Autonomous Airtransportation using Vision Navigation

Submitted to:

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

By:

June 30, 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	*Robot Operating System*
SLAM	*Simultaneous Localization And Mapping*
SITL	*Software In The Loop*
HITL	*Hardware In The Loop*
LIDAR	*Light Detection And Ranging*
KF	*Kalman Filter*
EKF	*Extended Kalman Filter*
IMU	*Inertial Measurement Unit*
EOM	*Equation Of Motion*
LTI	*Linear Time Invarient*
PID	*Proportional-Integral-Derivative Control*
GPU	*Graphical Processing Unit*
ODD	*Operational Design Domain*
MC	*Multicopter*
AAV	*Autonomous Aerial Vehicle*
VTOL	*Vertical Take-Off and Landing*
ATC	*Air Traffic Control*
AP	*Autopilot*

Contents

[1] the study of forces and moments on system in motion
[2] the study of motion without regard to forces or moments

List of Figures

List of Tables

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.

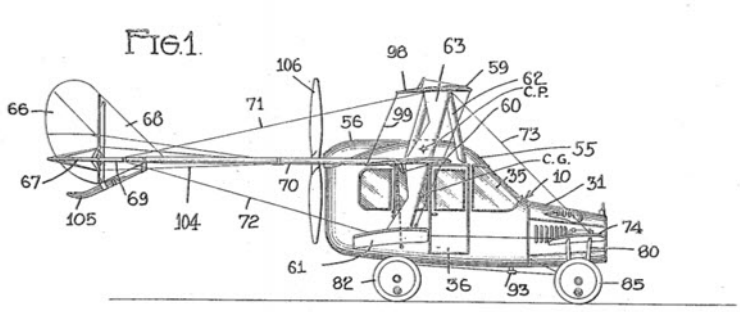


Figure 1.1.1: the Curtiss Autoplane

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 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[1] 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 since2017, 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.

[1]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

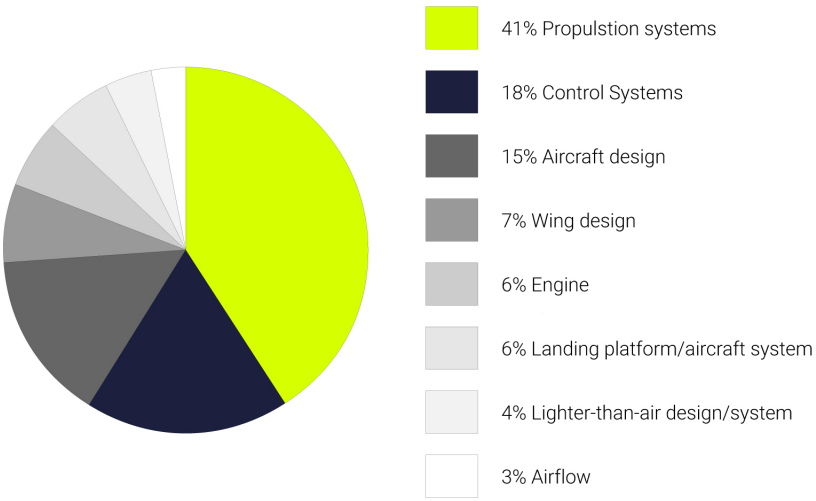


Figure 1.1.4: Technology Breakdown – patent filing activity

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 tiltable 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

In this chapter we will discuss the mission that we put in our consideration , So we can estimate the constraints and the initial weights, and this procedure lead us to start with the Aerodynamic design then go forward to the mechanical design.

2.1 Mission

The vehicles mission is to take the passenger from a certain location to a certain destination

2.1.1 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

2.1.1.1 Control payload

From the team working on control, the following payload is needed on the vehicle to be autonomously driven

1. Pixhawk
2. Pixhawk telemetry
3. Pixhawk power module
4. ZED stereo camera
5. Jetson nano
6. Li-PO battery

2.1.1.2 Passenger payload

comfortable chair

2.2 Aerodynamic Design

2.2.1 Introduction

When we started building the vehicle, we had some goals to achieve, and among of these goals is to get the highest performance and efficiency and to ensure that we must know the mission of the vehicle and its constraints:

- Max. takeoff weight about 3 Kg.
- Max. wide of vehicle 0.8 meter.
- Minimum size for easy landing at any place.

We choose XFLR5 program that is suitable platform to analyze this application because XFLR5 is a program to perform a plane analysis starting from airfoil design right to the stability of analysis of the complete plane and Define basic concepts of aerodynamics such as; lift, drag, Reylond's number and also things like viscosity, laminar flow, turbulence flow and transition for more flow to the other all things which are the core of the xflr5.

Finally we preferred to use **Tandem** wing plane to get greater lift than conventional wing, because we have two independent source of lift (two wings).

Basic definitions 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 (δ=α^{ω}-α^{p}): The relative angle of attack between two angles of attack for each wing.

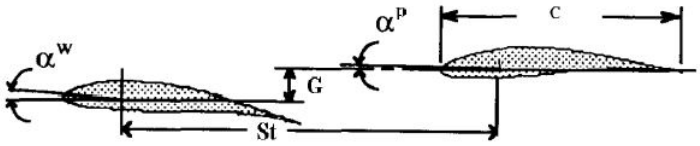


Figure 2.2.1: Tandem wing plane

2.2.2 Design Plan

Select the Airfoil of our plane that satisfied requirements; That the xflr5 program has a library of NACA series built in the program, but we can import another airfoils from UIUC Airfoil Data site that has all database of airfoils.

We make analyze for some airfoils that have high lift to carry the weight of vehicle; such as (Eppler421, Eppler423, MH14, NACA0018, NACA0015, Clark(Y) and NACA632615),Then we choose suitable airfoil dependent on: simplicity of manufacturing and lift generated.

So the most airfoils satisfy requirements are: Eppler423 and NACA0015,Then we now ready to analyse two airfoils and make it very smooth by re-panel foil from 100 to 150 panel to make distribution of panels is smoothly.

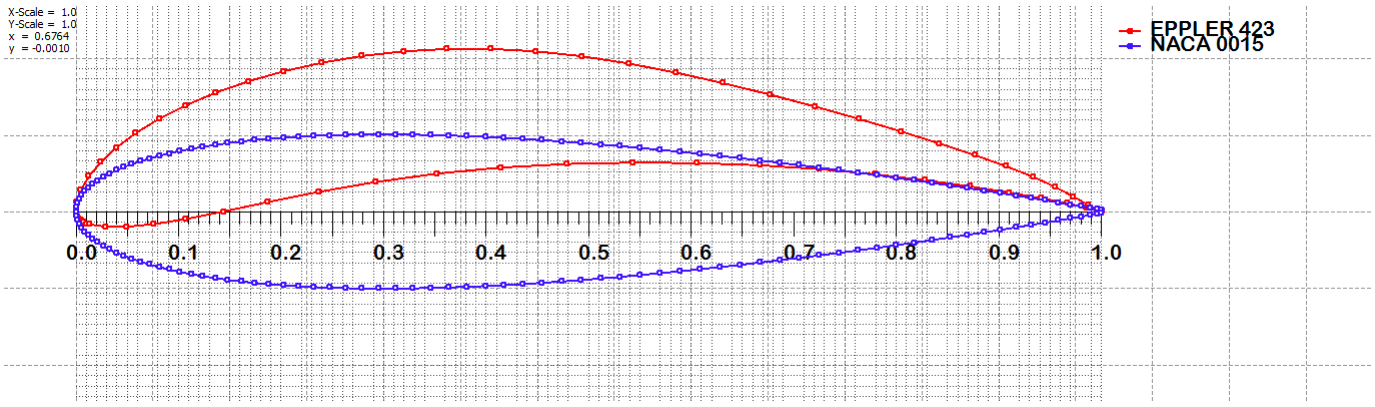


Figure 2.2.2: The 2 airfoils

- Then we can see the boundary layer and pressure distribution along the two airfoils in fig. below with change at angle of attack from: (0 to 10 & from 0 to -6) degree:

