y_{des}^k - Y_{des}^{k-1}}{X_{des}^k - X_{des}^{k-1}} \right) \quad (3.2.11)$$

$$\psi_{des} = \alpha_k \quad (3.2.12)$$

$$e_{cross-track} = -(X - X_{des}^{k-1})\sin(\alpha_k) + (Y - Y_{des}^{k-1})\cos(\alpha_k) \quad (3.2.13)$$

$$s = (X - X_{des}^{k-1})\cos(\alpha_k) + (Y - Y_{des}^{k-1})\sin(\alpha_k) \quad (3.2.14)$$

$$R = \sqrt{(Y_{des}^k - Y_{des}^{k-1})^2 + (X_{des}^k - X_{des}^{k-1})^2} \quad (3.2.15)$$

$$e_{along-track} = R - s \quad (3.2.16)$$

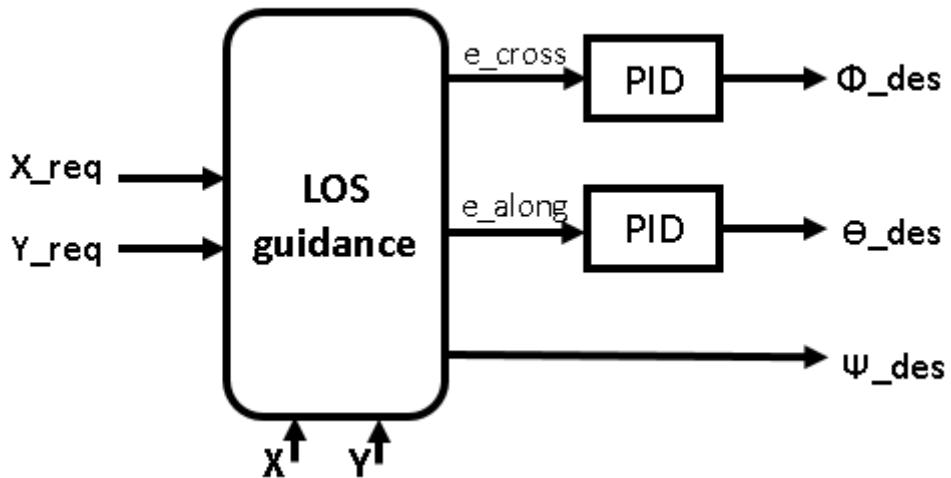


Figure 3.2.17: LOS block diagram

Our strategy to get PID values is using a linearized model to design a controller as initial estimate then using LabVIEW simulation of the nonlinear model we can tune the controller for better performance.

The attitude dynamics is much faster than position dynamics. so we can assume that heading angle is settled so the along-track error is in the x-body direction and the cross-track error is in the y-body direction.

From equ. (3.2.16) :  $e_{along-track} = R - s \Rightarrow \dot{e}_{along-track} = -u \Rightarrow e_{along-track} = \frac{-u}{s}$

from equ.(3.2.1):  $\Delta u = \frac{-g\theta}{s} \Rightarrow e_{along-track} = \frac{g\theta}{s^2}$

It's clear that the linearized model only accounts for g component effect but the  $\Delta T_{allmotors}$  component doesn't appear so tuning the nonlinear model is important to be done and it's expected that the deviation between linear and nonlinear is significant.

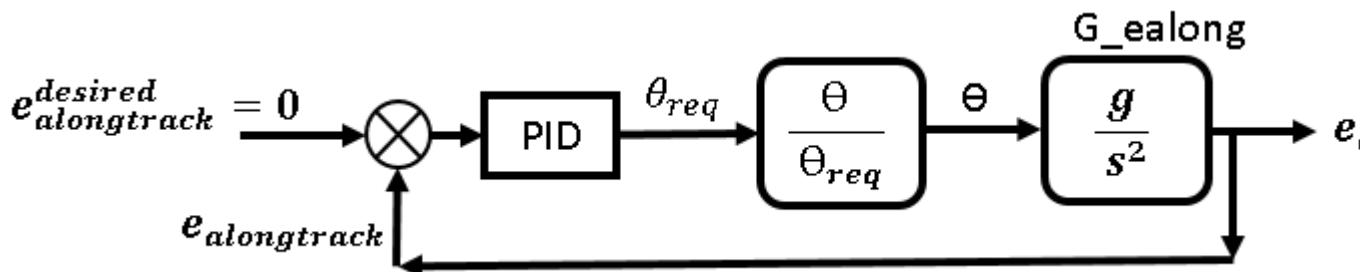


Figure 3.2.18: X guidance

Using SISO tool :  $P = -0.00063466$  ,  $D = -0.0159$  ,  $I = 0$

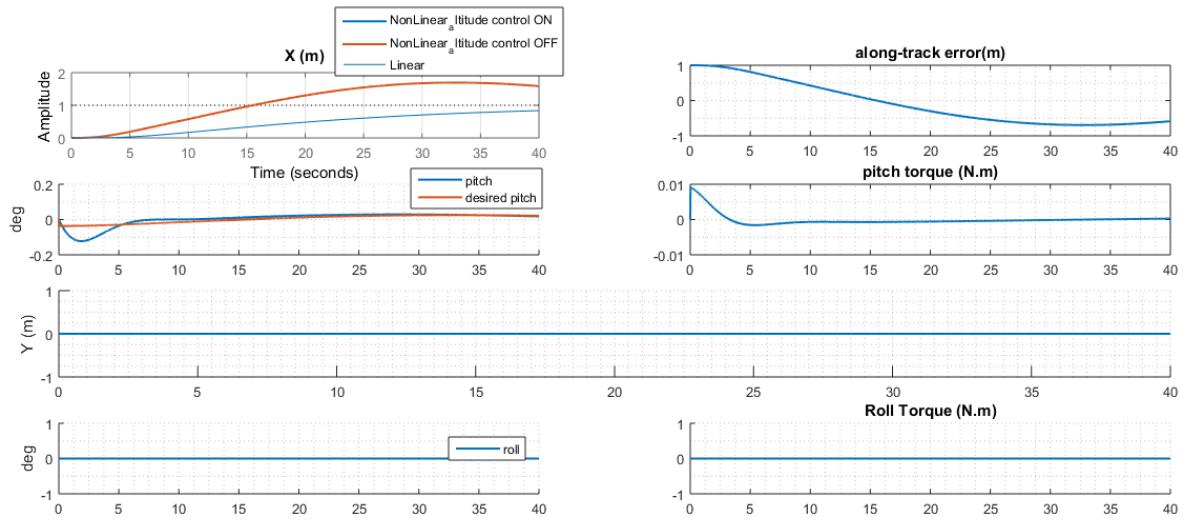


Figure 3.2.19: along-track control (trial : 1)

changing the gains in LabVIEW Simulation (Shown in the next section) by giving input way point ( $X=1$  ,  $Y=0$  ,  $Z=-1$ ):

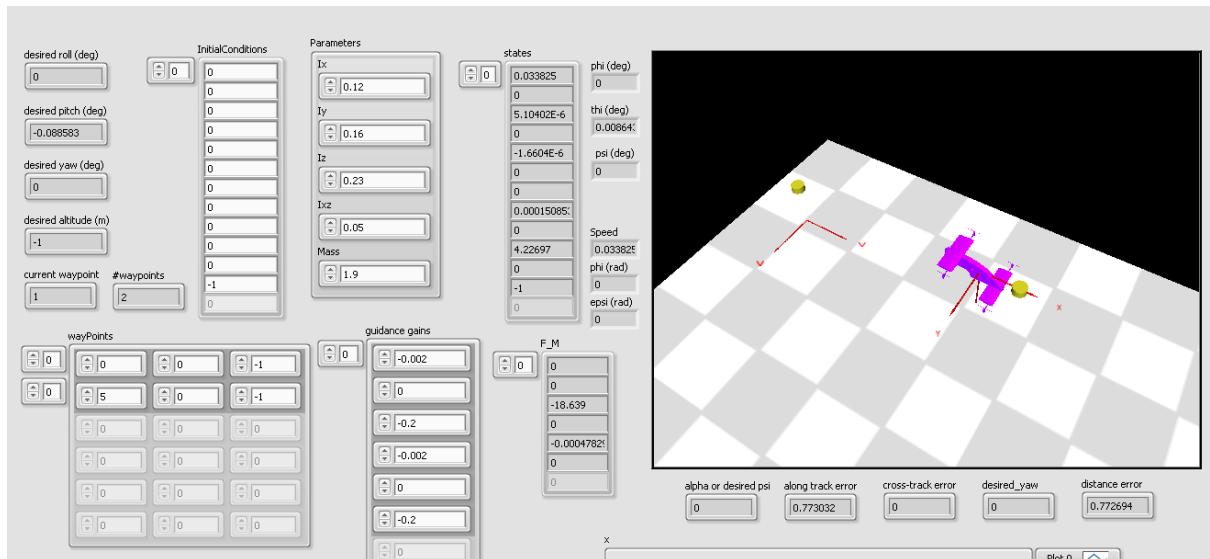


Figure 3.2.20: LabVIEW front panel

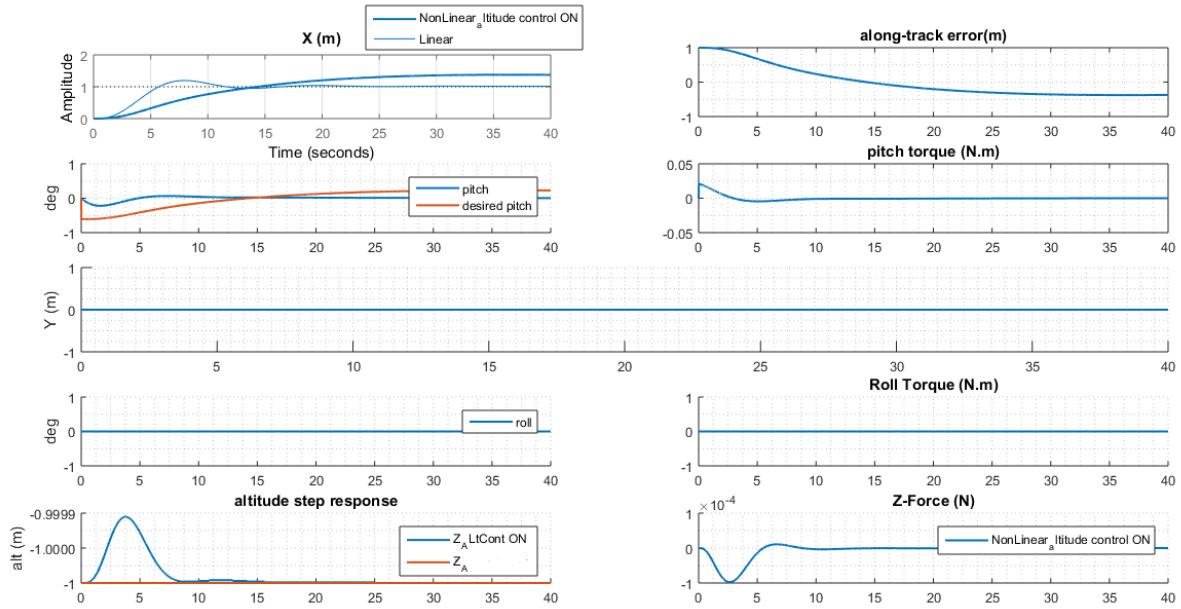


Figure 3.2.21: along-track control (trial : 2 increasing P) : high steady-state error

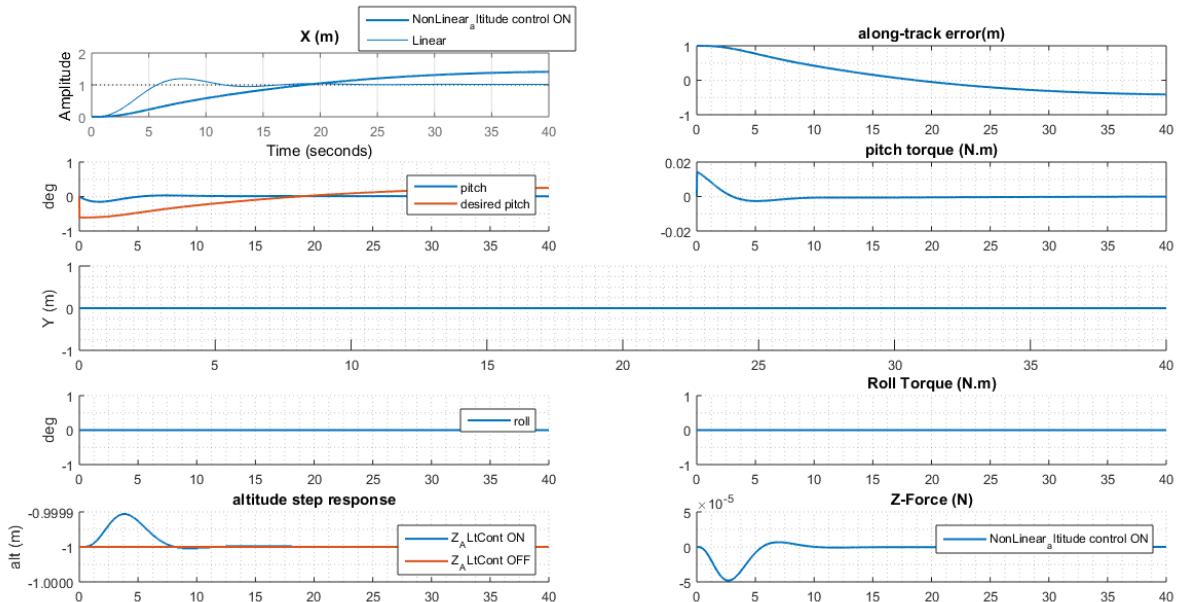


Figure 3.2.22: along-track control (trial : 2 decreasing D) : high steady-state error and the settling time is increased (as expected but we only wanted see the effect)

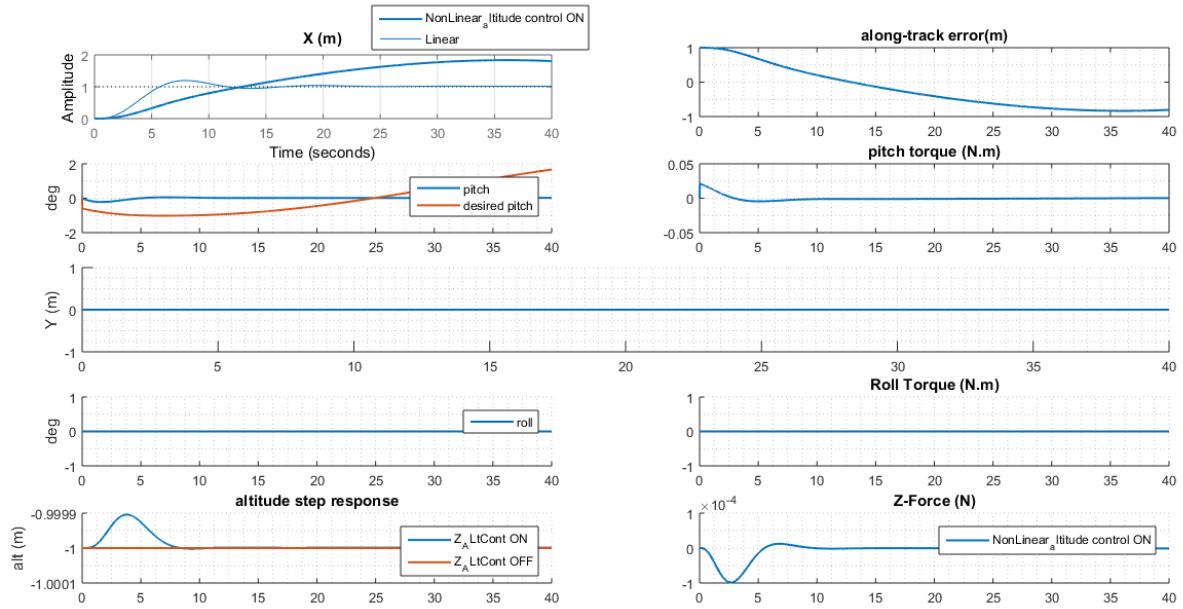


Figure 3.2.23: along-track control (trial : 3 adding small I term to eliminate steady-state error) : settling time is highly increased

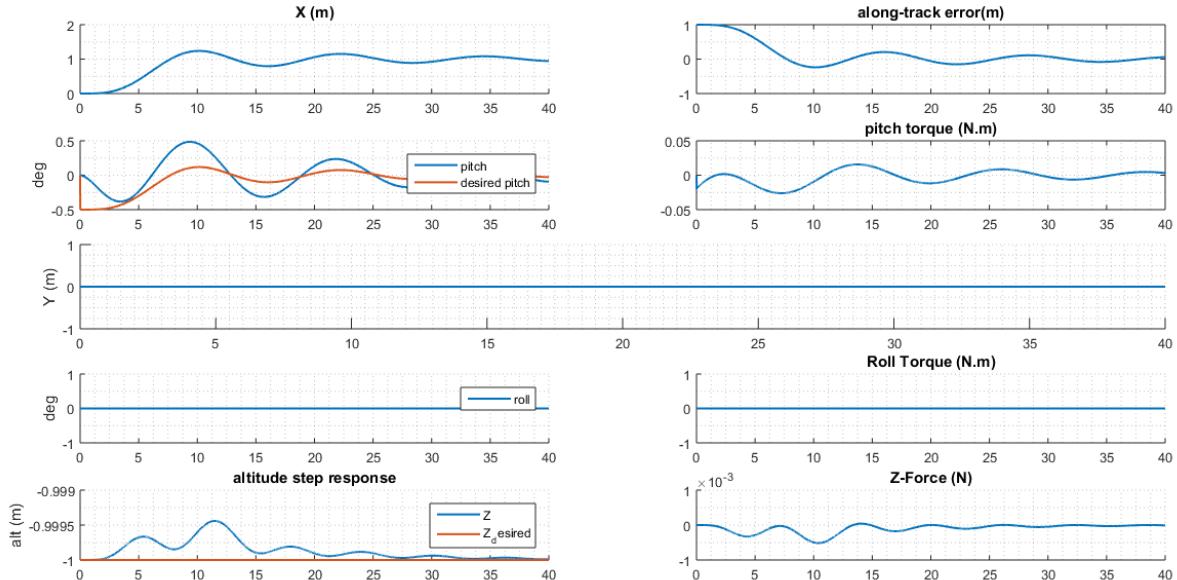


Figure 3.2.24: along-track error control (final trial) : increasing P and D

gains from linearized model :  $P = -0.00063466$  ,  $D = -0.0159$  ,  $I = 0$

new gains after tuning using nonlinear simulation :  $P = -0.002$  ,  $D = -0.2$  ,  $I = 0$   
in all previous results the altitude control is working as could be seen from Z-force response.