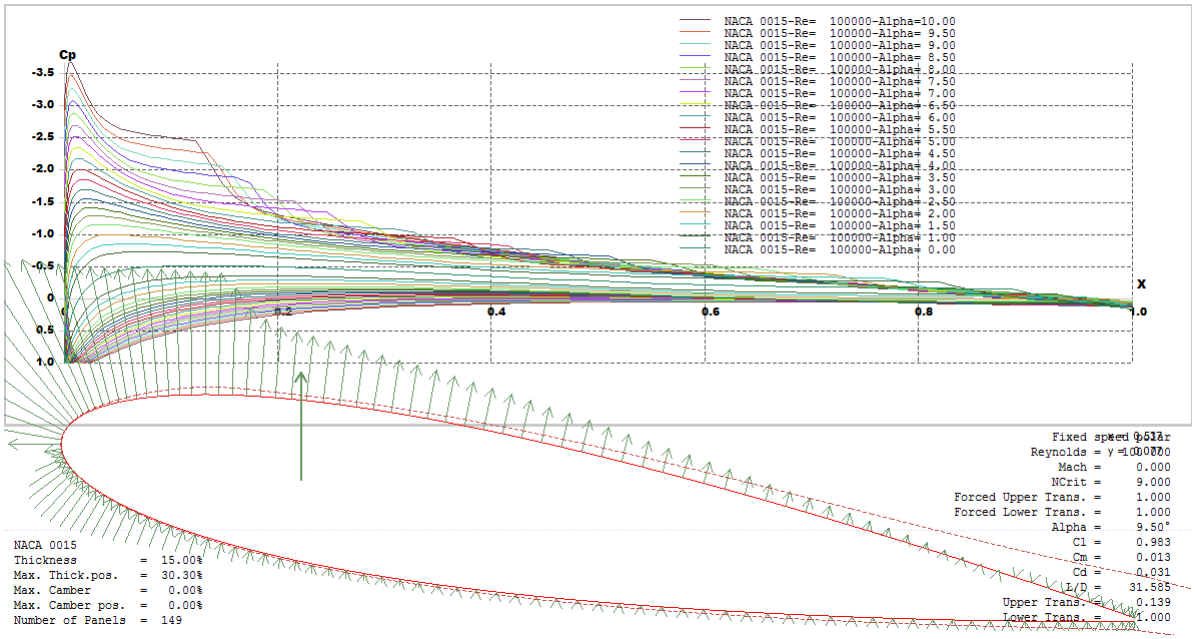


Figure 2.2.3: Pressure distribution along NACA0015 with difference angle of attack

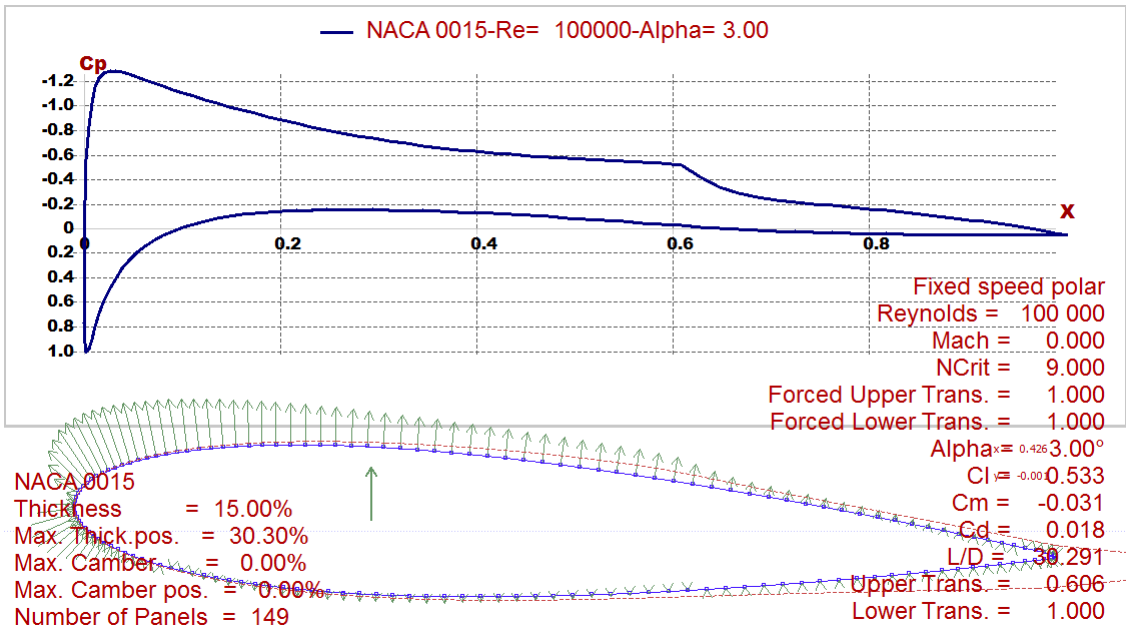


Figure 2.2.4: Pressure distribution along NACA0015 at certain angle of attack and Reylonds number

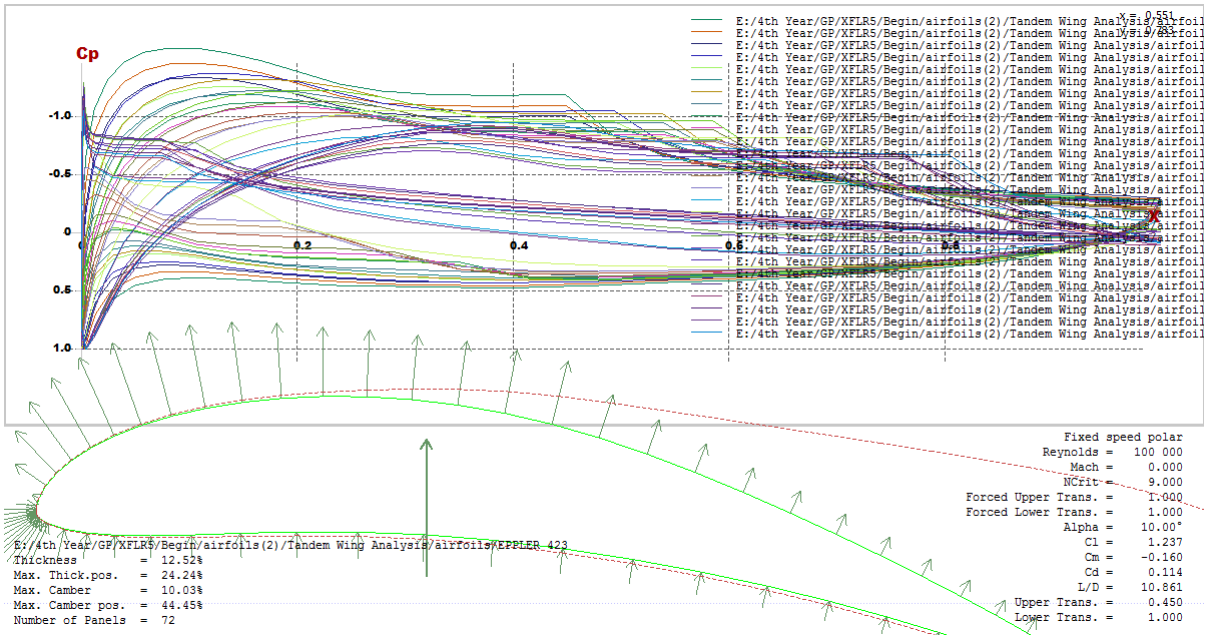


Figure 2.2.5: Pressure distribution along Eppler423 with difference angle of attack

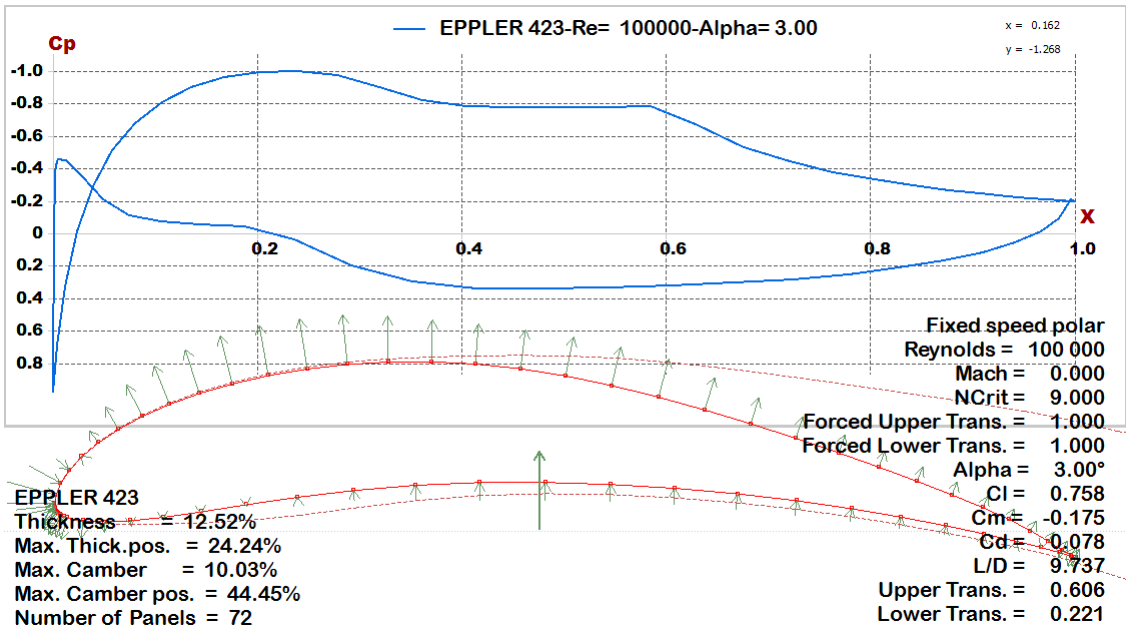


Figure 2.2.6: Pressure distribution along Eppler423 at certain angle of attack and Reylonds number