Similarly, For cross-track error we used the same procedure:  $\dot{v} = g\phi \Rightarrow \dot{e}_{cross-track} = v = \frac{g\phi}{s} \Rightarrow e_{cross-track} = \frac{g\phi}{s^2}$

As before it's clear that the linearized model only accounts for g component effect but the  $\Delta T_{allmotors}$  component doesn't appear

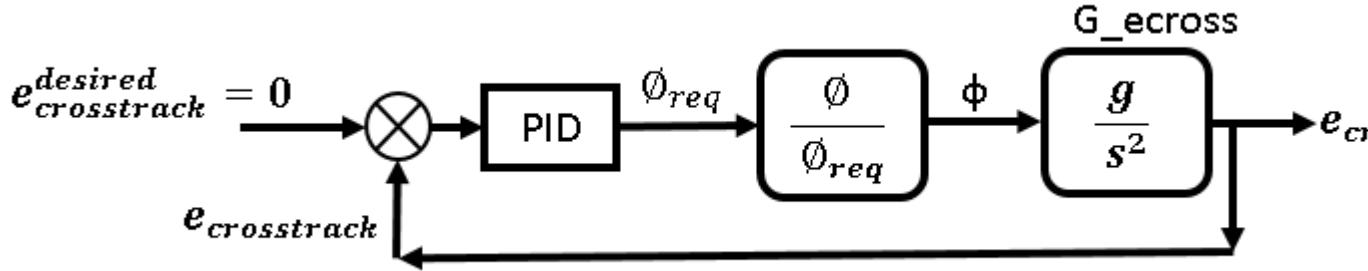


Figure 3.2.25: Y guidance

Using SISO tool :  $P = -0.00063473$  ,  $D = -0.0159$  ,  $I = 0$  approximately the same as linear X-guidance because the only difference is in the  $\frac{\phi}{\theta_{req}}$  instead of  $\frac{\theta}{\theta_{req}}$  which, from our designed attitude controller results, is fast compared to X , Y responses

So we used the previously tuned gains from nonlinear simulation:  $P = -0.002$  ,  $D = -0.2$  ,  $I = 0$

Now we run the nonlinear simulation by giving input waypoint(X=0 , Y=1 , Z=-1):

as seen from the figure below , the yaw angle is changed to -90 deg quickly so the cross-track became along-track error

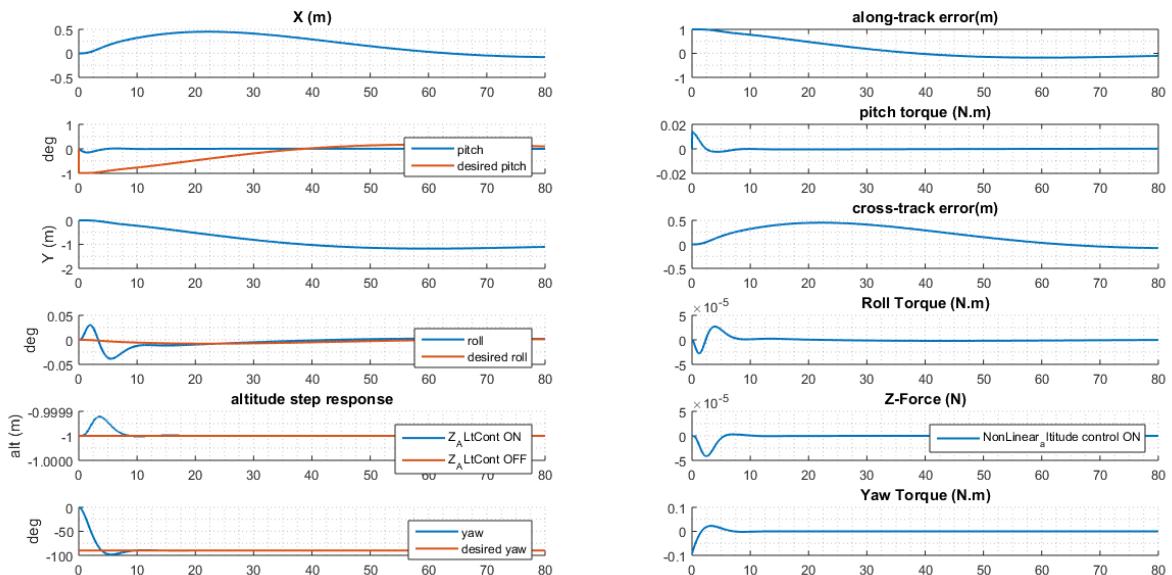
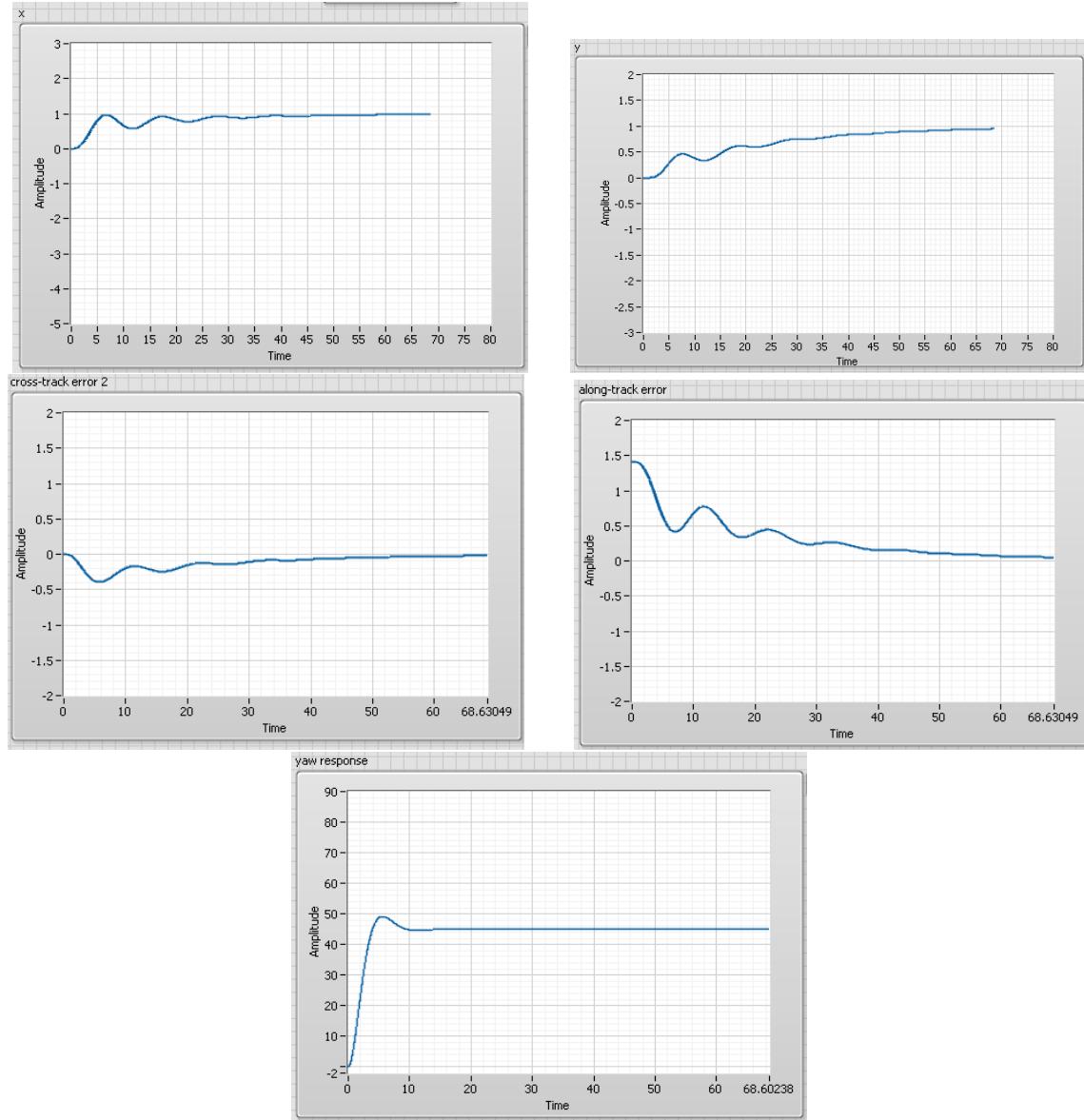


Figure 3.2.26: X,Y guidance waypoint(X=0 , Y=1 , Z=-1)

Intermediate case input waypoint (X=1 , Y=1 , Z=-1):

Figure 3.2.27: X,Y Guidance waypoint ( $X=1$  ,  $Y=1$  ,  $Z=-1$ )

### 3.2.2 Simulation using LabVIEW

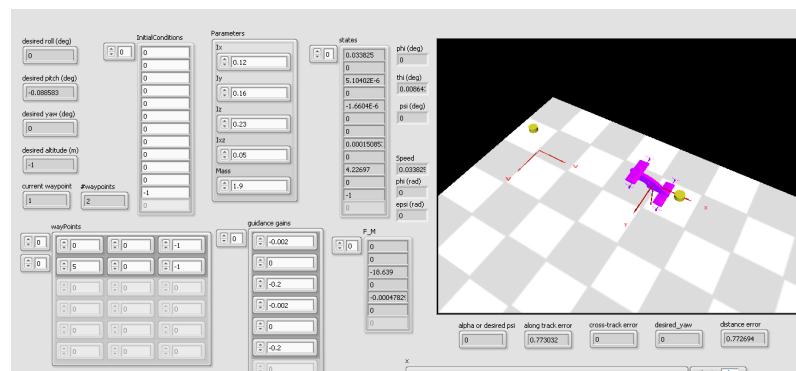


Figure 3.2.28: LabVIEW front panel

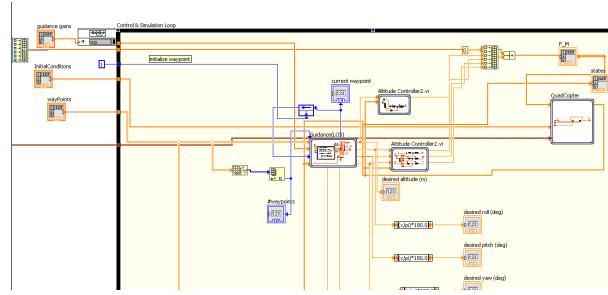


Figure 3.2.29: LabVIEW blockdiagram

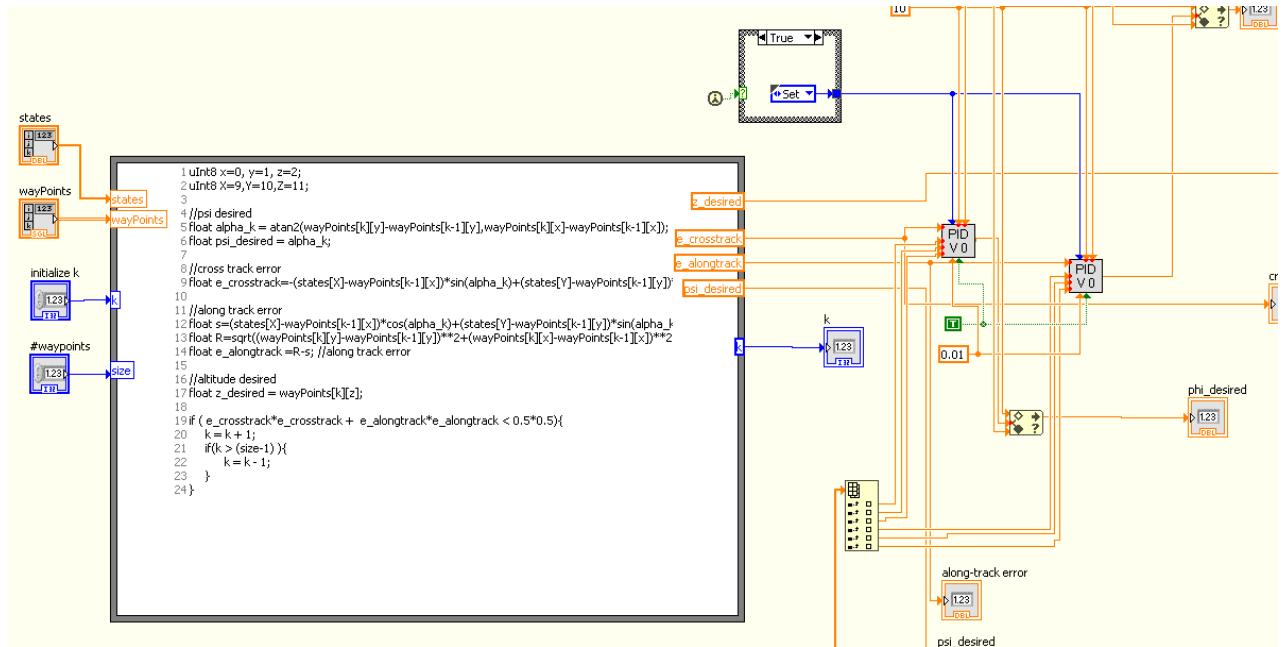


Figure 3.2.30: LabVIEW Guidance blockdiagram

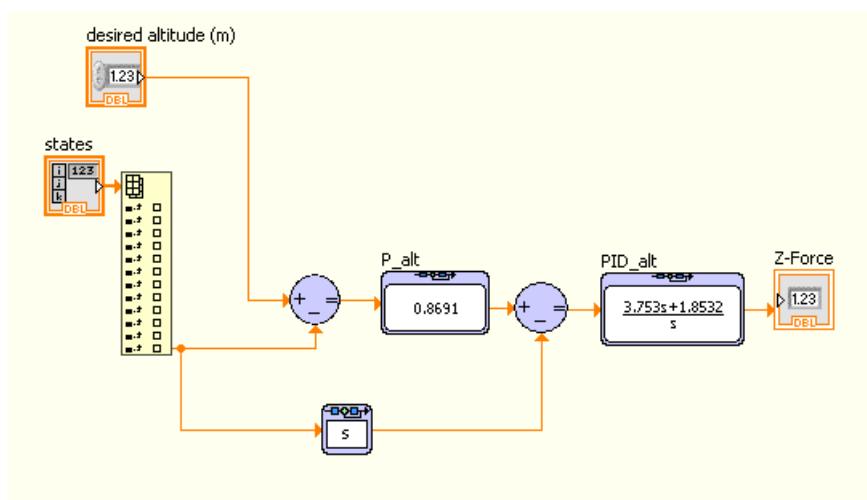


Figure 3.2.31: LabVIEW altitude BD

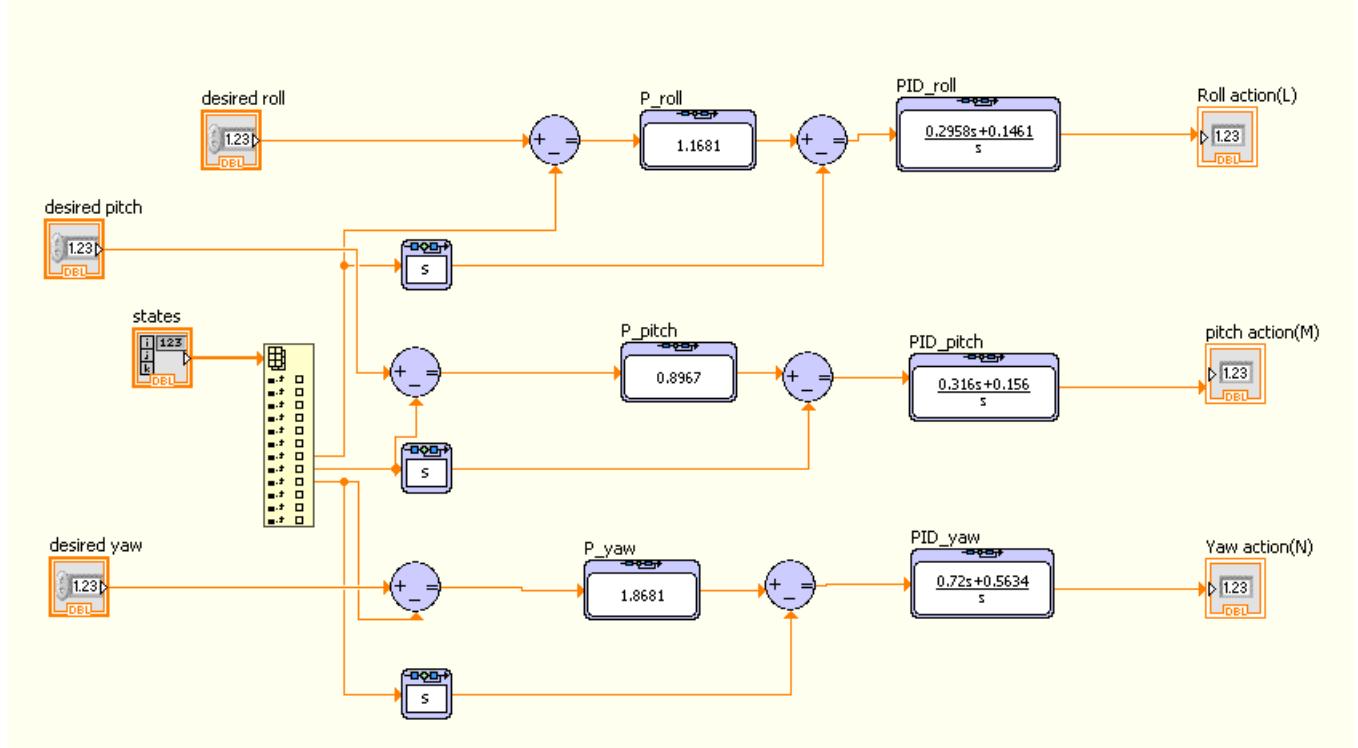


Figure 3.2.32: LabVIEW attitude controller BD

### 3.2.3 Commercial Autopilot

We used PIXHAWK as a commercial autopilot. PIXHAWK is an autopilot hardware for academic, hobby and developer communities and supports flight stacks software like PX4 and ArduPilot. We used PX4 software. PX4 software consists of 2 main parts: Middleware and Flightstack as follows:

PX4 Software	
Middleware	Flightstack
general robotics layer for any autonomous robot not only drones like rovers. It handles the internal communications between the different running programs using asynchronous message passing(uORB) and external communication using MAVLink or FastRTPS. It provides the hardware integration (GPS-IMU-....)	It contains the estimation and flight control system.

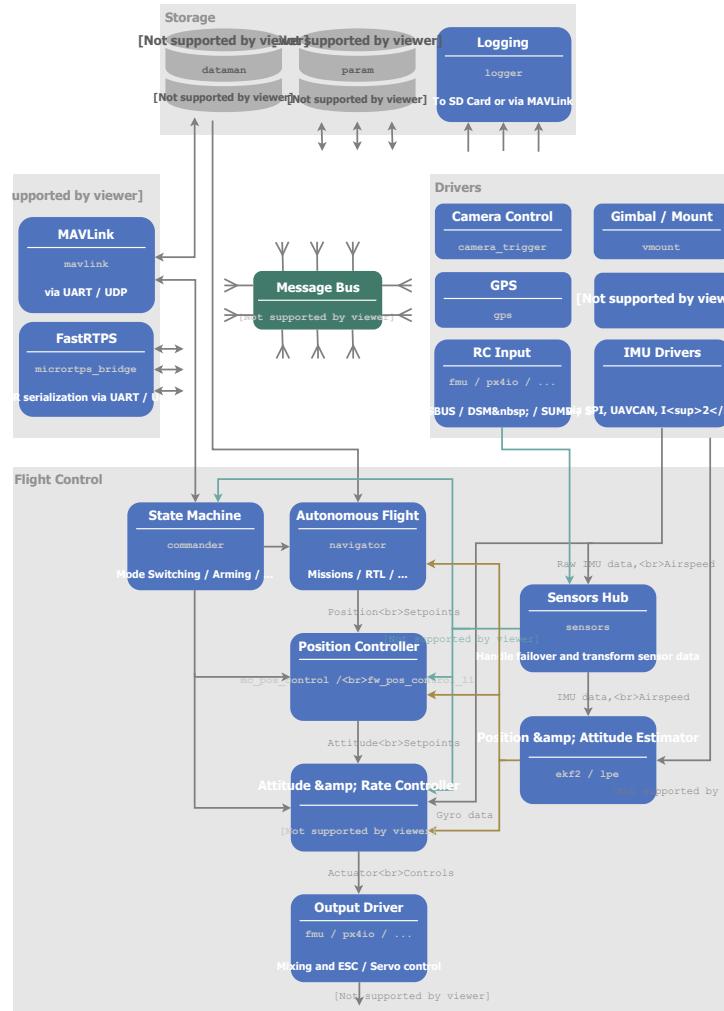


Figure 3.2.33: PX4 architecture from PX4 website

We are concerned more with the Flightstack so we will take a small dive to show the basics of each subsystem in the flightstack. The flightstack consists of 7 main subsystems: Commander - Navigator - Position Controller - Attitude & Rate Controller - Sensors Hub - Estimator - Output Driver

### 3.2.3.1 Commander

The commander is a state-machine architecture which can switch the flight modes based on specific conditions from the remote control or even from the mavlink (using companion computer). A simple example shows how the commander works: The commander has a main loop inside which check if there is an “Arming” signal coming from the remote control and if there is , it will call a function called “arming\_state\_transition()” and after checking if it’s valid to “Arm” the multicopter. the arm variable will be changed to the arming value and then published for other programs using the uORB message system which is built inside the PX4 middleware. When the output drivers programs read the arm variable, they will initiate the spinning of the motors. We will explain the “flight modes” later in this section.

### 3.2.3.2 Navigator

The navigator takes its **inputs** from database(store waypoints or mission),RC(Remote control) or the Commandar, and **outputs** position setpoints to be used by the position controller.

### 3.2.3.3 Position controller

inputs: Position setpoints & heading

outputs: **Thrust vector** which can be decomposed into attitude(angles) setpoints and thrust magnitude.

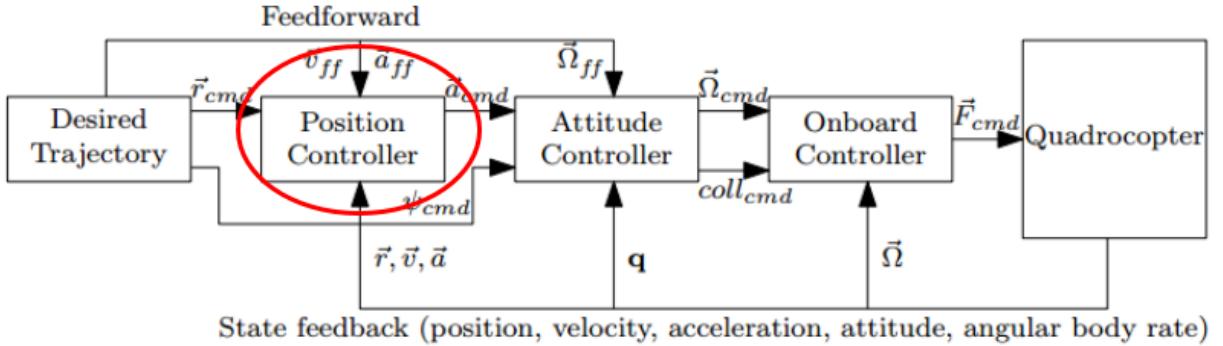


Figure 3.2.34: cascaded control architecture from “Nonlinear Quadrocopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello,2013”

The feedforward signal isn't used for multirotors.

The position controller is implemented such that it consists of 2 loops :

- 1st loop inputs: position setpoints , outputs: velocity setpoints , controller: P-controller

```
void PositionControl::positionController()
{
    // P-position controller
    const Vector3f vel_sp_position = (_pos_sp - _pos).emult(Vector3f(_param_mpc_xy_p.get(), _param_mpc_xy_p.get(),
        _param_mpc_z_p.get()));
    _vel_sp = vel_sp_position + _vel_sp;

    // Constrain horizontal velocity by prioritizing the velocity component along the
    // the desired position setpoint over the feed-forward term.
    const Vector2f vel_sp_xy = ControlMath::constrainXY(Vector2f(vel_sp_position),
        Vector2f(_vel_sp - vel_sp_position), _param_mpc_xy_vel_max.get());
    _vel_sp(0) = vel_sp_xy(0);
    _vel_sp(1) = vel_sp_xy(1);
    // Constrain velocity in z-direction.
    _vel_sp(2) = math::constrain(_vel_sp(2), -_constraints.speed_up, _constraints.speed_down);
}
```

- 2nd loop inputs: velocity setpoint , outputs: required thrust vector , controller: PID-controller

```
const Vector3f vel_err = _vel_sp - _vel;

// Consider thrust in D-direction.
float thrust_desired_D = _param_mpc_z_vel_p.get() * vel_err(2) + _param_mpc_z_vel_d.get() * _vel_dot(2) + _thr_int(2) - _param_mpc_thr_hover.get();

// PID-velocity controller for NE-direction.
Vector2f thrust_desired_NE;
thrust_desired_NE(0) = _param_mpc_xy_vel_p.get() * vel_err(0) + _param_mpc_xy_vel_d.get() * _vel_dot(0) + _thr_int(0);
thrust_desired_NE(1) = _param_mpc_xy_vel_p.get() * vel_err(1) + _param_mpc_xy_vel_d.get() * _vel_dot(1) + _thr_int(1);
```

But the D-term in the controller don't use the error rate as usual PDs , it uses the negative rate of the estimated plant velocity to avoid “the derivative kick” which causes high control action due to the discrete implementation of the PID.

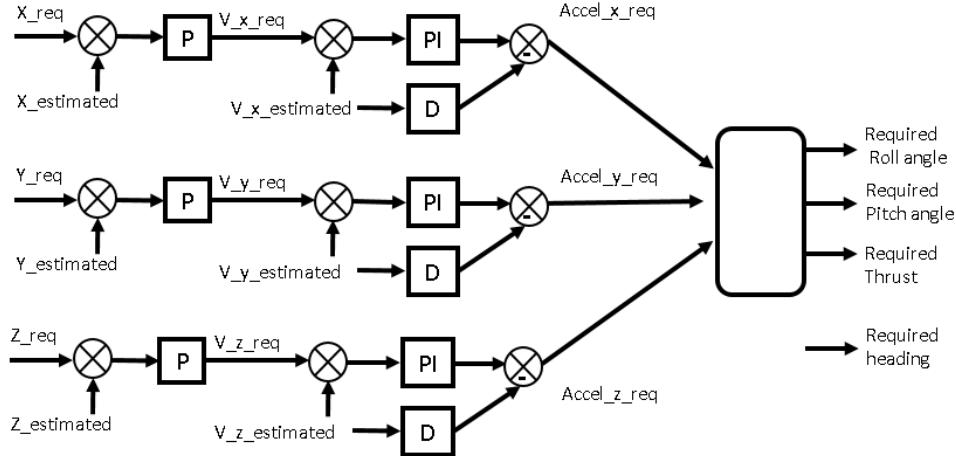


Figure 3.2.35: Position Controller

### 3.2.3.4 Attitude & Rate Controller

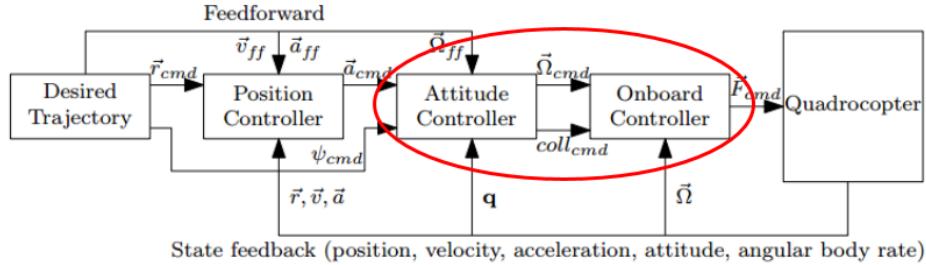


Figure 3.2.36: cascaded control architecture from “Nonlinear Quadrocopter Attitude Control Technical Report, Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello, 2013”

inputs: attitude setpoints(Roll-Pitch-Yaw)

outputs: **Required Forces and Moments**

The Attitude controller consists also of 2 loops :

- 1st loop      inputs: attitude setpoints , outputs: angular rates setpoints , controller: P-controller  $\left(\frac{1}{\tau}\right)$

**Control Law of Outer Loop of attitude controller**

```

// mix full and reduced desired attitude
Quatf q_mix = qd_red.inversed() * qd;
q_mix *= math::signNoZero(q_mix(0));
// catch numerical problems with the domain of acosf and asinf
q_mix(0) = math::constrain(q_mix(0), -1.f, 1.f);
q_mix(3) = math::constrain(q_mix(3), -1.f, 1.f);
qd = qd_red * Quatf(cosf(_yaw_w * acosf(q_mix(0))), 0, 0, sinf(_yaw_w * asinf(q_mix(3))));

// quaternion attitude control law, qe is rotation from q to qd
const Quatf qe = q.inverse() * qd;

// using sin(alpha/2) scaled rotation axis as attitude error (see quaternion definition by axis angle)
// also taking care of the antipodal unit quaternion ambiguity
Vector3f eq = 2.f * math::signNoZero(qe(0)) * qe.imag();

// calculate angular rates setpoint
matrix::Vector3f rate_setpoint = eq.emult(_proportional_gain);

```

- 2nd loop      inputs: angular rates setpoint , outputs: required forces and moments , controller: PID-controller

```

Vector3f rates_p_scaled = _rate_p.emult(pid_attenuations,_param_mc_tpa_break_p.get(), _param_mc_tpa_rate_p.get());
Vector3f rates_i_scaled = _rate_i.emult(pid_attenuations,_param_mc_tpa_break_i.get(), _param_mc_tpa_rate_i.get());
Vector3f rates_d_scaled = _rate_d.emult(pid_attenuations,_param_mc_tpa_break_d.get(), _param_mc_tpa_rate_d.get());

/* angular rates error */
Vector3f rates_err = _rates_sp - rates;

/* apply low-pass filtering to the rates for D-term */
Vector3f rates_filtered(_lp_filters_d.apply(rates));

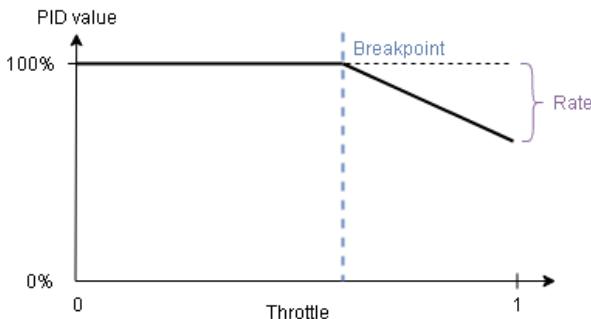
_att_control = rates_p_scaled.emult(rates_err) +
    _rates_int -
    rates_d_scaled.emult(rates_filtered - _rates_prev_filtered) / dt +
    _rate_ff.emult(_rates_sp);

_rates_prev = rates;
_rates_prev_filtered = rates_filtered;

```

The PX4 provides some measures to account for the nonlinear behavior of the motor in practice. For example if we noticed oscillations when we go towards full-throttle, we can solve this problem using 2 approaches:

1. Thrust Curve method :  $thrust = (1 - factor)PWM + (factor) PWM^2$   
the *factor* is stored in “THR\_MDL\_FAC” parameter so we can change it to model our motors characteristics.
2. PID attenuation method : after some value of the throttle (for example: 70%) called “break-point”, we can attenuate the gains to decrease the control effort according to a parameter called “rate”. The break-point value is configured using “MC\_TPA\_BREAK\_ \*” parameters , and the rate is configured using “MC\_TPA\_RATE\_ \* ” parameters.



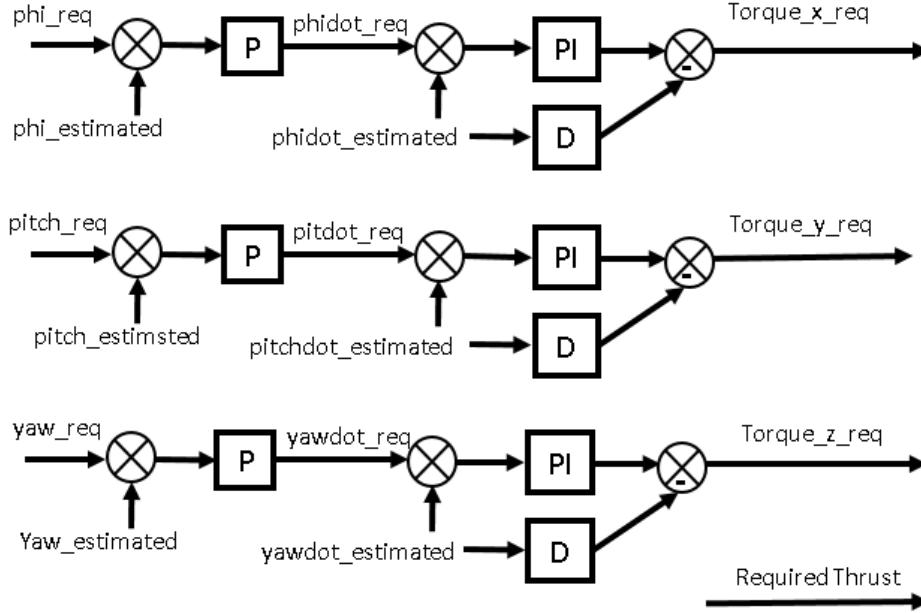


Figure 3.2.37: Attitude&amp;rate controller

### 3.2.3.5 Estimator

In PX4 flightstack, there are 4 types of estimators defined so we can choose the one that suits our application.

Estimator			
Q attitude estimator	INAV position estimator	LPE position estimator	EKF2
It's a quaternion based complementary filter for attitude.	It's a complementary filter for estimating 3D position and velocities.	It's an EKF(Extended Kalman filter) for estimating 3D position and velocities.	It's an EKF(Extended Kalman filter) for estimating 3D position , velocities , wind states and attitude.

We can choose the required estimator using `SYS_MC_EST_GROUP` parameter which has 3 possible values to choose the estimator as follows:

SYS_MC_EST_GROUP		
0	1	2
Q estimator (Complementary filter for attitude) INAV (Complementary filter for position)	Q estimator (Complementary filter for attitude) LPE (EKF for position)	EKF2 (all multicopter states and wind)

We used the **EKF2**.

### 3.2.3.6 Output drivers

They are responsible for the interfacing including the driver responsible for communicating with the motors ESCs to set the required PWM. They are include also a program called “Mixer” the required forces and moments goes to the mixer which separates the airframe configuraion from the controller. This generalizes the controller to various configurations”

There is also a very important software called “**Logger**”, it’s implemented in the Middleware. It’s very important to log data for tuning and performance analysis. The logger is automatically started when arming the vehicle and stopped when disarming it and a new log file is generated. There is 2 ways to have the logged data :

1. The most efficient way for large data size is using SD card on the PIXHAWK where the data is logged.
2. The Log Streaming way, which sends the same logging data via MAVLink. The requirement is that the link(for example WiFi) provides at least 50 KB/s and only one client can request log streaming at the same time. The connection doesn’t necessarily need to be reliable because the protocol can handle drops.

Both methods can be used independently or can be used at the same time. After getting the log file using any way we need to convert the generated “.ulg” file to another more useful data structure like “.csv” to open using excel or matlab, there is a lot of tools can handle the log file , analyze the data and generate graphs automatically offline like “PyFlightAnalysis” or online like “Flight Review”. We used a python tool to only convert the log file to excel files called “pylog”.

### 3.2.4 Gains Comparison

## 3.3 Quad-plane phase cruise controller

Our flying taxi doesn’t have control surfaces as conventional quad-plane but uses the quad-copter motors to achieve stability requirements. This configuration comes from the wing span constraint as flying taxi can’t have the wing span that should lift off the quad-copter motors as payload so we made use of them to help in lifting and stabilize the flying taxi during the quad-plane phase. The lift from the wings is used to increase the endurance and range by decreasing the thrust and power consumption of the quad-copter motors.

During the quad-plane phase the quad-copter controller is still on to stabilize the attitude and altitude of the flying taxi which means that  $\theta = 0, \Delta Z = 0$

To design the pusher controller which is used to achieve constant cruise speed, we can ignore the dynamics in the lateral directions as the flying taxi during this phase will conserve constant heading angle towards the goal beside the guidance controller will eliminate any cross-track error using roll angle. The pitch angle is zero and the altitude is held constant by the quad-copter controller. To design the cruise controller from equ.(3.1.13):

$$\dot{\Delta u} = -g\theta + \frac{\Delta X_{pusher}}{m} + \frac{X_u^{aero}}{m}\Delta u + \frac{X_w^{aero}}{m}\Delta w$$

based on the previous assumption of stabilized attitude and altitude this equation is reduces to:

$$\begin{aligned} \dot{\Delta u} &= \frac{\Delta X_{pusher}}{m} + \frac{X_u^{aero}}{m}\Delta u \Rightarrow \frac{\Delta u}{\Delta X_{pusher}} = \frac{1}{m(s - \frac{X_u}{m})} \\ X_u^{aero} &= -0.17669 \end{aligned}$$

The figure below shows the  $\Delta u$  step response and the required  $\Delta X_{pusher}$ :

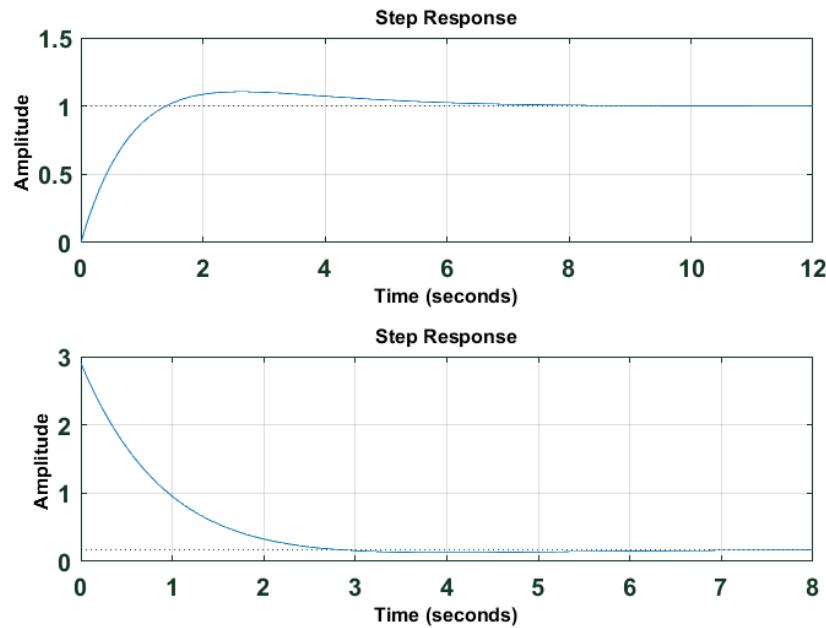


Figure 3.3.1: Cruise Controller

This figure shows the impulse response and the required control action:

## Chapter 4

# Vision navigation and Obstacle Avoidance

Unlike rovers or cars air vehicles don't have the option to use wheel encoders to determine its pose (state estimation) since they rely on GPS, IMU and vision to estimate their state, they subject to significant uncertainties. GPS accuracy is about 4 m but It's useful for long-term global position determination also GPS is with limited use indoor or closed places generally where there is a bad connection with satellites. The embedded estimator in the PIXHAWK which uses gyro and accelerometer to determine local position but the problem of using these inertial sensors is the dead-reckoning which drifts over time due to error accumulation. Although the reference gravity vector of the accelerometer is fused with the attitude from the gyros, long-term correction is still needed especially for high acceleration cases which causes deviation of the gravity vector. Usually GPS is used but it's insufficient due to its low accuracy and operating conditions. Here comes the need for visual odometry which is state estimation from visual data. Visual odometry is more accurate than GPS in state estimation and even in some cases from the inertial data when there is a suitable operating conditions and relatively low speed. For times of high speed or bad textured surrounding, the inertial data can aid the visual odometry. So GPS, IMU and Visual odometry can be used together for better state estimation. In our case, the ZED Stereo camera is used to get Point Cloud data<sup>1</sup> from which we can get visual odometry as well as information about obstacles. Obstacles information is used by the obstacle avoidance algorithm which evaluates the path planning and generates the reference trajectory. The visual odometry is fused with the PX4 position and/or attitude estimations to get a better state estimates.