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

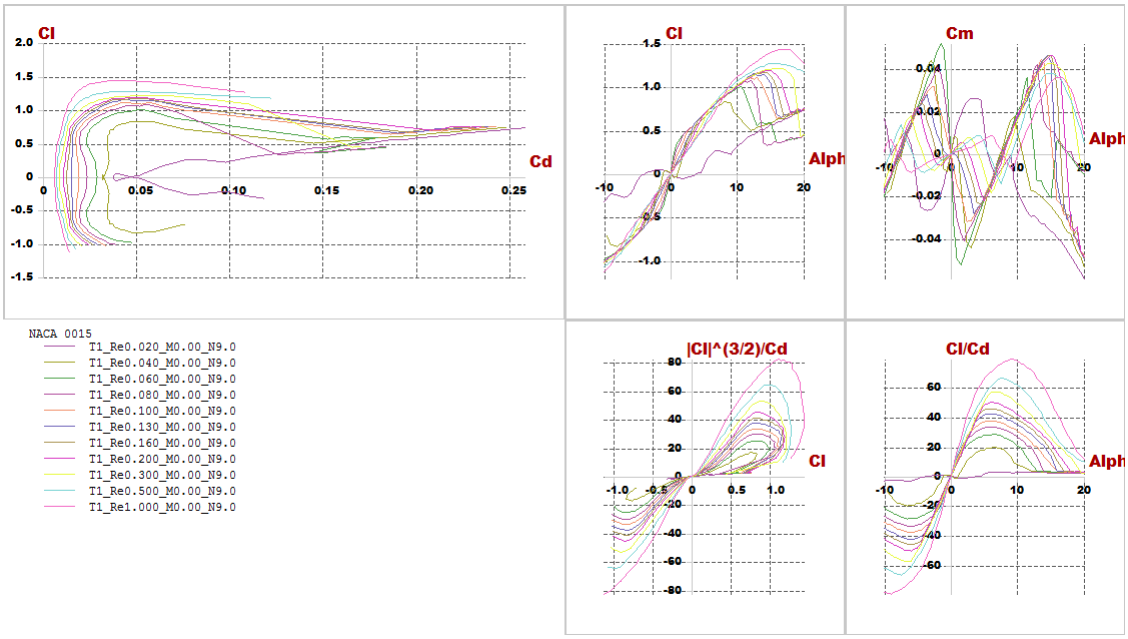


Figure 2.2.7: Polar graph of NACA0015

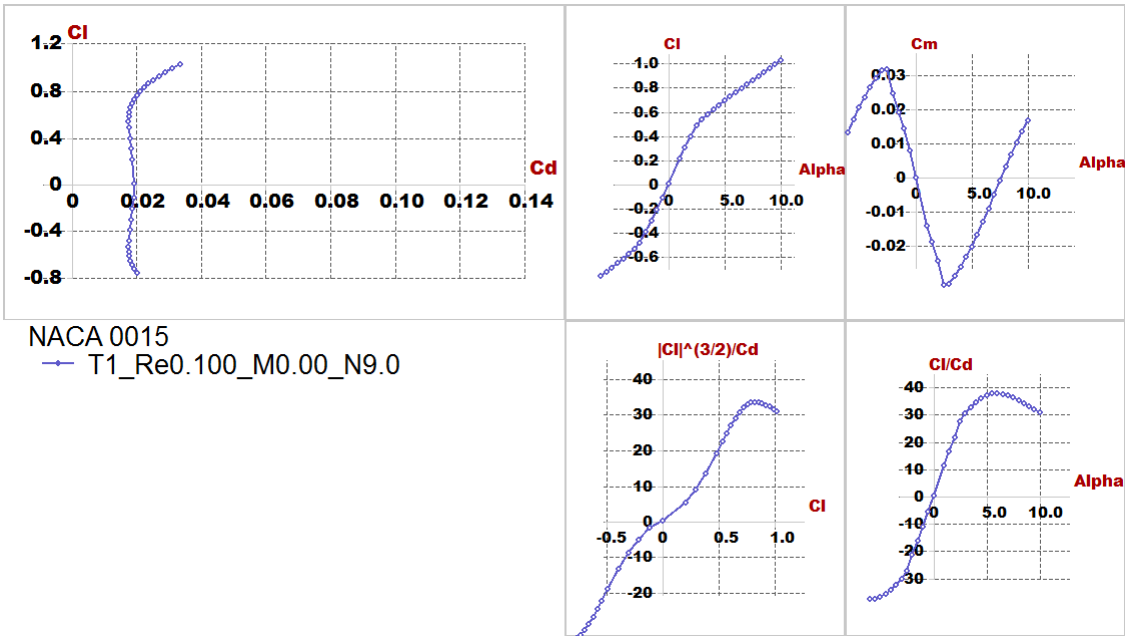


Figure 2.2.8: Polar graph of NACA0015 at a certain Reylonds number=100,000

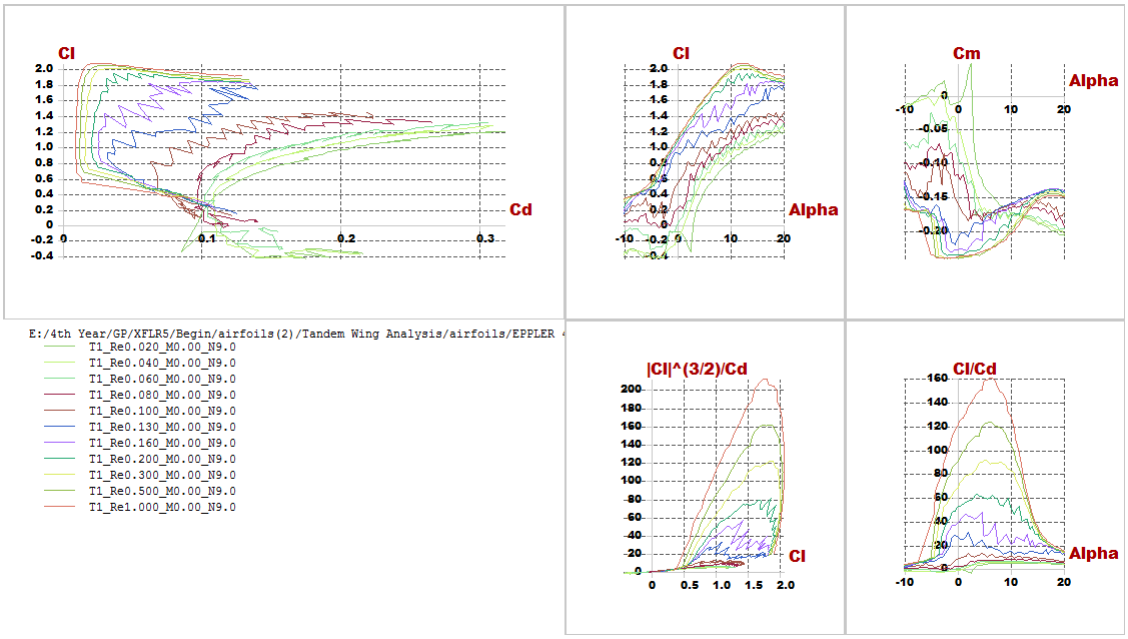


Figure 2.2.9: Polar graph of Eppler423

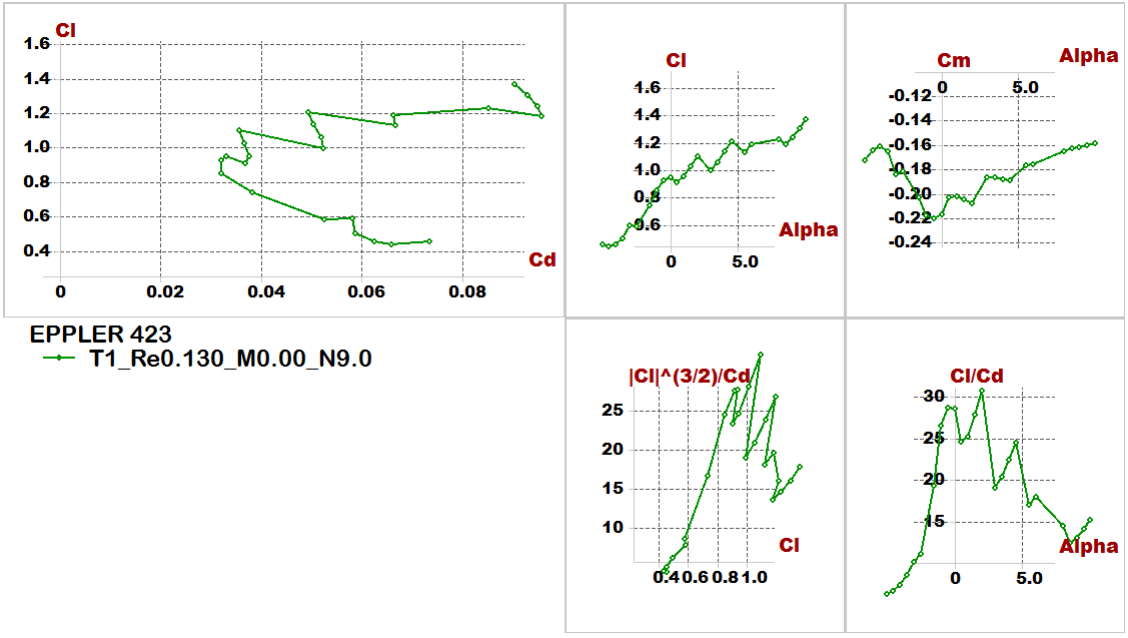


Figure 2.2.10: Polar graph of Eppler423 at a certain Reylonds number=130,000

Now we ready to build design after prepared airfoils, after trying different dimensions we get the best configuration then check stability and lift produced by vehicle we make analysis by different stream speed and found there isn't any effect on stability but affected at lift produced over Airfoil but the main reason affected on stability is the position of cg but not affected at another results shown in fig. below:

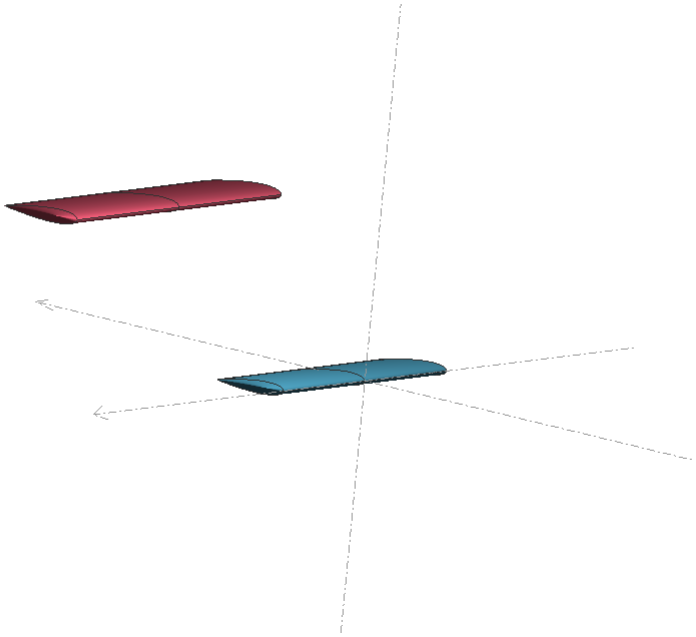


Figure 2.2.11: Configuration

- NACA0015

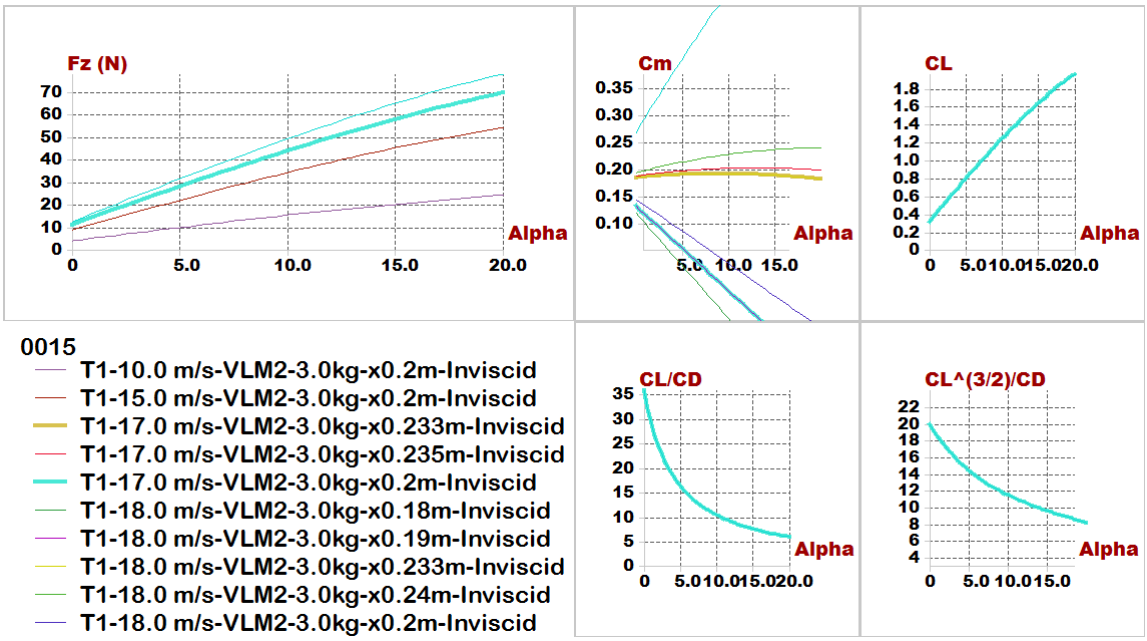


Figure 2.2.12: NACA0015 Lift and Cm-alpha

- Eppler423

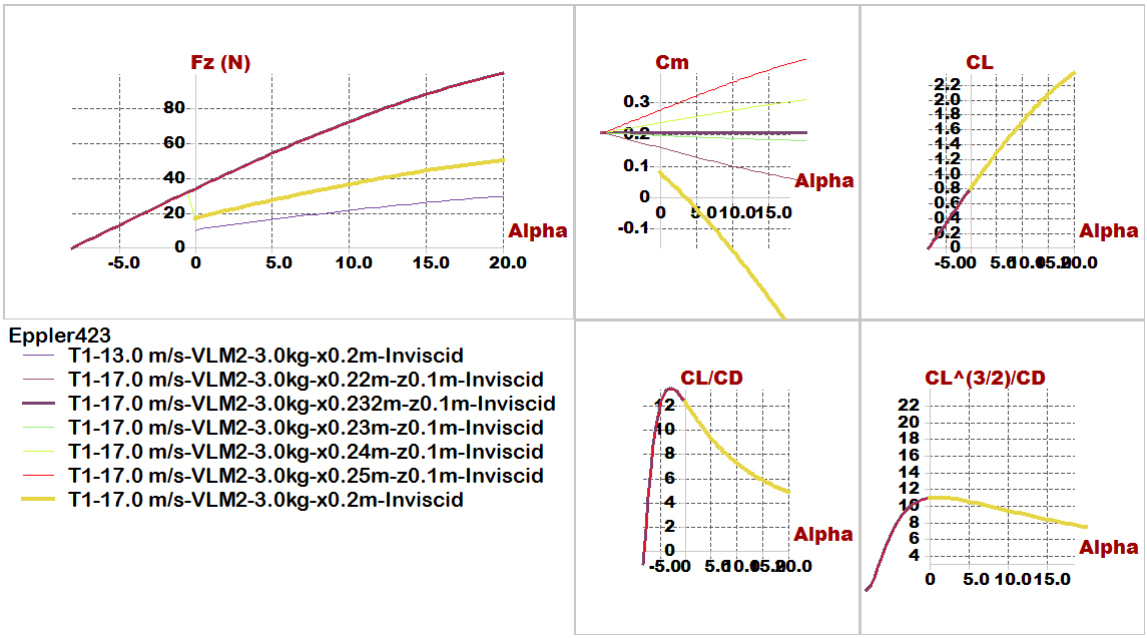


Figure 2.2.13: Eppler423 Lift and Cm

- **Quick comparison between 2 airfoils:**

Comparisons \ Models	Model 1 (NACA0015)	Model 2 (Eppler423)
Position of Main wing	x= 0 , z= -0.5 m	x= 0 , z= -0.05 m
Position of Second wing	x= 0.45 m , z= 0.2 m	x= 0.45 m , z= 0.15 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	4.5 degree	3.6 degree
$x_{C.G}$	0.19 m	0.2 m
$x_{N.P}$	0.243 m	0.25 m
Degree of Static Stability (S.M.)	21.2%	25%

Table 2.2.1: **comparison between 2 airfoils**

- Conclusion of results:

1. For configuration:
   - Before we illustration 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 noticeable 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 us to more stable quad-plane that increase stability due to wing spacing.