### 4.1 Stereo Vision

---

<sup>1</sup>Point cloud is a set of data points in space

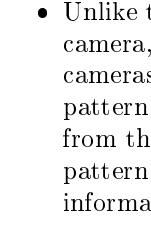
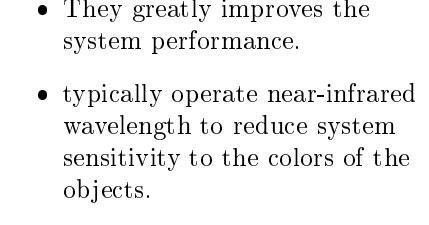
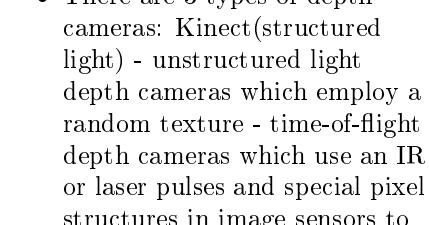
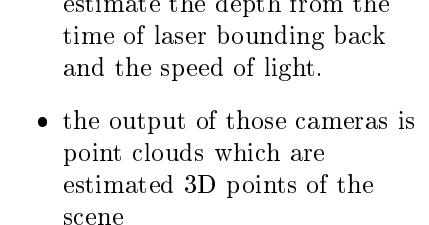
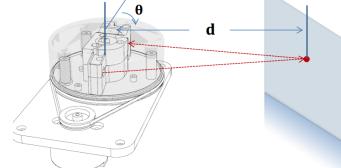
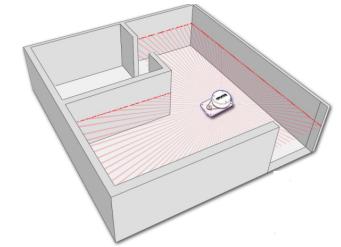
Stereo Camera	Depth Cameras	Laser Scanner
<ul style="list-style-type: none"> <li>• 2 cameras are rigidly mounted to a common mechanical structure.</li> <li>• Its performance depends on : mechanical design , resolution , lens type &amp; quality ,....</li> <li>• It can only estimate the distances to features like: sharp , high contrast corners (because they have high gradient in x,y directions). so It can't get distance to featureless wall but practically outdoor scenes have sufficient textures for stereo vision to work well.</li> <li>• ROS message: sensor\_msgs/Image and sensor\_msgs/CameraInfo</li> <li>• Image processing packages can be used to handle the outputs of the cameras to achieve the stereo vision</li> </ul>   <p>ZED stereo Camera : from 0.5 to 20m at 100FPS, indoors and outdoors.</p>	<ul style="list-style-type: none"> <li>• Unlike the passive stereo camera, they are active cameras that shine a texture pattern on the surfaces and from the deformation of the pattern they can get the informations of the scene.</li> <li>• They greatly improves the system performance.</li> <li>• typically operate near-infrared wavelength to reduce system sensitivity to the colors of the objects.</li> <li>• There are 3 types of depth cameras: Kinect(structured light) - unstructured light depth cameras which employ a random texture - time-of-flight depth cameras which use an IR or laser pulses and special pixel structures in image sensors to estimate the depth from the time of laser bounding back and the speed of light.</li> <li>• the output of those cameras is point clouds which are estimated 3D points of the scene</li> <li>• ROS message: sensor\_msgs/PointCloud2</li> </ul>     	<ul style="list-style-type: none"> <li>• superior accuracy (about 14 mm error)</li> <li>• longer sensing range than cameras (0.2 - 6 m)</li> </ul>   <ul style="list-style-type: none"> <li>• ROS message: sensor\_msgs/LaserScan</li> </ul>

Table 4.1.1: different sensors for vision

- **discussion**

Pose(position and orientation) estimation problem can be solved using :

- integration of IMU readings which degrads over time due to error integration
  - Visual Odometry (more accurate for pose estimation).  
In visual Odometry arises the scaling factor problem because we don't sense distance in real world and some possible solutions are:
    1. Known environment operation: for example, we can put a land mark with known size so when capture images we can extract scale factor and get distances from pixels.(off-board example[?], on-board example[?])
    2. We can use 1 camera + Ultrasound(which measures distances) [?]
    3. We can use 1 camera + accelerometer + pressure sensor [?]
    4. We can use 1 camera + IMU [?]
    5. We can use 2 cameras (Stereo vision) which eliminate the scaling factor problem but increases the computational cost
- Cameras are less energy consumer and lighter than 3D-localization sensors such as LIDARs
- There are 2 types of stereo processing algorithms:
  - Dense stereo algorithm: high computational cost. In Ref[?], they used 2 downward cameras and did the dense stereo processing on-board at frame rate of just 3Hz then they fused the visual odometry with laser scanner data.
  - Sparse stereo matching algorithm: faster - can maintain a high pose estimation rate of 30 Hz

- ZED stereo camera vs Kinect:

	ZED mini <sup>2</sup>	ZED stereo <sup>3</sup>	Kinect <sup>4</sup>	
Weight	62.9 gm	159 gm	425.2 gm	Weight is very important in aerospace applications
operating condition	indoor & outdoor	indoor & outdoor	indoor (due to the usage of IR)	Flaying Taxi is mainly outdoor application
Range	15 cm - 12 m	30 cm - 20m	1.2 m - 3.5 m	
Features	Motion Sensors : Gyroscope, Accelerometer with Sampling Rate: 800 Hz 6-axis Pose Accuracy : Position: +/- 1mm Orientation: 0.1° Technology : Visual-inertial stereo SLAM  Pose Update Rate : Up to 100Hz	-  same as mini  Technology : Real-time depth-based visual odometry and SLAM  Frequency : Up to 100Hz	RGB camera (640×480 pixels @ 30 Hz)  IR depth sensor (640×480 pixels @ 30 Hz)  multi-array microphone running proprietary software  full-body 3D motion capture, facial recognition and voice recognition capabilities	The ZED stereo does its calculations on a host hardware running SDK system which has the following requirements :  Dual-core 2,3GHz or faster processor / 4 GB RAM or more  Nvidia GPU with compute capability > 3.0 Note that ZED requires significant computation cost to apply the dense stereo matching at reasonable rate by using the GPU fucntions in OpenCV to accomplish this.
Connectivity	USB 3.0 Type-C port	USB 3.0 port with 1.5m integrated cable	USB 2.0	
Power	Power via USB 5V / 380mA	Power via USB 5V / 380mA	USB 12V, 1.08A (needs high power for the motors)	Kinect has a higher power consumption

Table 4.1.2: ZED stereo camera vs Kinect

<sup>2</sup><https://www.stereolabs.com/zed-mini/><sup>3</sup><https://www.stereolabs.com/zed/><sup>4</sup><https://en.wikipedia.org/wiki/Kinect>

- Using 4 cameras + IMU for self-localization[?]



Figure 4.1.1: Ref[?] MAV uses 4 cameras to get pose estimate real-time on-board

Paper in Ref[?] introduces novel design for an autonomous quadrotor by employing four cameras in two stereo configurations with on-board real-time processing:

- 2 forward cameras used with reduced(modified) SLAM based on PTAM<sup>5</sup> and it estimates the pose of the MAV
- 2 downward cameras used with ground-plane detection algorithm(to estimate the height , pitch and roll) and frame-to-frame tracking algorithm(to estimate the yaw and horizontal displacement)
- finally, they fused the data from the 2 stereo configurations and outputted an estimate for the MAV pose used to control the position and attitude of the MAV instead of IMU only unlike PIXHAWK
- They used stereo matching algoritnm twice for both the foraward stereo and backward stereo on-board
- They showed that adding 2 downward cameras significantly improved the self-localization

<sup>5</sup>Parallel Tracking And Mapping (PTAM), which is a popular open source SLAM implementation known for its robustness and efficiency.

- Hardware:
  - PIXHAWK
  - 4 USB cameras / gray scale image /  $640 \times 480$  :
    - \* 2 forward cameras : 11 cm baseline - frame rate of 30Hz
    - \* 2 downward cameras : 5 cm baseline - frame rate of 15Hz to reduce computations
    - \* both configurations are synchronised to capture data at the same time.
  - On-board computer with Intel Core 2 Due CPU with 1.86GHz to execute image processing and motion estimation
  - Microcontroller with IMU to send IMU data to the on-board computer and receive from it the high-level control instructions using I2c bus.

- The 2 forward facing cameras

- apply feature detection (800 feature upper bound) then do the sparse stereo matching
- SLAM employed to get pose from the successfully matched features

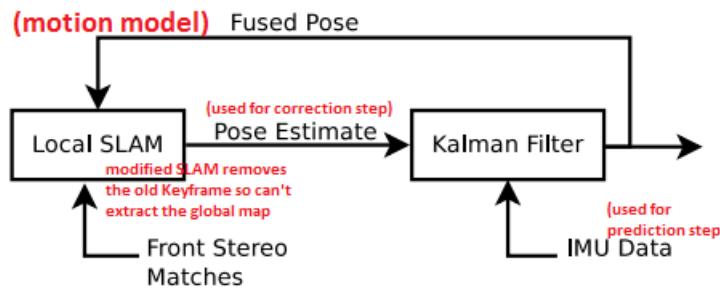


Figure 4.1.2: 2 forward-facing stereo configuration

- Rotation is predicted using ESM(Efficient 2nd order Minimization based image alignment method).

- The 2 downward facing cameras

- apply sparse stereo matching but with 300 feature at 15Hz because the algorithms used here are less sensitive to the features count and image rate than the above LocalSLAM
- RANSAC algorithm: to detect ground-plane & extract Height(h) , Pitch angle and Roll angle , unlike the height , pitch and roll extracted from the Local SLAM they here don't depend on the old values (In the LocalSLAM we use the old values in the motion model). So they resist drifting and give high accuracy
- frame-to-frame Tracking: They used the ESM algorithm to get the Affine transformation between 2 frames to get the Yaw and Horizontal displacement
- To have a large field of view (FOV) , they used small focal length which causes strong lens distortion which disrupt the frame-to-frame tracking, so they did image rectification first.

- Final Configuration

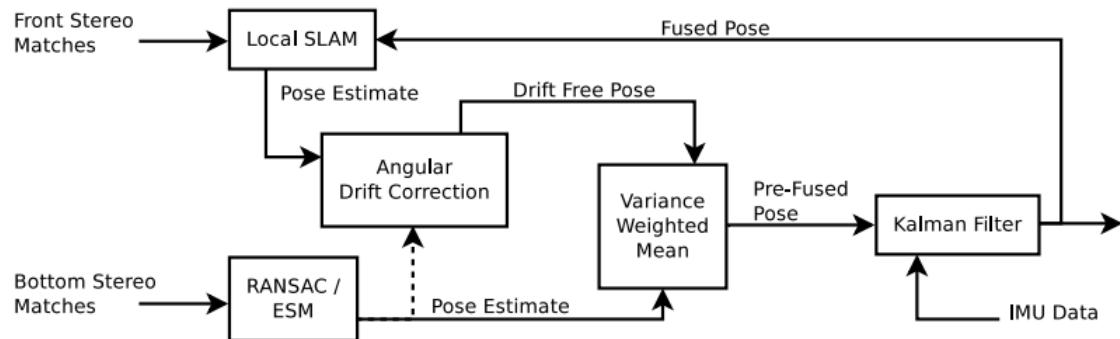
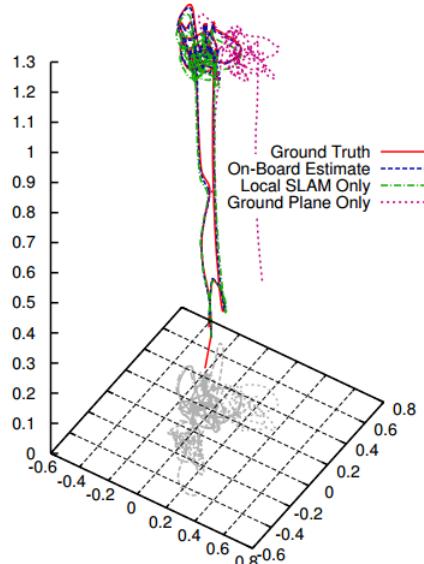


Figure 4.1.3: final configuration

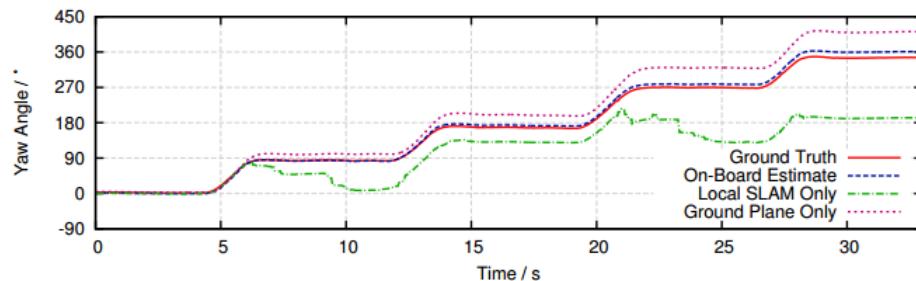
- Final Results



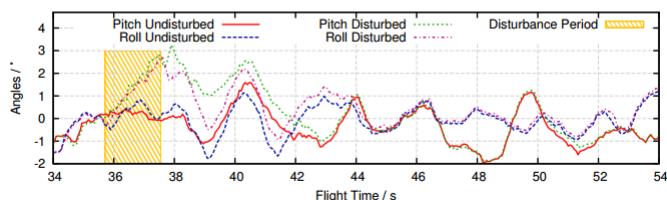
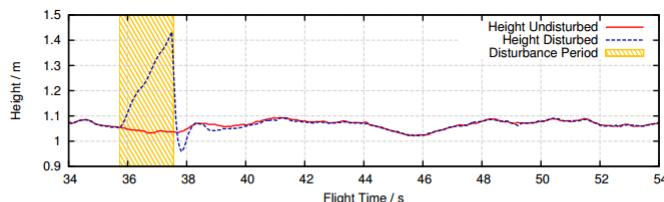
Ground truth and estimated position during autonomous take-off, hovering and landing. Scale is in meters.

Method	RMSE	Avg. Error
On-Board Estimate	1.41 cm	1.44 cm
Local SLAM only	3.14 cm	3.15 cm
Ground Plane Only	26.2 cm	23.7 cm

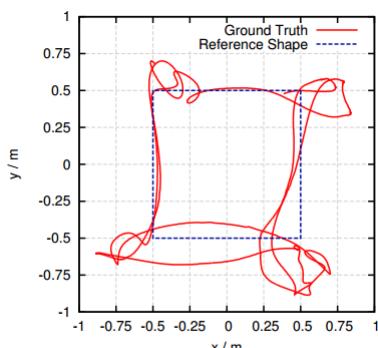
Position estimation errors for the examined processing methods.



Yaw angles measured during 360° yaw rotation.



Recovery of (top) height estimates and (bottom) roll- and pitch-angle after forceful disturbance.



Flight of a horizontal rectangle shape.

## 4.2 Obstacle Avoidance

Important terminologies in autonomous robots:

- Mapping : modelling the environment.
- Localization : estimating the pose of the robot with respect to a given model of the environment.
- SLAM (Simultaneous Localization and Mapping) : doing both tasks at the same time
- Motion Planning (Path planning) : determining the desired motion that satisfies constraints and optimize some aspect of movement.
- Navigation : determination of the robot position(**Localization**) and then planning a path towards some goal and avoiding obstacles(**Motion planning**). **Mapping** also can be done if needed and used for both Localization and Motion planning.

Autonomous robot architecture can be abstracted into 3 main layers:

Strategic Level (High-level planning)	Operational Level (Low-level planning)	Tactical Level (Execution level)
generating the waypoint or reference trajectory that the vehicle should follow. (Motion Planning)	how to follow the required path by generating the required forces and moments. (Guidance and Position&Attitude control)	how to achieve the required forces and moments using a specific motors configuration. (Mixer and Output drivers)

obstacle avoidance methods can be concluded as follows:

Global Obstacle avoidance	Local Obstacle avoidance
Grassfire algorithm - Dijkstra's algorithm - A* search	Bug algorithms - Vector field histogram

### 4.2.1 Local methods

It's called also "reactive" methods as they react to the sensor data at the current time step, unlike global methods they don't build a map of the environment so the generated path isn't optimal but they are memory efficient and fast enough to operate on-board.

- Bug Algorithms

inspired by insects which follow the obstacle wall to bypass it. There is many variants of Bug algorithms. They differ in sensors and amount of the memory used.

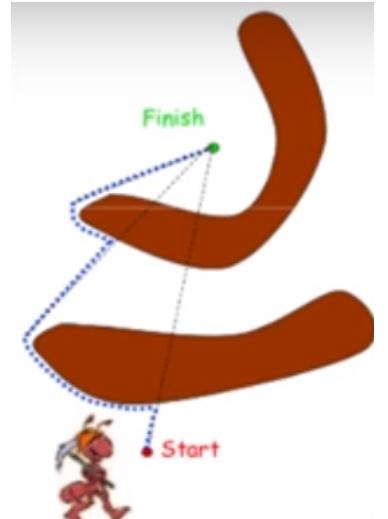
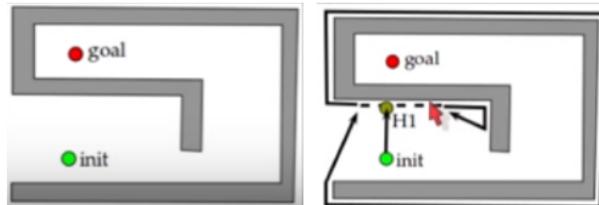
- Bug 0

\* follow obstacle boundary until goal is clear

\* No memory used

\* It's either right or left wall following

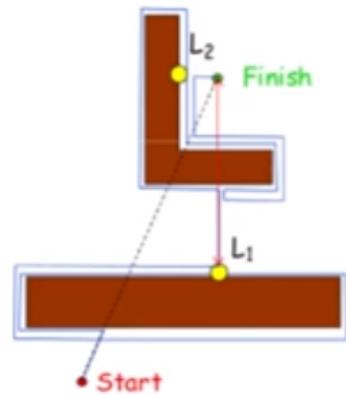
\* Don't guarantee **Completeness**<sup>a</sup>



<sup>a</sup>Assuming there is a path to the goal, Bug 0 don't guarantee finding it.

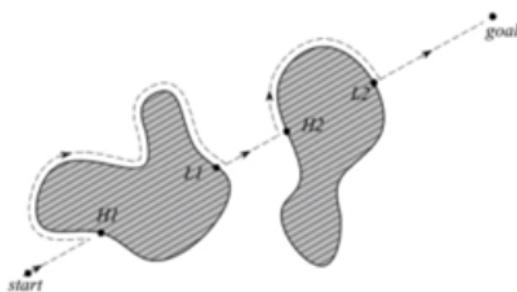
- Bug 1

- \* Fully circle the obstacle and stores the closest point to the goal.
- \* Leave at this point in the next circle
- \* inefficient but guarantee completeness



- Bug 2

- \* wall follow the obstacle and leave when meets the goal line such that the second point on the goal line is closer to the goal than the first point at which we left the goal line and started to follow the wall.
- \* This method is complete.<sup>a</sup>
- \* Faster than Bug.

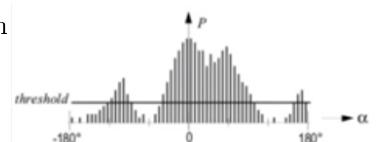


<sup>a</sup>If there is a path to the goal, it will find it.

- Vector Field Histogram

- VFH

creates local occupancy grid (1D polar histogram) via recent readings to specify the probability of object existence at every angle in the front view of the vehicle.