2. For incidence angle: All models we build it before almost of them have a negative incidence angle for tail not exceed -3 degree, So we highlight 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: We know the S.M. must be (from 5 to 20 %), and the S.M. in model (1) is 21.2 that may acceptable range, but at model (2) has 25%.

4. For Lift: We find trim angle of attack at 4.5 degree, and we go to Fz VS Alpha graph get lift that carry aircraft we get Fz=28(N) at 4.5 degree, which carry 2.85 Kg maximum tack-off weight, all of calculations reference to 2.7 Kg maximum tack-off weight.

So, we select model (1) because model (1) is more effective than model (2)„ next step we modified the design to satisfy position of C.G.

In this phase we tried to put point of C.G. in the middle of the vehicle to equalize the value of moments resulting from two wings to optimize the ($C_m - \alpha$) graph, First we make several trails with constraints of stability:

- We selected variables which allowed to change them.
- Then we set variables and change one of them, to convert slope of stability from positive to negative slope.
- We concluded variables that affect the slope:

1. Arm
2. The height of second wing
3. Incidence of angles of 2 wings
4. Geometry of 2 wings specially wing chord
5. Speed of vehicle

- Put position of C.G. and the arm, then change variables.

By trial and error after fail some trails we get the best configuration 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_{CG}	0.36 m
Z_{CG}	0.15 m
X_{NP}	0.383 m
Degree of Static Stability (S.M.)	11.5%

2.2.2.1 Final results

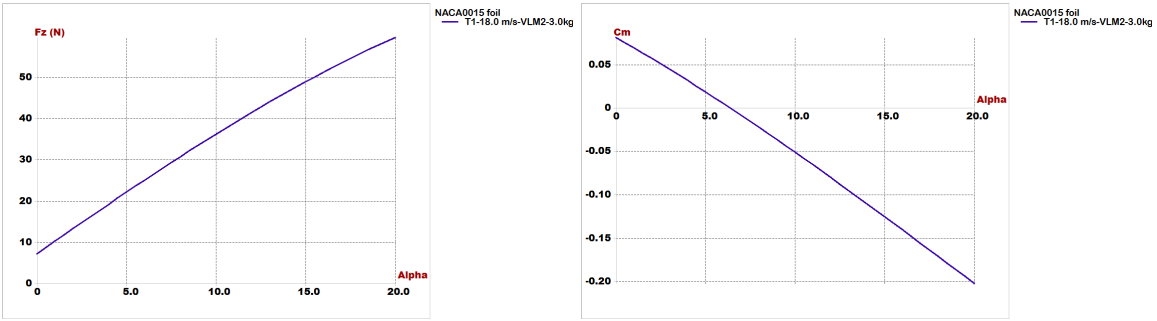


Figure 2.2.14: Lift Vs alpha and Cm Vs alpha

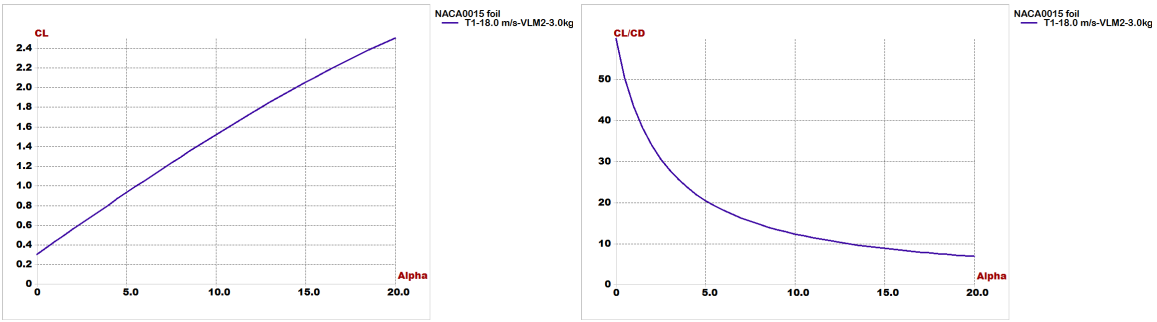


Figure 2.2.15: Cl Vs alpha and Cl/Cd Vs alpha

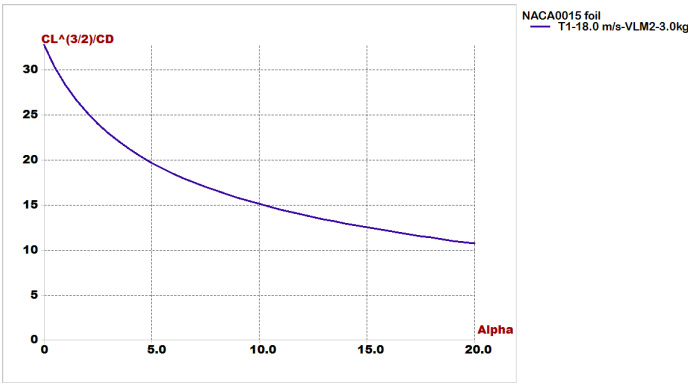


Figure 2.2.16: $Cl^{\frac{3}{2}}/Cd$ Vs alpha

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

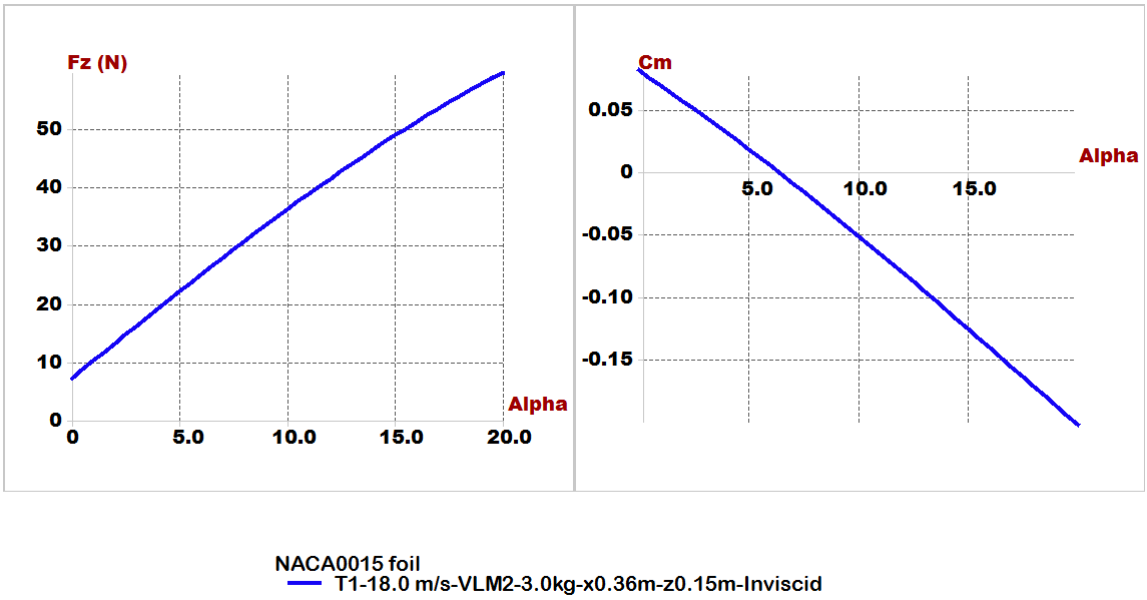


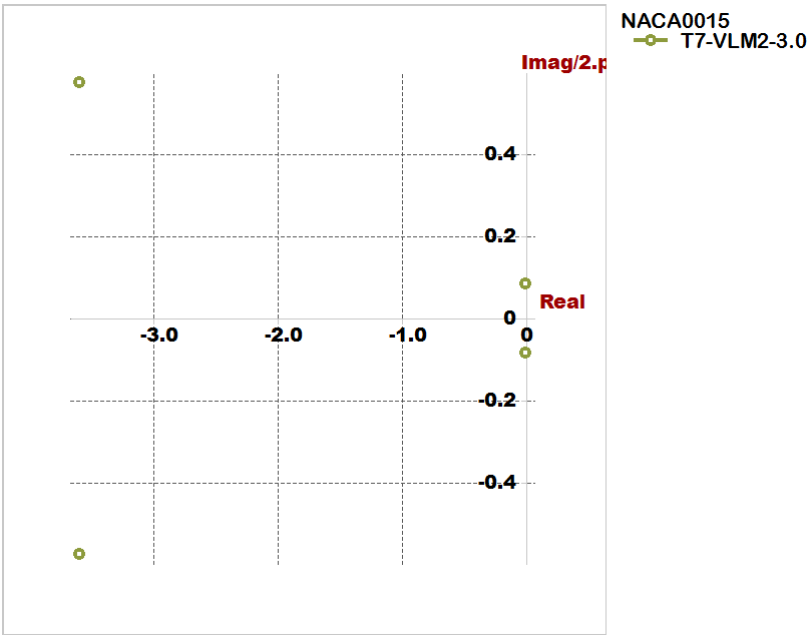
Figure 2.2.17: Fz vs alpha and Cm vs alpha

2.2.3 Stability Analysis

We will illustrate this part by video its shown modes of longitudinal and lateral stability of plane by root locus and time response

2.2.3.1 For longitudinal

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



___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 2.2.18: zero pole map for Longitudinal mode

- **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	

- **Time response characteristics:**

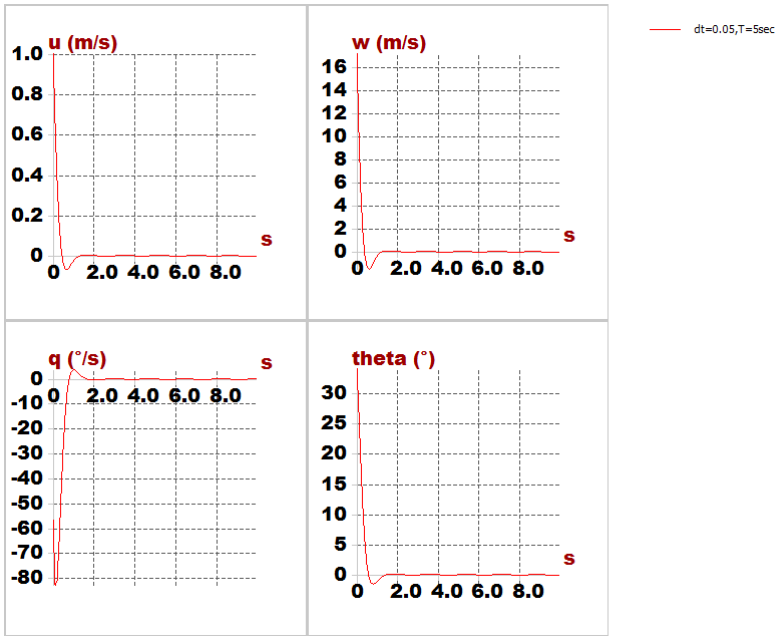


Figure 2.2.19: Short period mode

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.

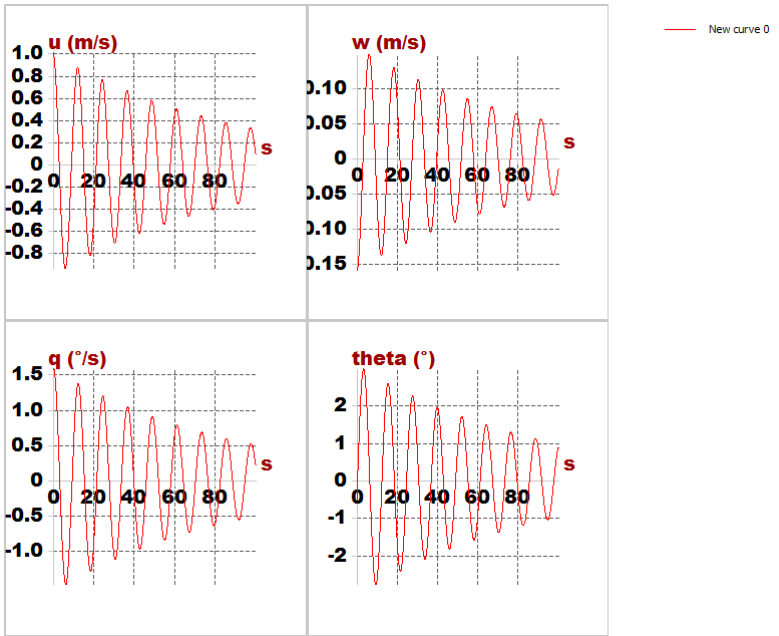
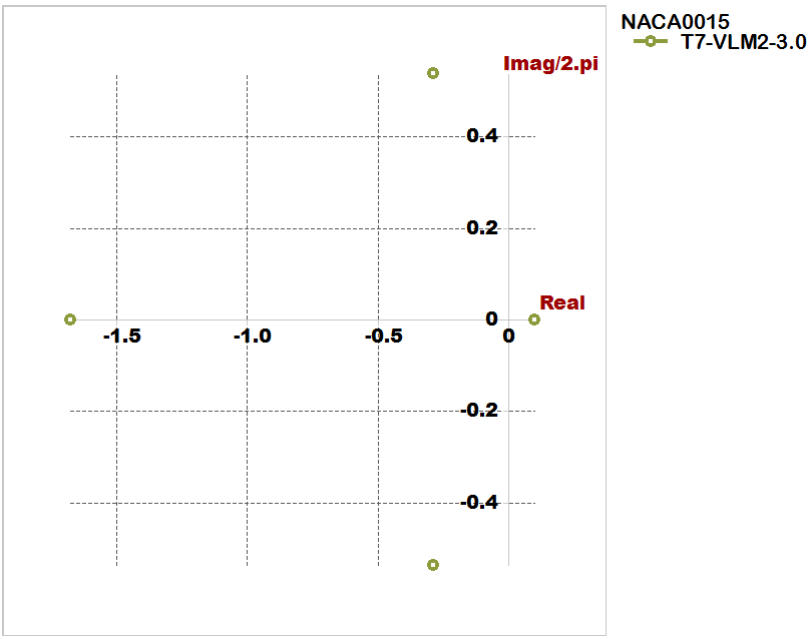


Figure 2.2.20: zero at very small time. Motion around steady flight mode

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

2.2.3.2 For lateral

- Type1: roll damping.
- Type2,3: dutch roll mode.
- Type4: spiral mode.



___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 2.2.21: zero pole map for Lateral 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

- **Time response characteristics**

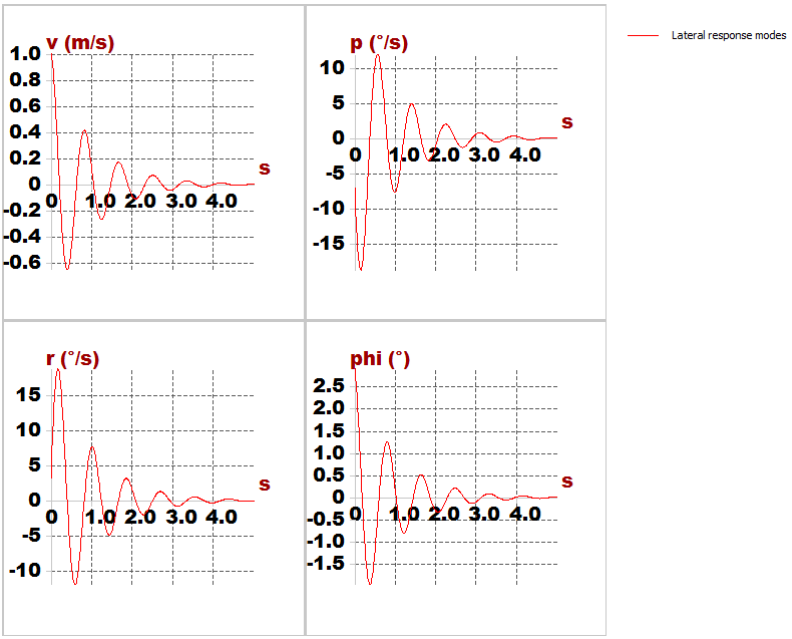


Figure 2.2.22: Lateral response modes

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

- **Non-dimensional Stability Derivatives calculated by program:**

CXu = -0.12599	CYb = -0.24520
CLu = 0.00057	Clb = -0.05277
Cmu = 0.00000	Cnb = 0.12824
CXa = 0.28216	CYp = -0.06327
CLa = 6.95194	Clp = -0.90835
Cma = -0.78024	Cnp = -0.15792
CXq = -0.24026	CYr = 0.29289
CLq = 13.60765	Clr = 0.38927
Cmq = -42.88605	Cnr = -0.15451

2.3 Mechanical Design

2.3.1 Introduction

In this chapter, we will discuss how to build and 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.