- VFH+

includes kinematic constraints due to which the vehicle can't go in specific directions not only object existence constraint. Finally, the chosen path is evaluated using cost function which has 2 weighting parameters: the target direction<sup>6</sup> and the previous direction<sup>7</sup>

- VFH\*

enhance the global optimallity problem in VFH & VFH+ by using A\* search algorithm to minimize the cost function and heuristic function used to guide the search in the goal direction. It has been shown that it can successfully deal with problematic situations that VFH & VFH+ can't handle.

<sup>6</sup>

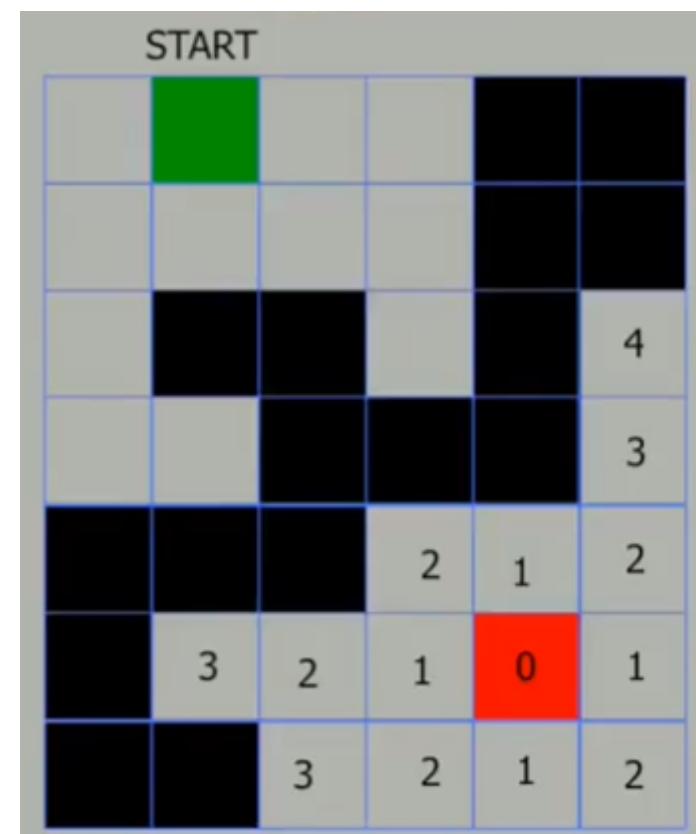
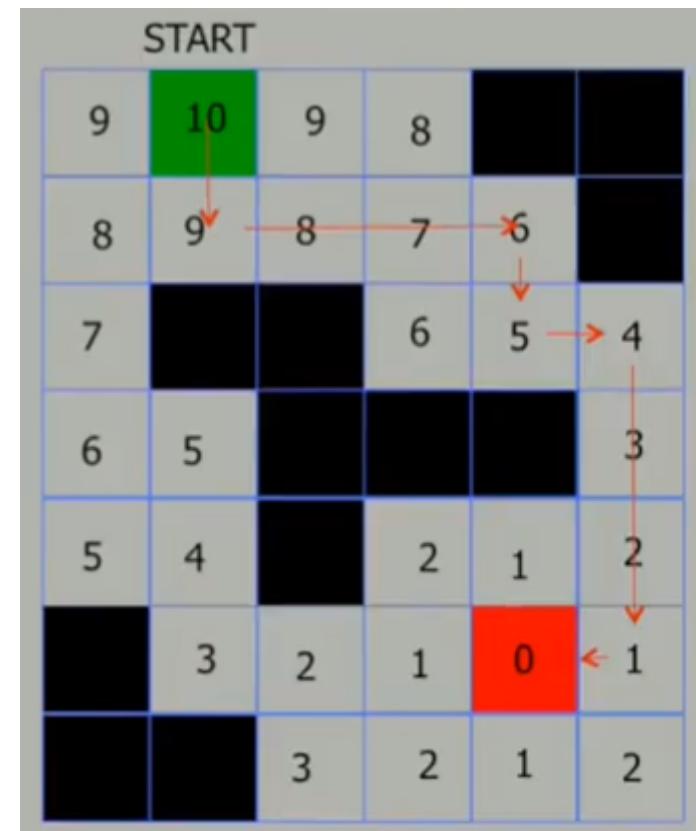
\* difference between the new direction and the goal direction

<sup>7</sup>difference between the previously selected direction and the new direction

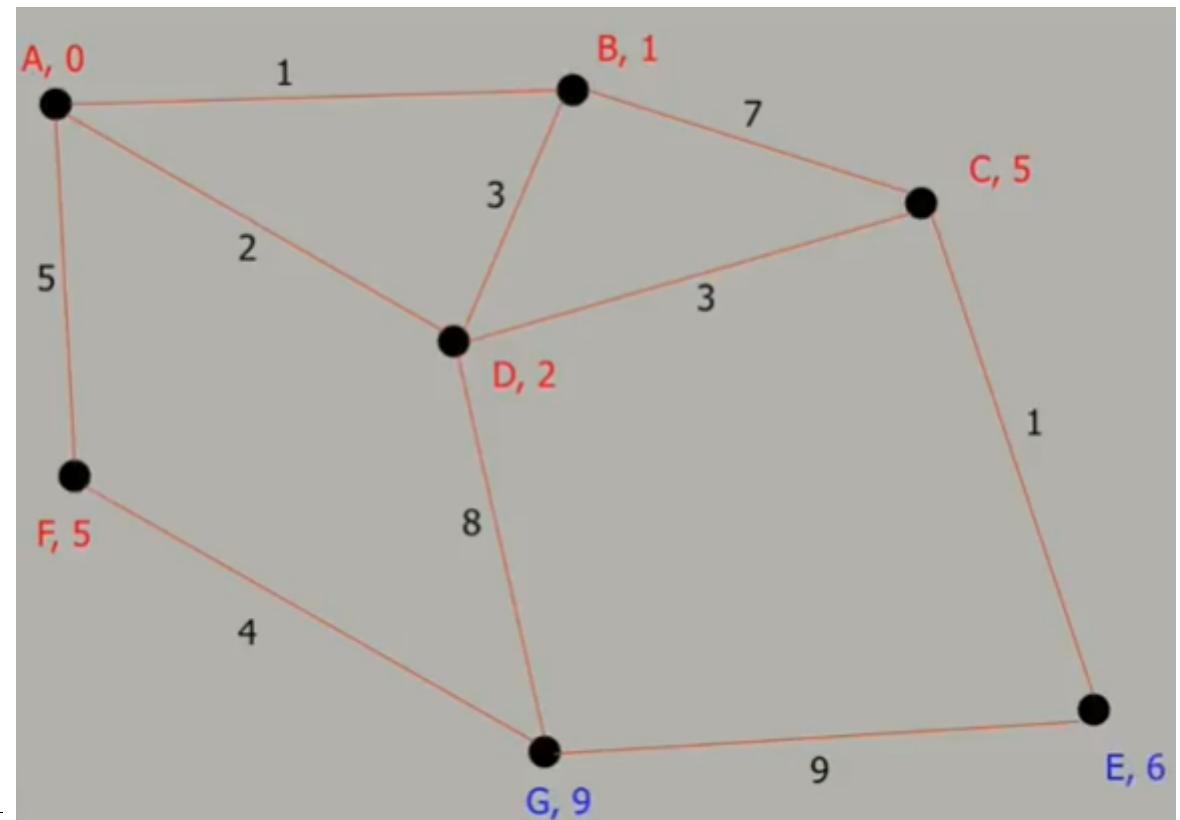
### 4.2.2 Global methods

The vehicle has a map of the environment or builds it at run-time then computational motion planning methods, it finds the optimal path to the goal which avoids any obstacles.

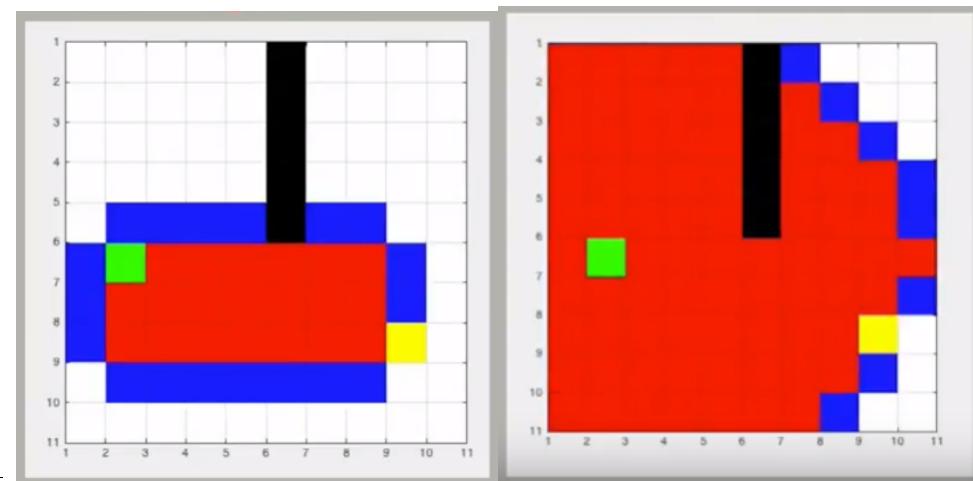
- Grassfire algorithm



- Dijkstra's algorithm



- A\* search algorithm



# Chapter 5

# Implementation

## 5.1 ROS

### 5.1.1 What is ROS

- ROS is “Robot Operating System”
- An open source frame work for building robots
- provides tools and libraries for helping software developers to create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, ... .
- ROS is licensed under an open source, BSD license.



Figure 5.1.1: ROS architecture

#### 5.1.1.1 History of ROS

- originally developed in 2007 at the Stanford Artificial Intelligence Laboratory.
- Since 2013 managed by OSRF<sup>1</sup>.
- Today used by many commercial robots, universities and companies.
- It became a standard for robot programming

<sup>1</sup>Open Source Robotics Foundation

### 5.1.1.2 ROS Philosophy

**Peer-to-Peer-communication:** individual programs communicate over defined API<sup>2</sup>, (ROS messages—Services- Action)

**Merit:** everyone can work in separate part from the big project easily also there will be clear structure of the interactions between different subsystems in our UAV.

**Distributed** can be run on multiple computers and communicate over the network.

**Merit:** It is very useful in SLAM projects

**Multi-lingual** we can use Python , C++ , and even LabVIEW and Arduino C . . .

**Light-weight** standalone libraries are wrapped around with a thin ROS layer.

**Merit:** It is very important in SLAM because we will distribute the jobs between multi-computers so every HW will have only the SW it needs.

**Free & Open source** most ROS software is open source & free to use.

**Eco-system** Large supporting community , tutorials and powerful documentation (WikiROS). It is used by most if not all the robotics community so It will be good if we can contribute with our project in this community.

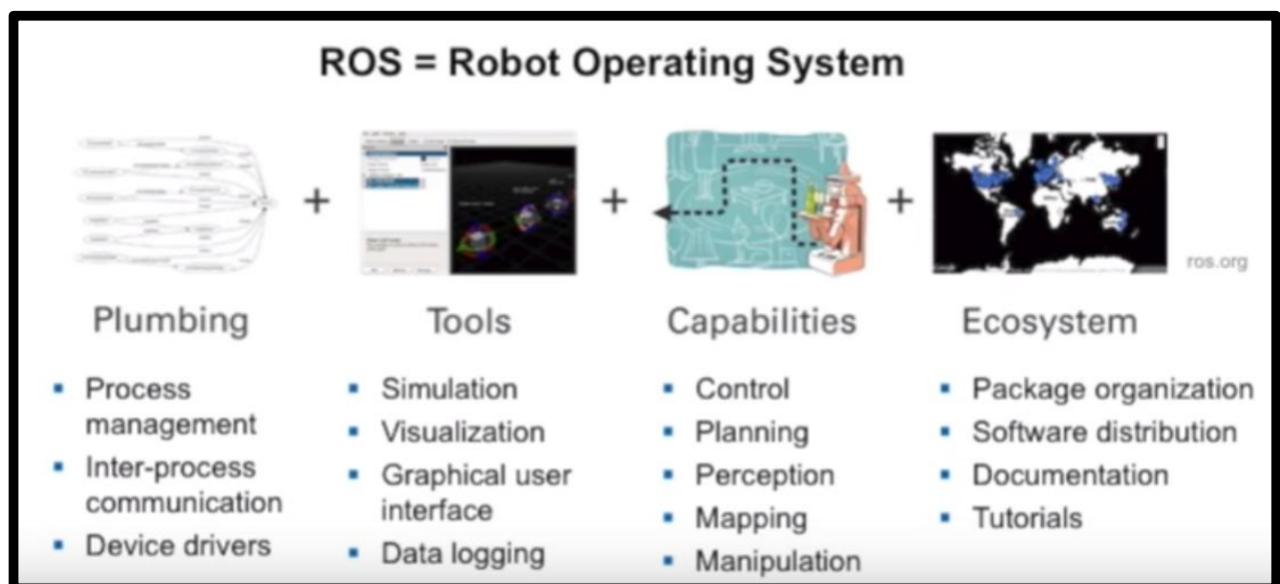


Figure 5.1.2: Why ROS?

#### NOTE

Despite the proved power of using LabVIEW&MyRIO in building Autopilots , in all LabVIEW community there are only 2 topics (in the time this project is started) which talking about using LabVIEW + LabVIEW Robotics Environment Simulator in implementing simple SLAM (EKF based) and Obstacle avoidance on an 2D differential-drive robot. So it is very useful to integrate ROS with LabVIEW&MyRIO or replacing them with ready-made (C++ or Python) packages if possible<sup>3</sup>to reduce the cost (MyRIO is expenssive).

<sup>2</sup>Application programming interface : it is a set of methods of communication among various components

<sup>3</sup>If that will achieve the real-time requirements which is a must in Aeronautical applications

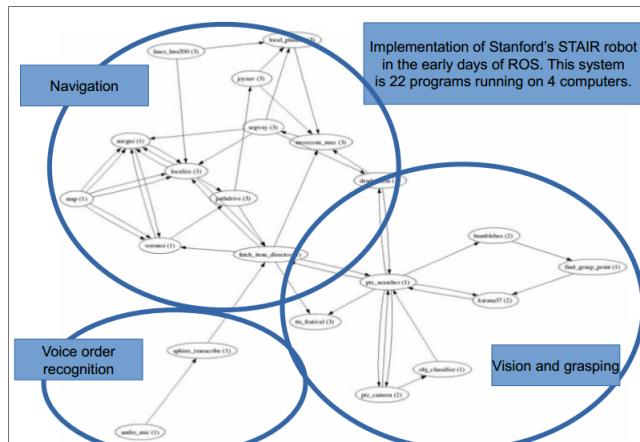


Figure 5.1.3: ROS graph of Stanford's STAIR robot nodes.

### 5.1.2 Pros and Cons of ROS

#### 5.1.2.1 Advantages:

1. It's a common software platform for those who are building and using robots
2. Makes people share code and ideas more readily so we do not have to spend much time writing software infrastructure before robot starts moving!
3. ROS has been remarkably successful in 2015 there were: over 2,000 software packages, 80 commercial robots are supported and at least 1850 academic papers that mention ROS.
4. We no longer have to write everything from scratch so we can concentrate on the part we are working on control,state-estimation,planning,... .
5. We no longer need to write device drivers as a set of drivers that let you read data from sensors and send commands to motors and other actuators written already for many hardwares
6. A large collection of fundamental robotics algorithms that allow you to build maps , navigate, represent and interpret sensor data, plan motions,manipulate objects, ... .
7. All of the computational infrastructure that allows you to move data around, to connect the various components of a complex robot system, and to incorporate your own algorithms. ROS allows you to split the workload across multiple computers.
8. ROS is an ecosystem includes an extensive set of resources, such as a wiki , a question-and-answer site.

#### 5.1.2.2 Disadvantages:

There's a fair amount of complexity in ROS such as : distributed computation, multi-threading , event-driven programming, and other concepts lie at the heart of the system. If you're not already familiar with at least some of these, ROS can have a hard learning curve<sup>4</sup>

### 5.1.3 Example

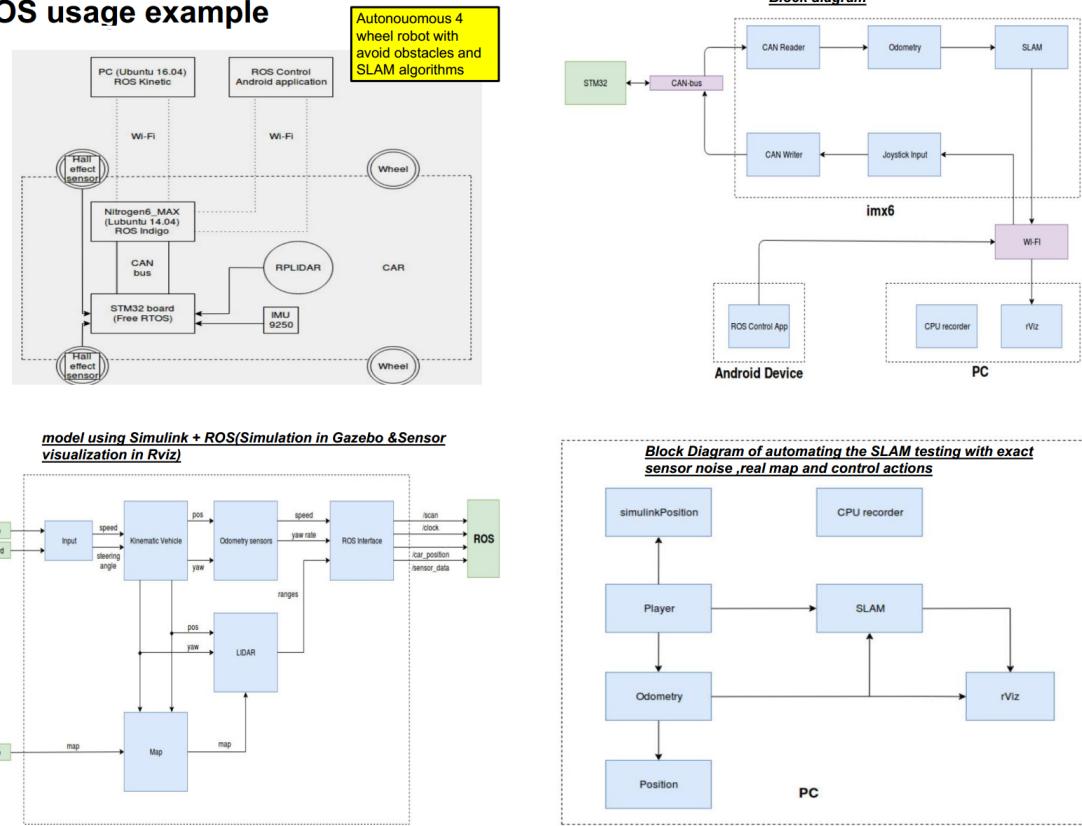
This is an example of a ROS-based project of a 4-wheel autonomous robot with obstacle avoidance and SLAM algorithms.

- a PC operating ROS is used as a ground control station to communicate with the robot
- an android device operating ROS control application is used as a joystick.
- a mini-onboard PC operating ROS is used to receive all controls (Waypoints,...) and to process SLAM algorithm and obstacle avoidance.
- STM32 board running free-RTOS is used as a low-level controller (deals with the motors) also it's used as an interface for gathering data from sensors(Hall effect sensor, IMU sensor and RPLIDAR) to send these data to the mini-onboard PC to be processed.

---

<sup>4</sup>but can be flattened out a bit by introducing the basics of ROS and having some practical examples of how to use it for real applications on real (and simulated) robots which is already exist in many books and online courses.