2.3.2 Building a 3D 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.

2.3.2.1 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)

2.3.2.2 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

2.3.2.3 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

2.3.3 Configurations

In this part various configurations will be discussed and a certain configuration will be chosen

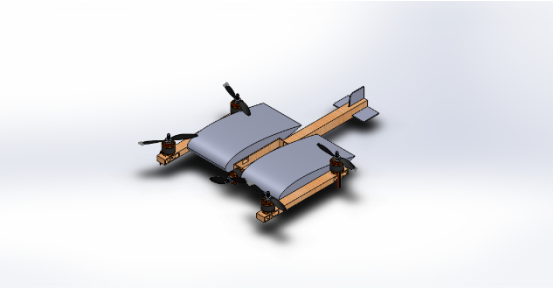


Figure 2.3.1: Conventional configuration

Figure 2.3.2: Flying wing configuration

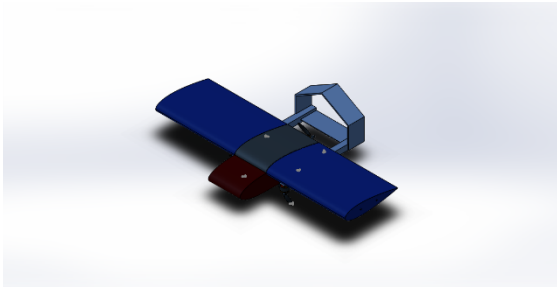


Figure 2.3.3: Pod and Boom Configuration

After deep research, Tandem wing configuration is found to be the most suitable configuration for a flying taxi vehicle where it needs to land and takeoff in streets between buildings so it needs to be with very small span, tandem wing configurations provide high left with small span