## ROS usage example



Rviz used to visualize the sensors outputs

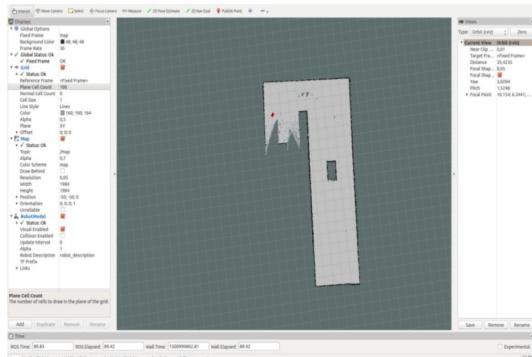


Figure 5.1.4: ROS example

### 5.1.4 Summary of some important features in ROS

- Catkin

1. Catkin is a collection of CMake macros and associated code **used to build packages** used in ROS.

2. It was initially introduced as part of the ROS Fuerte release where it was used for a small set of base packages.

For Groovy and Hydro it was significantly modified, and used by many more packages. All released Hydro packages were built using catkin, although existing rosbuild packages can still be built from source on top of the catkin packages.

Indigo is very similar, except for some deprecated features that were removed.

- Catkin Workspace

1. catkin packages can be built as a standalone project, in the same way that normal cmake projects can be built, but catkin also provides the concept of workspaces, **where you can build multiple, interdependent packages together all at once**.

2. A catkin workspace **is a folder where you modify, build, and install catkin packages**. The following is the recommended and typical catkin workspace layout:

```

workspace_folder/          -- WORKSPACE
src/                      -- SOURCE SPACE
  CMakeLists.txt          -- The 'toplevel' CMake file
  package_1/
    CMakeLists.txt
    package.xml
    ...
  package_n/
    CATKIN_IGNORE          -- Optional empty file to exclude package_n from being processed
    CMakeLists.txt
    package.xml
    ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE          -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...

```

- Source Space (Folder)

The source space contains the source code of catkin packages. This is where you can extract , checkout , or clone source code for the packages you want to build. Each folder within the source space contains one or more catkin packages. This space should remain unchanged by configuring, building, or installing.

- Build Space (Folder)

The build space is where CMake is invoked to build the catkin packages in the source space. CMake and catkin keep their cache information and other intermediate files here. The build space does not have to be contained within the workspace nor does it have to be outside of the source space, but this is recommended.

- Development (Devel) Space (Folder)

The development space (or devel space) is where built targets are placed prior to being installed. The way targets are organized in the devel space is the same as their layout when they are installed. This provides a useful testing and development environment which does not require invoking the installation step.

- Install Space (Folder)

Once targets are built, they can be installed into the install space by invoking the install target, usually with make install. The install space does not have to be contained within the workspace. Since the install space is set by the CMAKE\_INSTALL\_PREFIX, it defaults to /usr/local, which you should not use (because uninstall is near-impossible, and using multiple ROS distributions does not work either).

- default workspace is loaded with:

```
>source /opt/ros/indigoa/setup.bash
```

<sup>a</sup>replace with your ROS distribution

- usually this instruction is written in the ~/.bashrc to launch automatically when you open a terminal

overlay your catkin workspace with :

```
>cd ~/catkin_ws  
>source devel/setup.bash
```

Or another approach if you are using only one workspace in your /opt/ros/indigo/setup.bash write :

```
>source ~/catkin_ws/devel/setup.bash
```

check your catkin workspace with by:

```
>echo $ROS_PACKAGE_PATH
```

Or by writing

```
>roscore
```

you should get to directory: ~/catkin\_ws/ not /opt/ros/indigo/

- A catkin workspace can contain up to four different spaces which each serve a different role in the software development process.
- To clean the build and devel folders without touching src folder so you can build the src again use:

```
>catkin clean
```

- **ROS master**

- manages the communication between nodes
- Every node registers at startup with the master
- start a master with:

```
>roscore
```

starts multiple elements including rosmaster

---

- **ROS nodes**

- single-purpose executable program
- individually compiled, executed and managed
- organized in packages
- Run a node:

```
>rosrun package_name node_name
```

See active nodes:

```
>rosnode list
```

Retrieve info about a node:

```
>rosnode info node_name
```

- ROS can launch identical nodes into separate namespaces for example if we have a node that generates PWM to one motor called “motor\_node”, we can run it as /left\_motor/motor\_node and /right\_motor/motor\_node and when running the motor\_node we can pass to it the pin number of the motor depending on the namespace it will be run in.
- 

- **ROS topics**

- Typically, there is a node which is a publisher (talks or send msgs) and another one which is a subscriber (listens or receives msgs). They can communicate with each other using Topics which act as a channel to pass data from publishers to subscribers.
- It is very normal to have multiple subscribers on the same topic but we should avoid multiple publishers to avoid race conditions

- >rostopic list

```
>rostopic echo /topic
```

```
>rostopic info /topic
```

to publish in a topic manually (not from a real node):

```
rostopic pub /cmd_vel geometry_msgs/Twist "Linear: x:0.0 y:0.0 z:0.0 angular: x:0.0 y:0.0 z:0.0"
```

/cmd\_vel is a topic used usually to control angular and linear velocities in 3D. It holds a msg called “Twist” which is defined in a package called “geometry\_msgs” and this msg contains Linear and Angular velocities in x,y, and z directions

- remapping is a very useful feature ROS can do on topics. For example if I have a node called “image\_view” that reads from a topic called “image” which is published by the “camera” node but we have two cameras (left and right) so we can remap the topic to exist in two separate namespaces each holding different data for each camera but definitely the same msg so now we can have “/right/image” topic and “/left/image” topic and make “image\_view” node to read from both.
-

- **ROS messages**

- Data structure defining the type of a topic.
- composed of a nested structure of integers, floats, booleans, strings, arrays of objects, ...
- We can define one in \*.msg files but It is preferred to use ready-made ROS messages to keep compatible with the community besides there is a variety of messages that we may never need to define new one.
- >rostopic type /topic\_name  
>rostopic pub /topic\_name msg\_type data

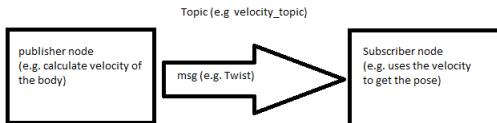


Figure 5.1.5: ROS msgs

- **ROS Launch**

- launch is a tool for launching multiple nodes (as well as setting parameters) because it isn't practical to use rosrun on every node.
- We can define launch file as \*.launch in launch folder in our package.
- If the roscore isn't running, launch automatically starts a roscore
- > roslaunch package\_name launchfile.launch

- **ROS Packages**

- can be thought of as a collection of resources that are built and distributed together.
- Package folder tree:

```

package_name
config ==> parameter files(YAML)
include/package_name ==>c++ header files
launch ==> *.launch files
src ==> source files
test ==> ROS tests
CMakeLists.txt ==> CMake build file
package.xml ==> package information

```

- Usually package depends on other packages for example other package holds our custom msgs, services and actions :

```

package_name_msgs
action ==> action definitions
msg ==> message definitions
srv ==> service definitions
CMakeLists.txt
package.xml

```

this separation of our custom msgs in a separate package from other packages is a good practice.

- for creating a package go to ~/catkin\_ws/src and use this instruction in the terminal:  
>catkin\_create\_pkg package\_name <dependencies> {for example: package\_name\_msgs}

- package.xml defines:
    - package name
    - version number
    - authors
    - dependencies on other packages
  - CMakeLists.txt It is the input to the CMake build system and defines:
    - required CMake version
    - Package name
    - find other CMake or catkin packages needed for build: `find_package()`
    - for adding msgs / actions / services: `add_message_files()` - `add_action_files()` - `add_service_files()`
    - invoke msg / service / action generation: `generate_messages()`
    - Libraries and Executables to build: `add_library()` - `add_executable()` - `target_link_libraries()`
- 

- **ROS services**

- Unlike topics which multiple nodes can read and write from and into the topic, service consists of server (to conduct the service) and a node (that receive the service).
  - Unlike action (as we will see in ??) the node will wait for the server to finish and then it will continue working.
  - From this we can say that we can use services in tasks that executed instantaneously and actions in tasks that takes much time to be executed like for a robot to go to a specific point.
  - To call a service from a terminal:

```
>rosservice call /service_demo "{}"
```

“{}” is an Empty msg we can use if we want to only excite the server without data needed to be send. This msg is defined in package called “std\_msgs”
-

- ROS actions

- Similar to service but the node doesn't need to stop until the action is finished
- Internally actions uses topics to communicate with the node that called the action server

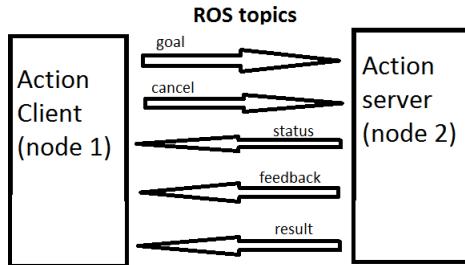


Figure 5.1.6: ROS actions

- status : 0(pending) - 1(active) - 2(done) - 3(warning) - 4(error)  
we can get the status in node1.py :

```
status = client.get_status()
DONE = 2
while state_result < DONE
    DO SOMETHING unlike server where we waited for server to finish
```

ETH zürich

### ROS Parameters, Dynamic Reconfigure, Topics, Services, and Actions Comparison

	Parameters	Dynamic Reconfigure	Topics	Services	Actions
Description	Global constant parameters	Local, changeable parameters	Continuous data streams	Blocking call for processing a request	Non-blocking, preemptable goal oriented tasks
Application	Constant settings	Tuning parameters	One-way continuous data flow	Short triggers or calculations	Task executions and robot actions
Examples	Topic names, camera settings, calibration data, robot setup	Controller parameters	Sensor data, robot state	Trigger change, request state, compute quantity	Navigation, grasping, motion execution

Figure 5.1.7: comparison between ROS parameters, Dynamic Reconfigure , Topics , Services , and Actions

## 5.2 Proof of Concept

### 5.2.1 State Estimation (Rover)

We wanted to increase our experience in ROS so we decided to build a differential drive robot using ROS. In addition the very useful feature of ROS is that we can use the Odometry message resulting from encoders (very simple sensors) to simulate the results of algorithms that handles the vision navigation. Hence, this project deepens our understanding of ROS independently of the complexity of the sensors (Cameras and Lidars) and the controlled system(Multi-rotor which is more critical in control than DD robots).

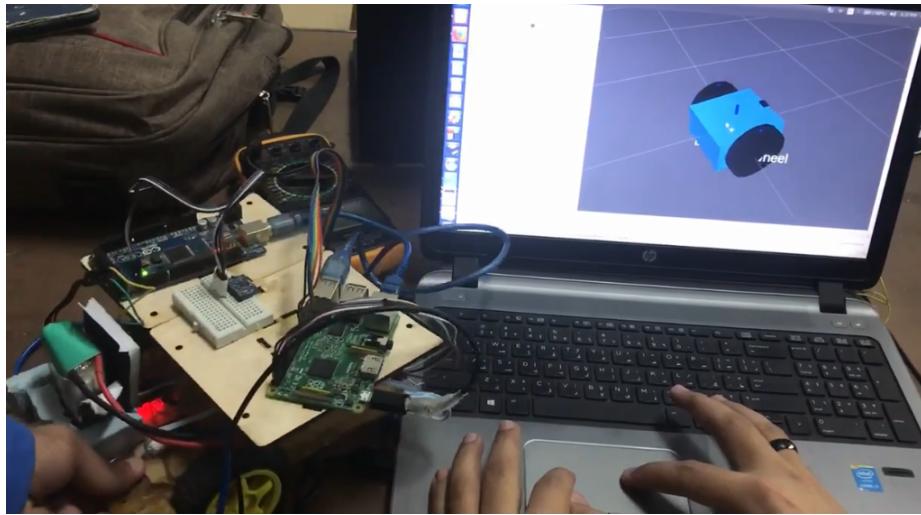


Figure 5.2.1: Differential Drive Robot (DD robot)

- **Objective**

1. Control the DD robot linear velocity and angular velocity , which will be the required inputs to the DD robot from the high-level algorithms as shown in ?? , this control needs encoders and IMU as feedback in case of using Incremental encoders , encoders only in case of using quadrature encoders , or fusion of encoders and IMU. In our case , It is very fine that our DD robot won't have backward movement for motors so we don't need to neither fuse data for the linear and angular velocities control loop nor using quadrature encoders.
2. Localize the DD robot by fusing the data from IMU and odometry from encoders

- **Modeling**

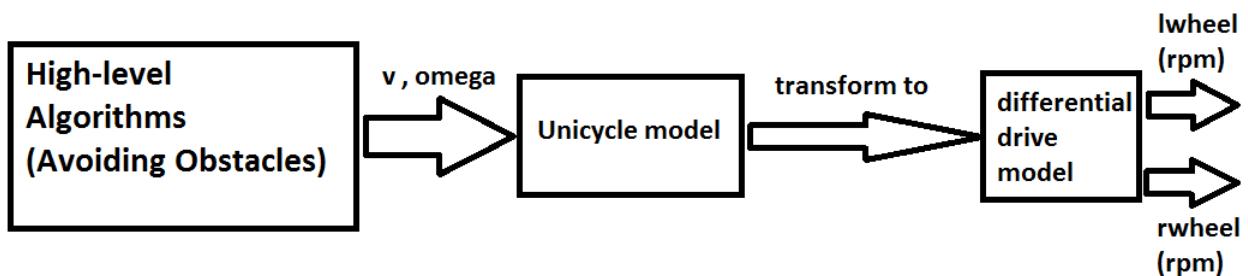


Figure 5.2.2: differential drive model

- **Block Diagram**

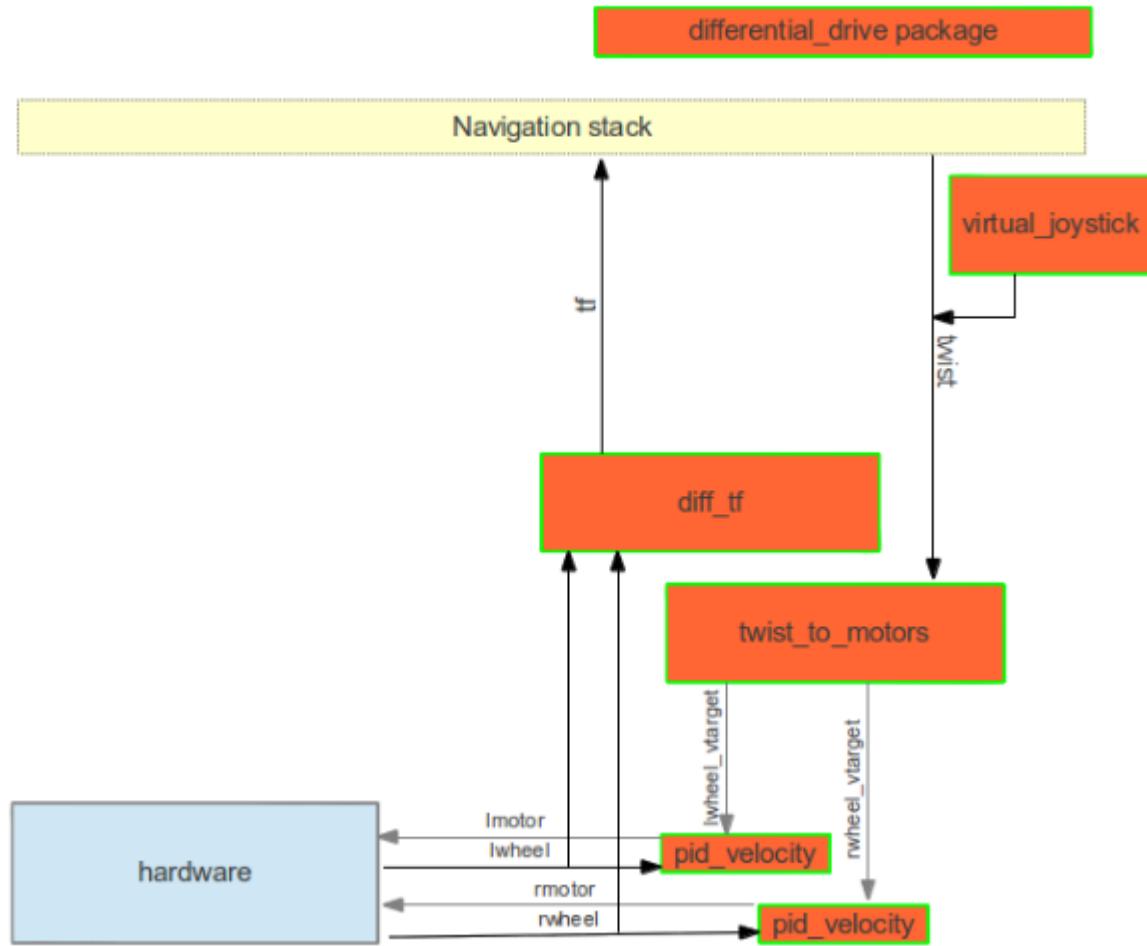
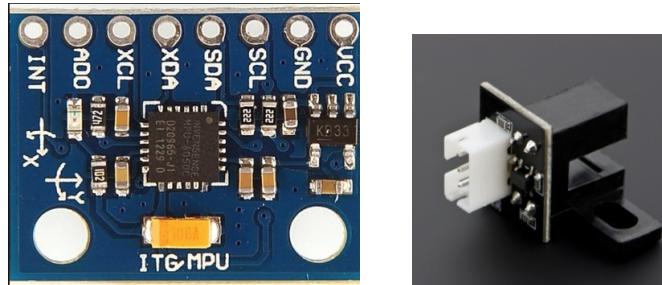


Figure 5.2.3: differential\_drive package

- **Hardware**

MPU 6050 & Encoder



### 5.2.1.1 Software

Control the DD robot linear velocity and angular velocity

- nodes

package	node name	description	parameters
differential_drive	twist_to_motors	transforms the required linear velocity and angular velocity(Twist) of the DD robot to required velocity of motors.	~rate (float, default:10) ticks_meter (int, default:50) ~base_width (float, default:0.245) ~base_frame_id (string, default:"base_link") ~odom_frame_id (string, default:"odom") encoder_min (int, default:-32768) encoder_max (int, default: 32768) wheel_low_wrap wheel_high_wrap
			~Kp (float, default:10) ~Ki (float, default:10) ~Kd (float, default:0.001) ~out_min (float, default: -255) ~out_max (float, default: 255) ~rate (float, default: 20) ~rolling_pts (float, default: 2) ~timeout_ticks (int, default: 2) ticks_meter (float, default: 20) encoder_min (int, default: -32768) encoder_max (int, default: 32768) wheel_low_wrap wheel_high_wrap
	diff_tf	produces the axis transformations of the DDrobot due to the movement which detected by encoders, and publishes the odometry(the current pose and twist) of the robot.	~rate (float, default:10) ticks_meter (int, default:50) ~base_width (float, default:0.245) ~base_frame_id (string, default:"base_link") ~odom_frame_id (string, default:"odom") encoder_min (int, default: -32768) encoder_max (int, default: 32768) wheel_low_wrap wheel_high_wrap

- topics

name	msg	Published by	Subscribers
odom	nav_msgs/Odometry	diff_tf	any one
tf	tf/tfMessage	diff_tf	/tf
lwheel	std_msgs/Int16	left encoder	pid_velocity and diff_tf
rwheel	std_msgs/Int16	right encoder	pid_velocity and diff_tf

### 5.2.1.2 Localization

- nodes

package	node name	description	parameters
teleop_twist_keyboard[?]	teleop_twist_keyboard	publishes the required linear velocity and angular velocity in the /cmd_vel topic	<p>_speed is the required linear velocity</p> <p>_turn is the required angular velocity</p>

- topics

name	msg	Published by	Subscribers
/cmd_vel remapped to /Twist to be identified by twist_to_motors node	nav_msgs/Odometry	teleop_twist_keyboard	differential_drive_pkg/twist_to_motors node

### 5.2.2 Control (half quad)

- Block diagram & Software

Purpose of this experiment is to validate ROS performance in aerospace applications and compare its performance with Ardupilot.



- Configuration 1 (ROS pid)

using ready-made PID in ROS community with Low-pass filter on derivative term.

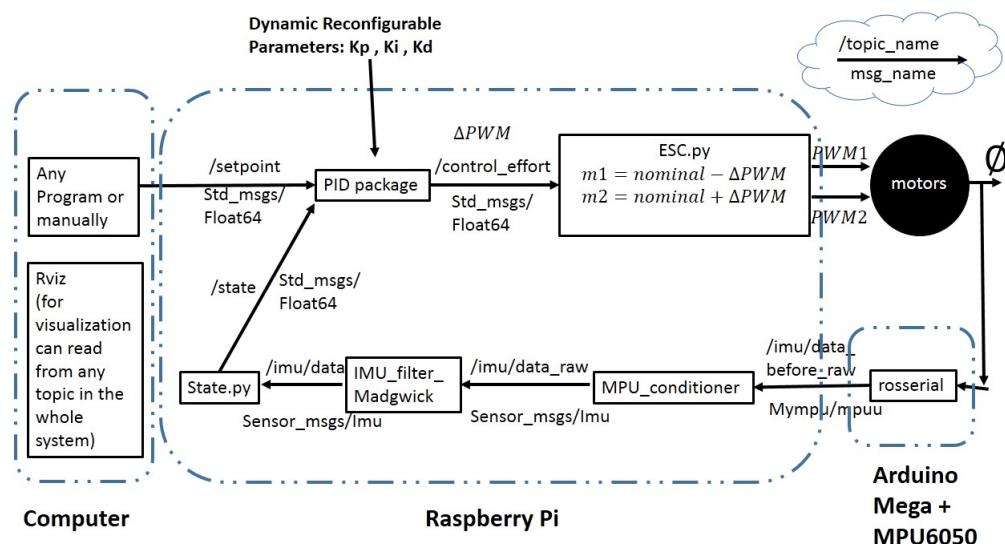


Figure 5.2.4: 1D Bi-rotor conf1 Block Diagram

- Configuration 2 (myPIDA)

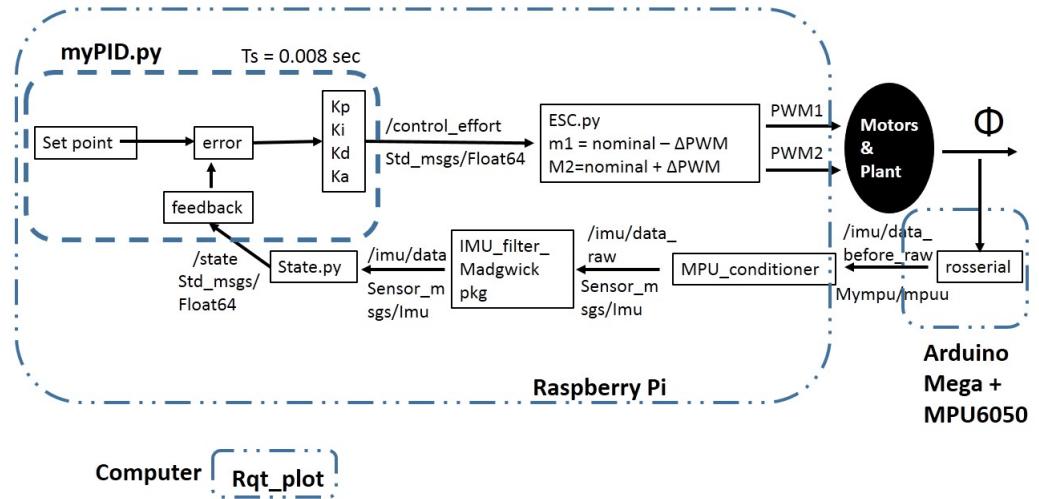


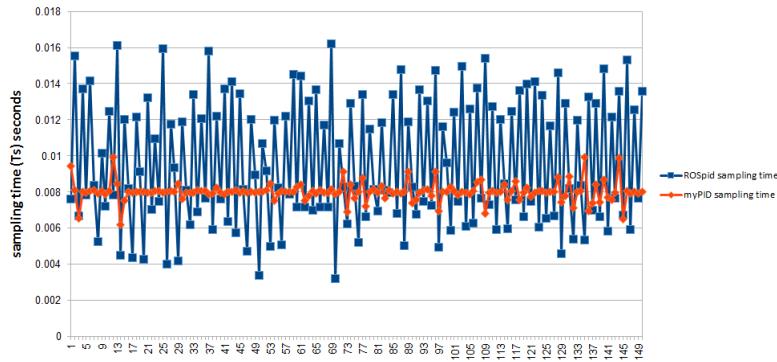
Figure 5.2.5: 1D Bi-rotor conf2 Block Diagram

Package	Node	Parameters
myPID	PID_node	$T_s$ sampling time = 0.008 (default)
		Kp
		Ki
		Kd
		Ka

### 5.2.2.1 Results & Conclusions

- Sampling Time

configuration 2 has better control on sampling time than configuration 1 because config1 depends on the frequency of the setpoint and the feedback , we build a similar code to ROSpid and the Ts was collected and plotted alongside with Ts achieved by myPID as shown



To show the effect of not constant sampling time , we choose the gains as  $K_p = \text{some value}$  ,  $K_i = K_d = K_a = 0$  , so the control should be  $= K_p * \text{error} = - K_p * \text{state}$  which is simply a scaled reflection of the state if the sampling time is constant and synchronous with the feedback.

- IMU Bias

We noticed that when motors are on , there is biasing in the IMU readings

- Gains tuning

## 5.3 px4 and Mavros

### 5.3.1 Introduction

The definition of Auto pilot in general that it is a system used to control the trajectory of an aircraft without constant 'hands-on' control by a human operator being required.

Autopilots do not replace human operators but assist them in controlling the aircraft, allowing them to focus on broader aspects of operations such as monitoring the trajectory, weather and systems.

- **What's Pixhawk**

Pixhawk is an independent open-hardware project that aims to provide the standard for readily-available, high-quality and low-cost autopilot hardware designs for the academic, hobby and developer communities.

- **Other devices like Pixhawk**

**Arducopter :** This is the full-featured, open-source multicopter UAV controller that won the Sparkfun 2013 and 2014 Autonomous Vehicle Competition (dominating with the top five spots).

A team of developers from around the globe are constantly improving and refining the performance and capabilities of ArduCopter.

Copter is capable of the full range of flight requirements from fast paced FPV racing to smooth aerial photography, and fully autonomous complex missions which can be programmed through a number of compatible software ground stations. The entire package is designed to be safe, feature rich, open-ended for custom applications, and is increasingly easy to use even for the novice.

- **Comparison between various autopilot devices**

The APM flight controller uses an Atmega2560, uses an I2c bus and runs at 16mhz, Pixhawk uses an Arm Cortex 32bit STM32 F4 running at 168mhz on a much faster SPI bus, has CAN bus, also faster and has a backup processor STM32 F1 chip that runs at 72 Mhz. All the original CC3D flight controllers use the STM32 F1 chip, the better ones use the STM32 F3 chip that adds a floating point coprocessor and more memory, newer ones use the STM32 F4 and the newest ones use STM32 F7 at 216mhz. A faster processor computes better PID loop times and creates a more stable and responsive platform with increased capabilities. All these run their motion sensors on the faster SPI bus, they use the slower I2c bus for less critical things.

- **Why Pixhawk**

We choose the Pixhawk over Arducopter for the following reasons

- It has much more capabilities and processing power .
- It can connect using two telemetry ports one for ground station and the other for manual control using a flight controller.
- It has faster response for actions which is good.
- It supports quad plane mode which is similar to our design.
- Available in Egypt.

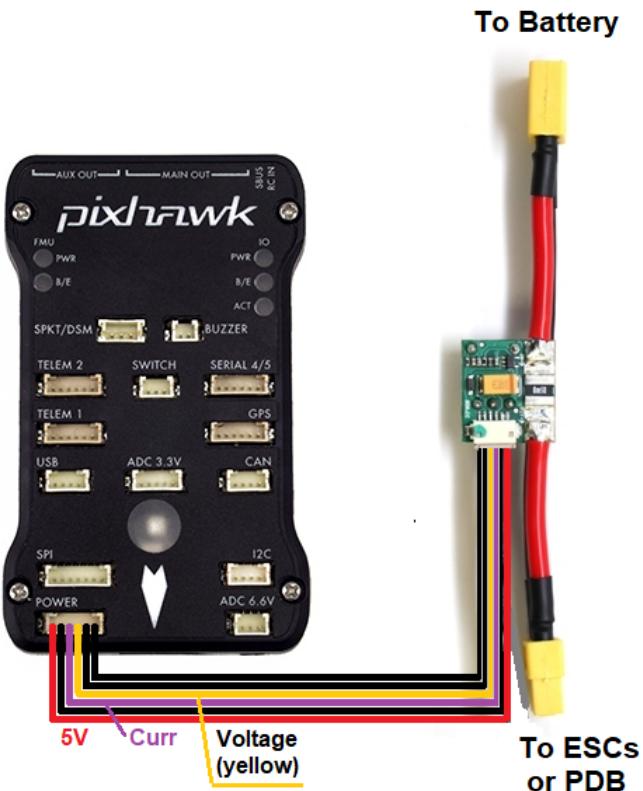
So Pixhawk was a better choice for our project as we will be processing images and we will use SLAM so we need a lot of processing power for our mission .

### 5.3.2 User guides to Pixhawk

- Step 1: Wiring and Connections

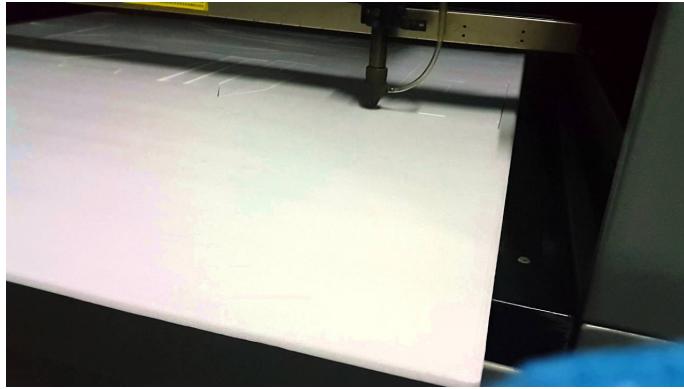
In order to get the Pixhawk running we will need the following components

- Power module
- Telemetry
- PPM
- Flight controller and receiver
- Power Distribution Board (PDB)
- Jumper wires
- Android USB cable
- **power module:** The power module is used in order to power the Pixhawk from the battery or from any power supply and to power the ESCs through the PDB , and it is connected to the Pixhawk as shown



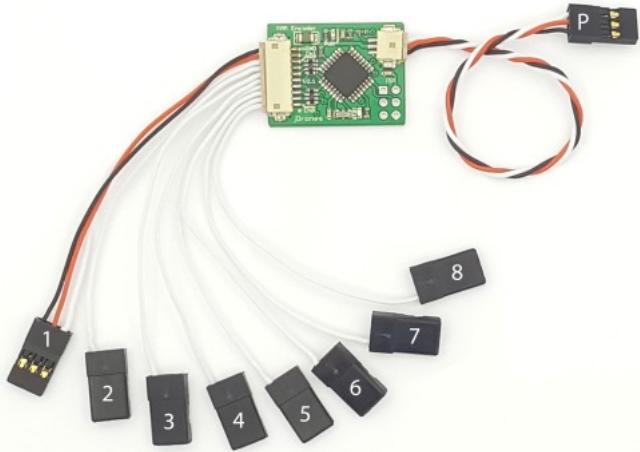
- **Telemetry:** The telemetry is divided into two parts , the first part is connected to the ground station to receive and send signals to the other part which is onboard to receive and send signals from and to the ground station .

It is connected to the Pixhawk as shown



- Vcc-GND-Rx-Tx must be connected.
- RTS - CTS can be ignored.
- RX in Pixhawk is connected to Tx in receiver.
- TX in Pixhawk is connected to Rx in receiver.

**PPM:** It is a module used to connect the channels of flight controller receiver to the Pixhawk ,as the Pixhawk has only one input for all channels so we used it to do this mission to connect the flight controller receiver to the Pixhawk.

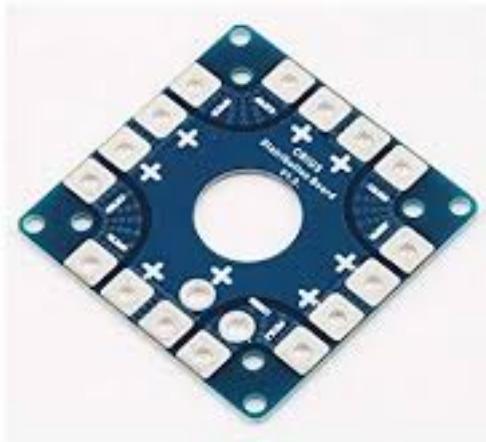


- **Flight controller and receiver:**

The flight controller is used to control the vehicle manually by sending PWM signals to the receiver which sends the signals to the PPM then the PPM fuses them to one channel and sends them to the Pixhawk to perform the control action sent.

- **Power Distribution Board (PDB):**

Is a board used to distribute the power that comes from the battery or power supply to the ESCs used.



- **Android USB cable:**

The cable will only be used during the Calibration of the gyros and ESCs.

- **Step 2: Connect to PC**

First connect the Pixhawk only to the PC using the usb cable.

- **Step 3: Ground Station**

Open Qgroundcontrol app to connect the Pixhawk to the ground station.

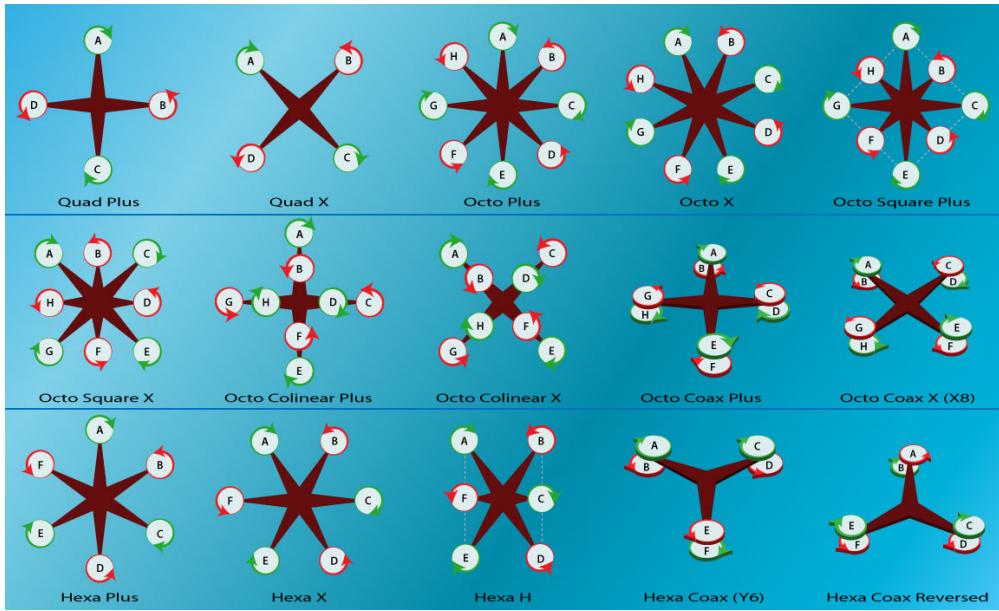
- **Step 4: Firmware selection**

After connecting the Pixhawk to the Qgroundcontrol we will need to install the PX4 firmware on the Pixhawk by selecting it from the setting menu, it will be downloaded and installed on the Pixhawk.

- note that before connecting the Pixhawk you need to remove any other power source to avoid spoiling the device.

- **Step 5: Frame Selection**

Next we will select the frame type of the vehicle you are using.



- Take care of the direction of the propellers (clockwise & anti clockwise).
- **Step 6: Calibration**

The next step is calibrating the gyros and compass of the Pixhawk just go from setting to calibration and follow the steps one by one and you will be done.

calibration can be done in two ways

- The first way is doing it on the Pixhawk alone.(recommended and easier)
- the second way is doing it after manufacturing the vehicle.

- **Step 7: Radio calibration**

In this phase we will need to connect the flight controller to the Pixhawk using the receiver and PPM encoder as shown before.

- Select radio calibration from setting menu.
- Set all trimming to zero before start the calibration.
- start the calibration and follow the steps given by the Qgroundcontrol.

- **Step 8: Modes**

- Select Flight modes from setting menu.
- Choose manual mode and set it to an AUX channel.
- Choose stabilize mode and set it to an AUX channel.

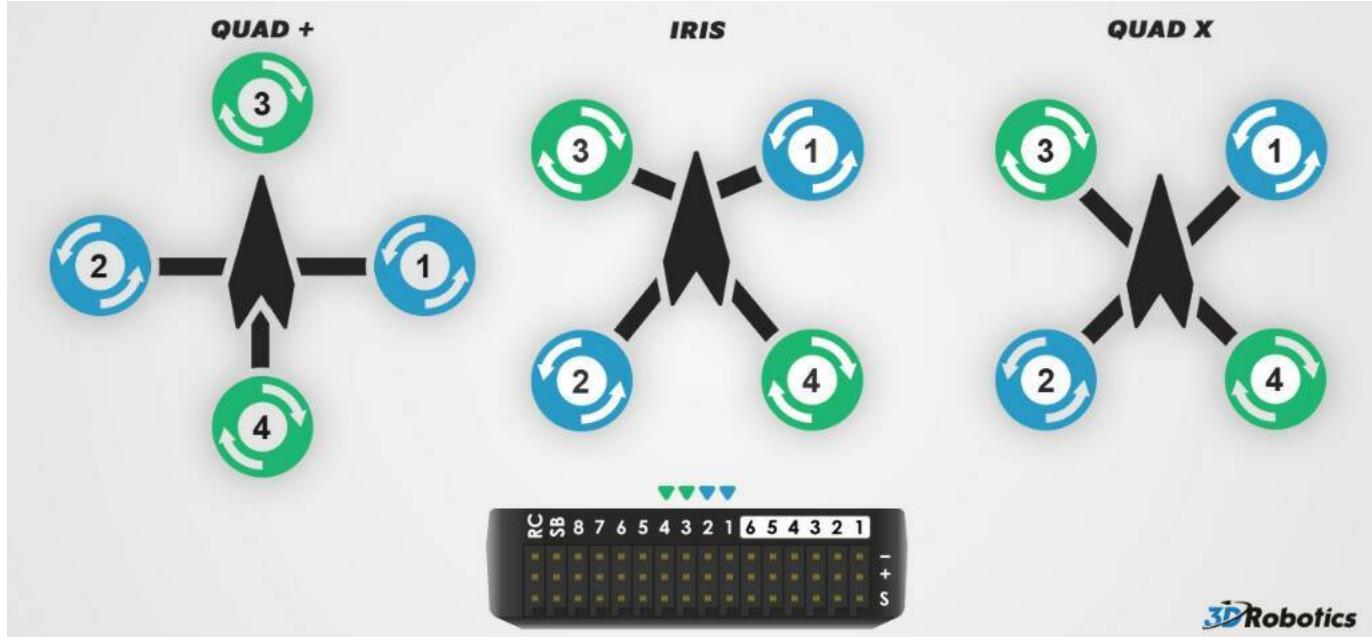
We will be using stabilize mode during testing and flight.

- **Step 9: ESCs and motors**

In this phase we will recommend that you follow the following steps exactly to avoid any damage to the Pixhawk.

- Remove any power source from the Pixhawk (USB till now).
- Connect the motors to the ESCs.
- Connect the ESCs to the PDB.

- Connect the PDB to the Power module (DO NOT CONNECT THE BATTERY OR POWER SUPPLY).
- Connect the signal pins of the ESCs to the Pixhawk output channels according to the order of the selected frame before.

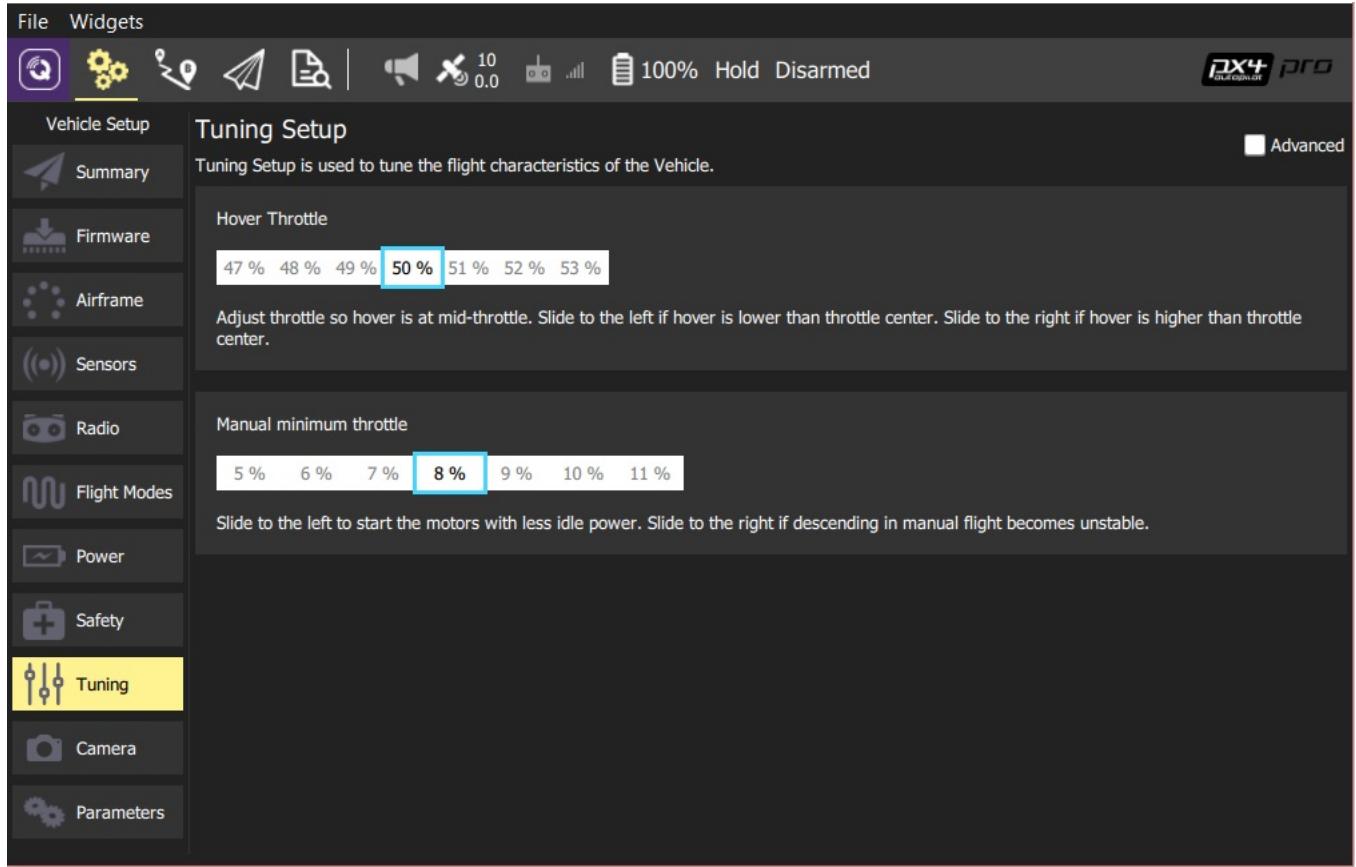


- From setting select Power.
- Start the calibration of the ESCs.
- You will be asked to connect the battery(power supply), note that this is the only case to connect two power sources together (USB,battery).
- The ESCs will make some sound and noise wait until its done(about 10-15 sec).
- Disconnect the battery then remove the USB cable .
- Connect the USB cable again and we are done with ESCs calibration.

- **Step 9: Tuning**

- Throttle Tuning: The first thing to do before tuning is to set your vehicle to stablize mode .

Now we should set the hovering thrust of the vehicle in order to make it mapped on the thrust lever of the flight controller.



In our case the Hover Throttle is set to 80% because the power supply is not strong enough to provide the ESCs with the power needed.

The Manual minimum throttle is set to 8%.

- The next step is going into the advanced PID tuning of the vehicle by clicking on “advanced”.

PID Tuning: Stabilize Roll/Pitch P controls the responsiveness of the copter’s roll and pitch to pilot input and errors between the desired and actual roll and pitch angles. The default of 4.5 will command a 4.5deg/sec rotation rate for each 1 degree of error in the angle. A higher gain such as 7 or 8 will allow you to have a more responsive copter and resist wind gusts more quickly.

- A low stabilize P will cause the copter to rotate very slowly and may cause the copter to feel unresponsive and could cause a crash if the wind disturbs it. Try lowering the RC\_Feel parameter before lowering Stability P if smoother flight is desired.
- Rate Roll/Pitch P, I and D terms control the output to the motors based on the desired rotation rate from the upper Stabilize (i.e. angular) controller. These terms are generally related to the power-to-weight ratio of the copter with more powerful copters requiring lower rate PID values. For example a copter with high thrust might have Rate Roll/Pitch P number of 0.08 while a lower thrust copter might use 0.18 or even higher.
  - Rate Roll/Pitch P is the single most important value to tune correctly for your copter.
  - The higher the P the higher the motor response to achieve the desired turn rate.
  - Default is P = 0.15 for standard Copter.
  - Rate Roll/Pitch I is used to compensate for outside forces that would make your copter not maintain the desired rate for a longer period of time
  - A high I term will ramp quickly to hold the desired rate, and will ramp down quickly to avoid overshoot.
  - Rate Roll/Pitch D is used to dampen the response of the copter to accelerations toward the desired set point.

- A high D can cause very unusual vibrations and a “memory” effect where the controls feel like they are slow or unresponsive. A properly mounted controller should allow a Rate D value of .011.
- Values as low as 0.001 and as high as .02 have all been used depending upon the vehicle.
- In our case after tuning and testing the final values set to the first quad copter prototype PID is as following for pitching and rolling and yawing.



## 5.4 Ground Station

### 5.4.1 Introduction

#### 5.4.1.1 What's a Ground Station

Ground Station is a fully managed service that lets you control the air vehicle communications, process data, and scale your operations without having to worry about building or managing your own ground station infrastructure. Drones are used for a wide variety of use cases, including surface imaging, communications, and video broadcasts. Ground stations form the core of communication between drone and user. These facilities provide communications between the ground and the vehicle in the air. Today, you must either build your own ground stations, or obtain long-term contracts with ground station providers.

#### 5.4.1.2 Ground Station software

There is a lot of ground station softwares available for users, in our project we are using two softwares which are

- Qgroundcontrol
- Mission Planner

### 5.4.2 User guides to Qgroundcontrol

Qgroundcontrol is a ground station software which is very simple to use and also is an OPEN SOURCE software so it's free to use and develop.

For beginners Qgroundcontrol is a very good software to start with as it is very simple and has a very friendly interface which will help you to get your vehicle moving and flying as fast as possible.

QGroundControl provides full flight control and mission planning for any MAVLink enabled drone. Its primary goal is ease of use for professional users and developers. All the code is open-source source, so you can contribute and evolve it as you want.

- Step 1: Installation

First of all, all you need to get started is to download the software from it's official website.

Then follow the installation step by step then you are done with installation and you are ready to start with the vehicle.

QgroundControl is available for windows,Mac,Android and IOS.

- Step 2: Connect to Pixhawk

The first thing to do after installation is to connect to your Vehicle.

You can do that through various methods

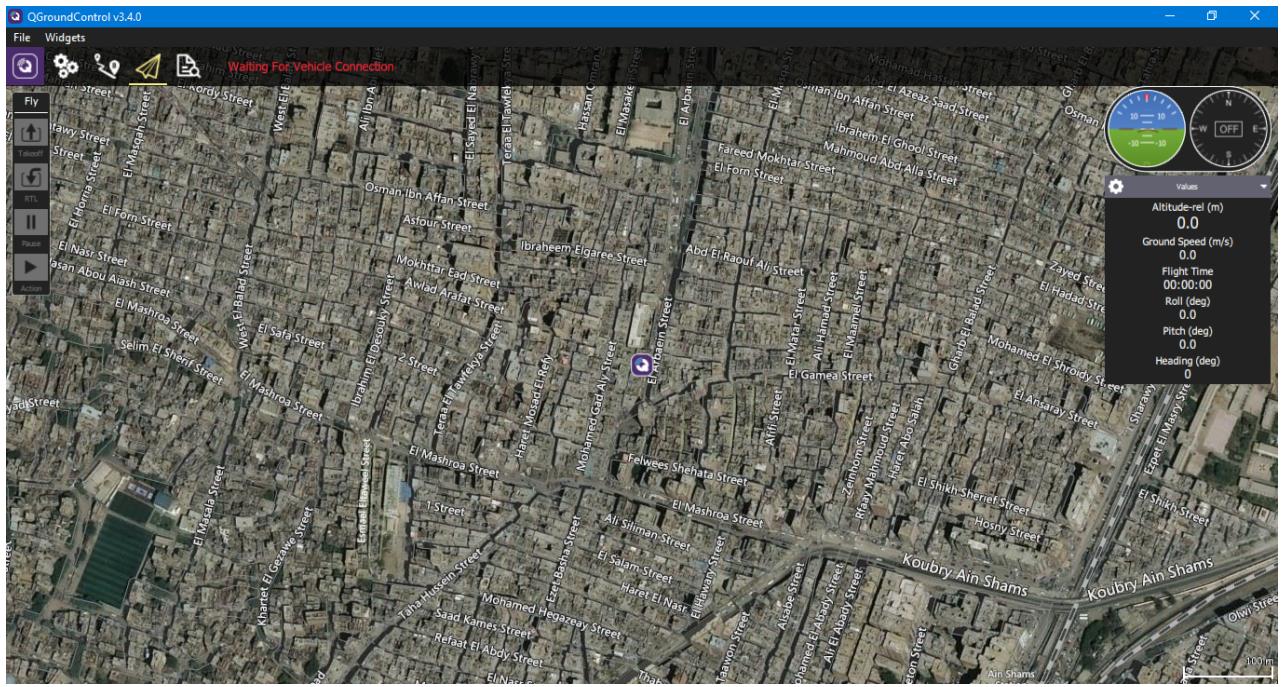
- USB cable
- Telemetry

At the beginning we will use USB.

### 5.4.3 User interface of Qground

Most of the user interface has been discussed in the previous chapter except for a few tabs.

Before you are connected to the Pixhawk the Interface you get is as shown



After you connect the Pixhawk you will have more tabs available for more options to access the autopilot control parameters.

#### 5.4.4 User guides to Mission Planner

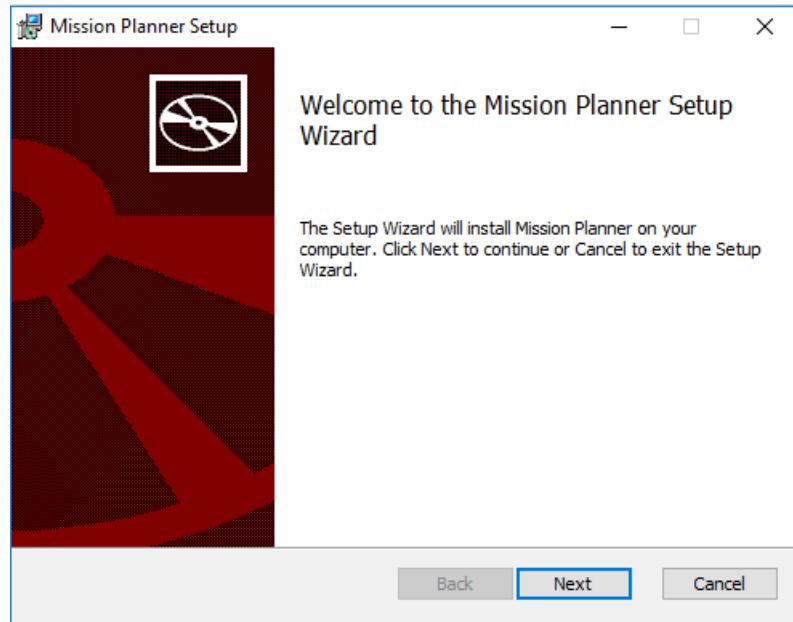
Mission Planner is a ground control station for Plane, Copter and Rover. It is compatible with Windows only. Mission Planner can be used as a configuration utility or as a dynamic control supplement for your autonomous vehicle. Here are just a few things you can do with Mission Planner:

- Load the firmware (the software) into the autopilot board (i.e. Pixhawk series) that controls your vehicle.
- Setup, configure, and tune your vehicle for optimum performance.
- Plan, save and load autonomous missions into your autopilot with simple point-and-click way-point entry on Google or other maps.
- Download and analyze mission logs created by your autopilot.
- Interface with a PC flight simulator to create a full hardware-in-the-loop UAV simulator.
- With appropriate telemetry hardware you can:
  - Monitor your vehicle's status while in operation.
  - Record telemetry logs which contain much more information than the on-board autopilot logs.
  - View and analyze the telemetry logs.
  - Operate your vehicle in FPV (first person view).

Mission Planner is also an OPEN SOURCE software.

- Step 1: Installation

The first thing to do is to download the software from its official site, and then follow the installation instruction step by step till you finish installation then you are ready to work with the Mission Planner as Ground station.

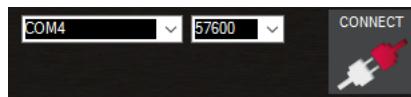


- Step 2: Connect to Pixhawk

Once you have installed a ground station on your computer, connect the flight controller using the micro USB cable as shown below.

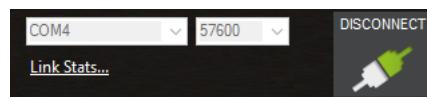


On Mission Planner, the connection and data rate are set up using the drop down boxes in the upper right portion of the screen.



Once you have attached the USB or Telemetry Radio, Windows will automatically assign your autopilot a COM port number, and that will show in the drop-down menu (the actual number does not matter). The appropriate data rate for the connection is also set (typically the USB connection data rate is 115200 and the radio connection rate is 57600).

Select the desired port and data rate and then press the Connect button to connect to the autopilot. After connecting Mission Planner will download parameters from the autopilot and the button will change to Disconnect as shown:



### 5.4.5 User interface of Mission Planner

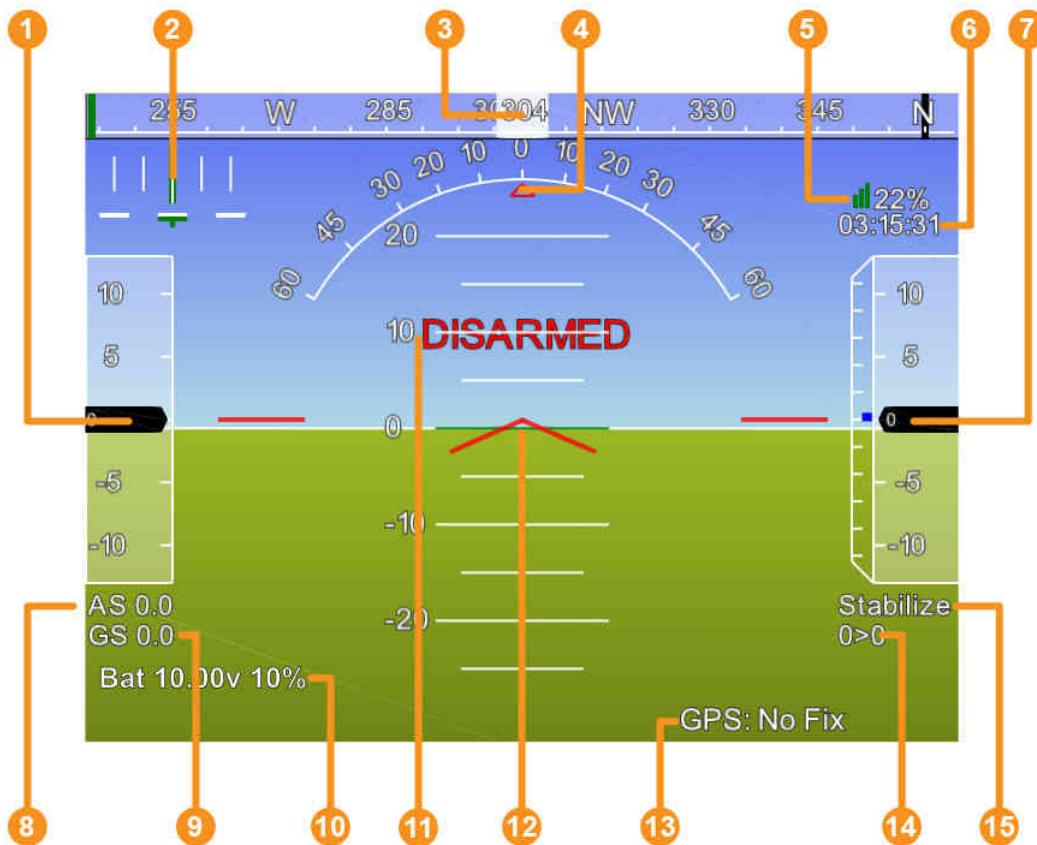
The sections are organized to match the major section of the Mission Planner as selected in the menu along the top of the Mission Planner window.



- Connect (Upper right corner) - How to connect the Mission Planner to your ArduPilot. Selecting communication devices and rates.
- Flight Data - Information about what you see, and things you can do in the Flight Data screens.
- Flight Plan - Information about the various aspects of preparing flight plans (Missions).
- Initial Setup - Information about what you see and things you can do in the Initial Setup screens.
- Configuration Tuning - Information about what you see and things you can do in the Configuration/Tuning screens.
- Simulation - How you can use the Mission Planner and a flight simulator to ‘simulate’ flying.
- Terminal - Information about what you see and things you can do in the Terminal screens.
- Help - About the help screen, and how to get help with your questions about Mission Planner.
- Other Mission Planner Features - Catch all for miscellaneous items.
- Flight Data Screen.
- Initial Setup.
- Simulation Screen.
- Configuration and Tuning Screen.
- Compass Calibration.
- Accelerometer Calibration.
- Radio Control Calibration.
- RC Transmitter Flight Mode Configuration.

- Flight Plan.
- Flight Data.
- Language Translations.
- Other Mission Planner Features





1. Air speed ( Ground speed if no airspeed sensor is fitted )
2. Crosstrack error and turn rate (T)
3. Heading direction
4. Bank angle
5. Telemetry connection link quality (averaged percentage of good packets)
6. GPS time
7. Altitude ( blue bar is rate of climb )
8. Air speed
9. Ground speed
10. Battery status
11. Artificial Horizon
12. Aircraft Attitude
13. GPS Status
14. Current Waypoint Number > Distance to Waypoint
15. Current Flight Mode

For more details you can find in documentation on the official website for both Qgroundcontrol and Mission Planner.

In our Project we used both Qgroundcontrol and Mission Planner in order to get the required mission, We will discuss how We have done that and how We managed to fly Our vehicle using both ground stations in the next chapter.

# References

# Appendix A

# ROS Examples

**A.1 publisher.py**

**A.2 subscriber.py**

**A.3 publisher.cpp**

**A.4 subscriber.cpp**

**A.5 ROS services using python**

**A.5.1 Server**

**A.5.2 node uses server**

**A.6 ROS actions using python**

**A.6.1 Action server**

**A.6.2 node uses action**