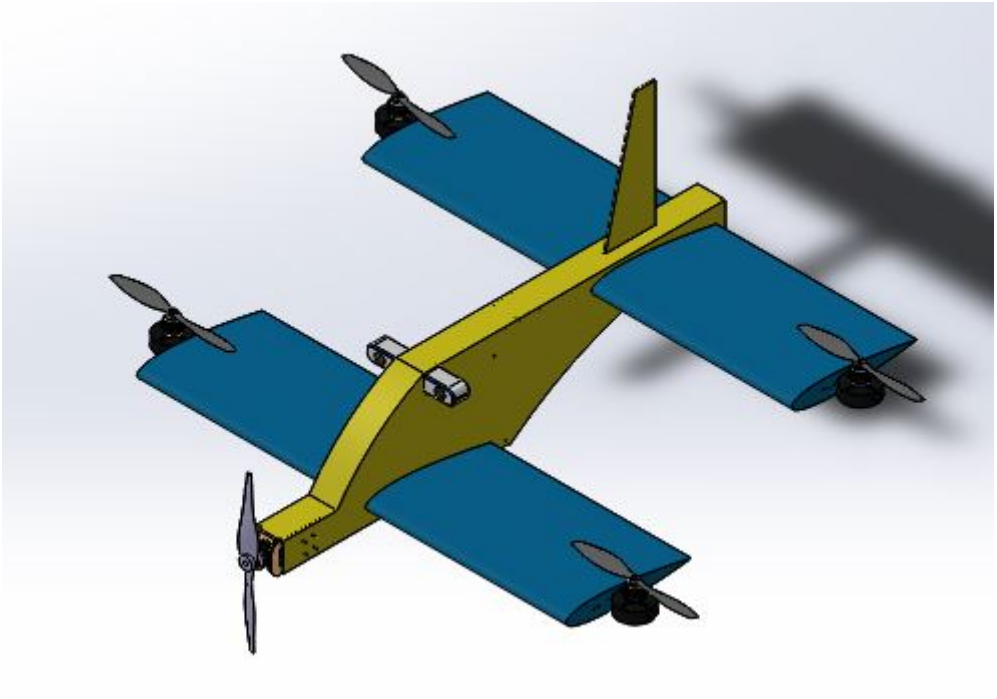


Figure 2.3.4: Tandem wing configuration

2.3.4 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

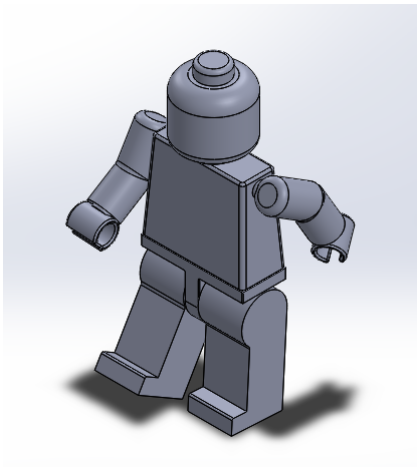


Figure 2.3.5: 1/1000 Scaled Passemger

- Chair

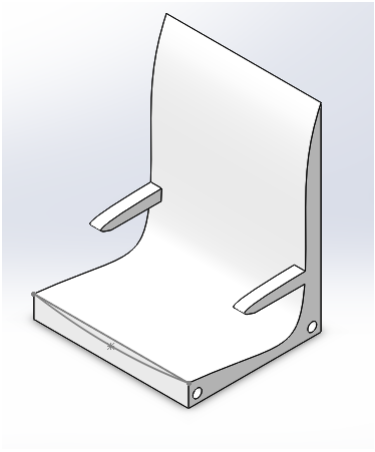


Figure 2.3.6: Chair

- Pixhawk

  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

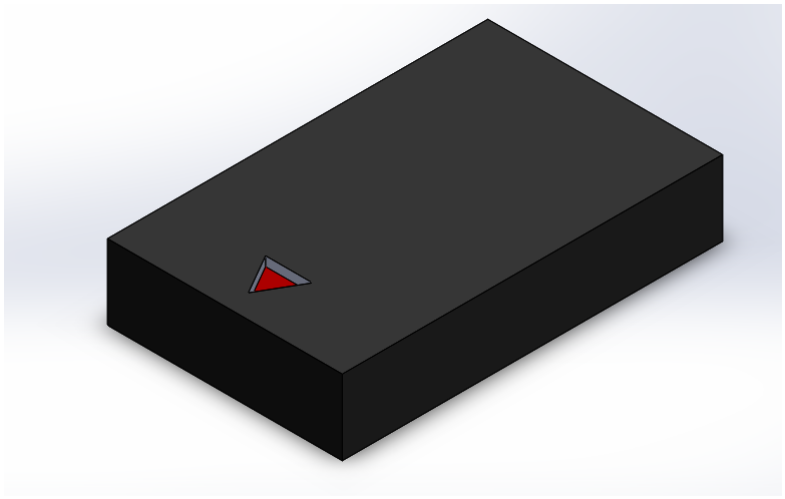


Figure 2.3.7: Pixhawk 3D model

- Pixhawk mount

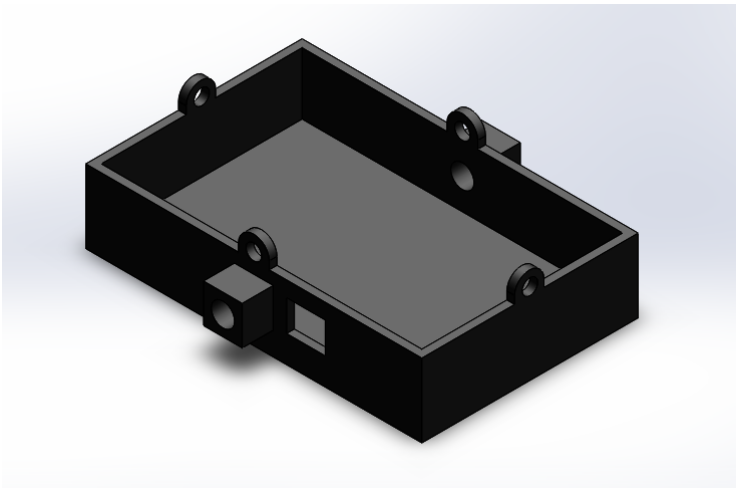


Figure 2.3.8: Pixhawk mount 3D model

- ZED camera

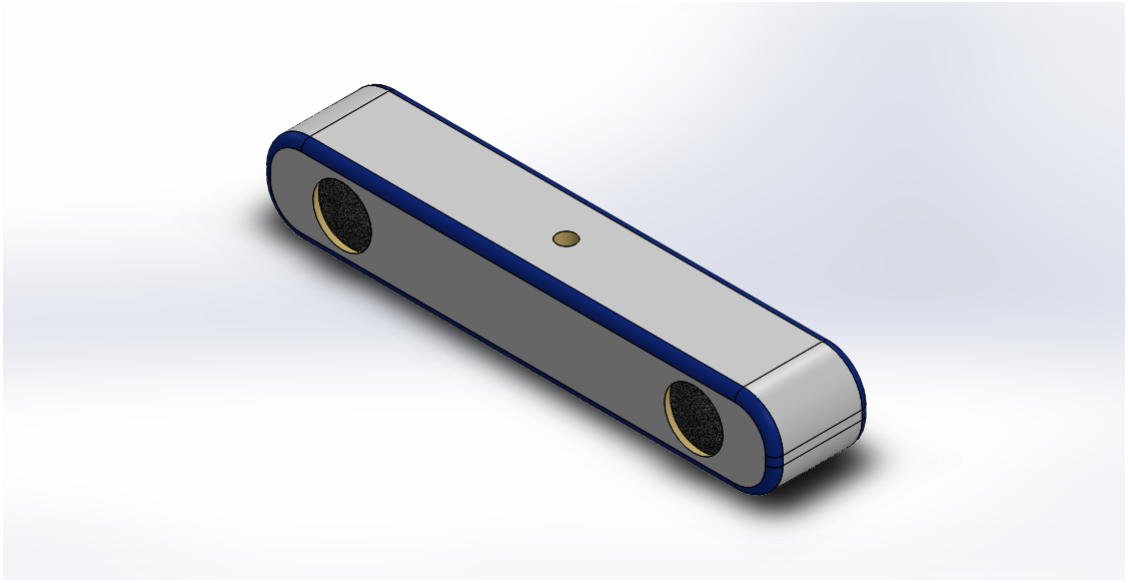


Figure 2.3.9: ZED stereo camera 3D model

- RP Lidar A1

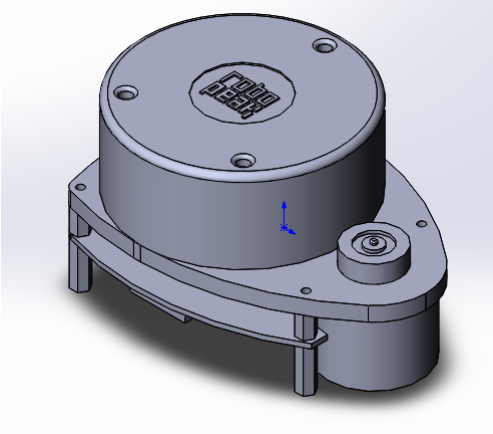


Figure 2.3.10: RP Lidar A1 3D model

- Jetson Nano KIT
- Quad Motor

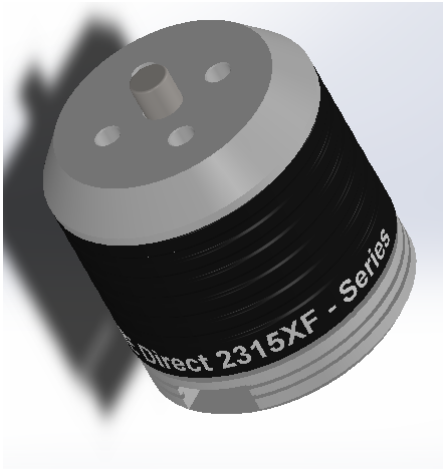


Figure 2.3.11: Hovering quad motor 3D model

- Pusher motor

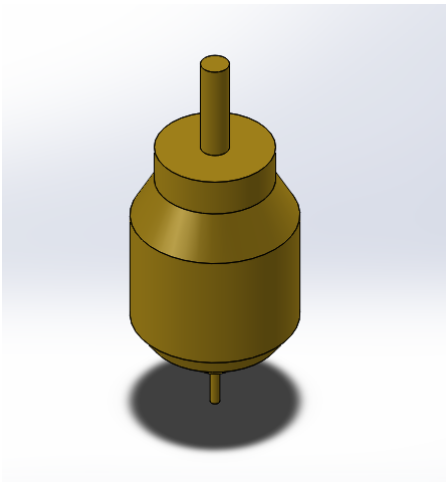


Figure 2.3.12: Pusher motor 3D model

- Battery

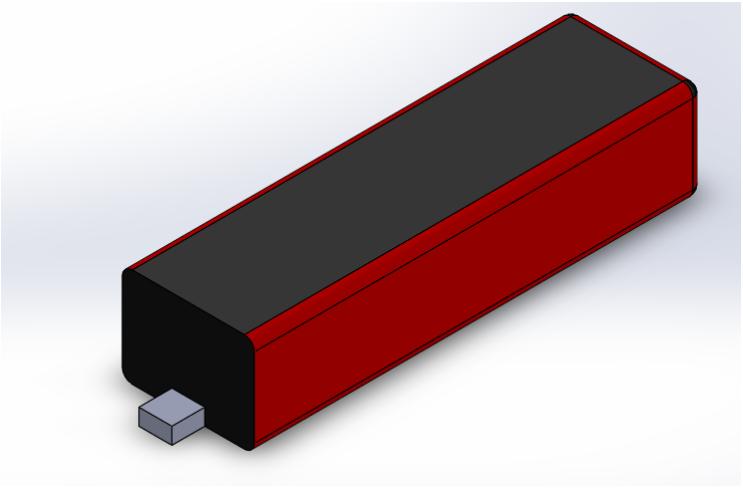


Figure 2.3.13: Battery 3D model

2.3.5 Wing fixation method

2.3.5.1 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:

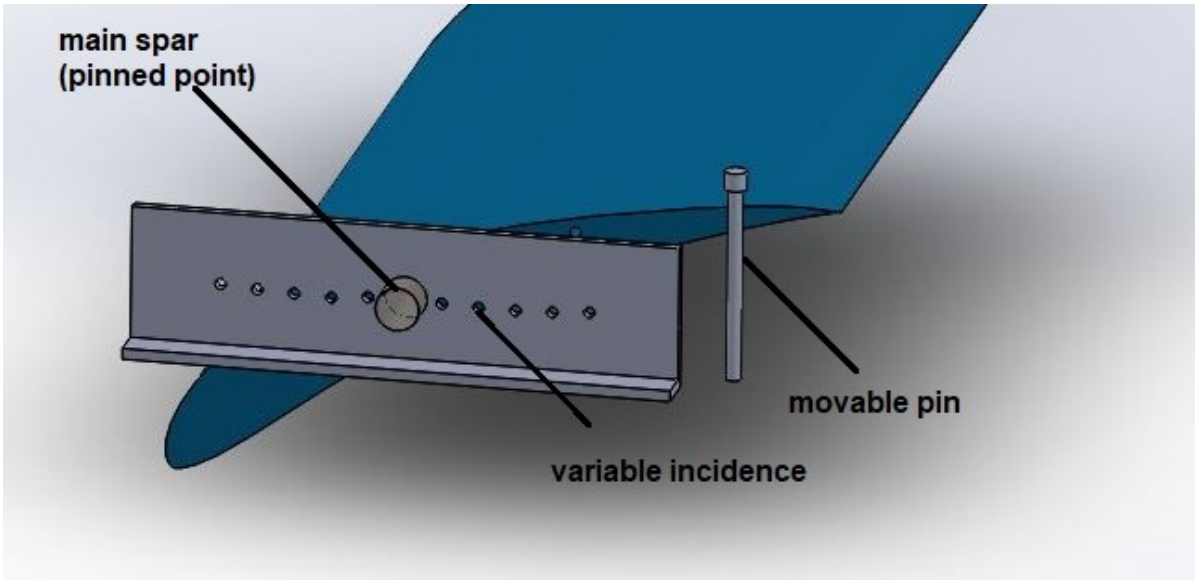


Figure 2.3.14: variable incidence fixation assembly

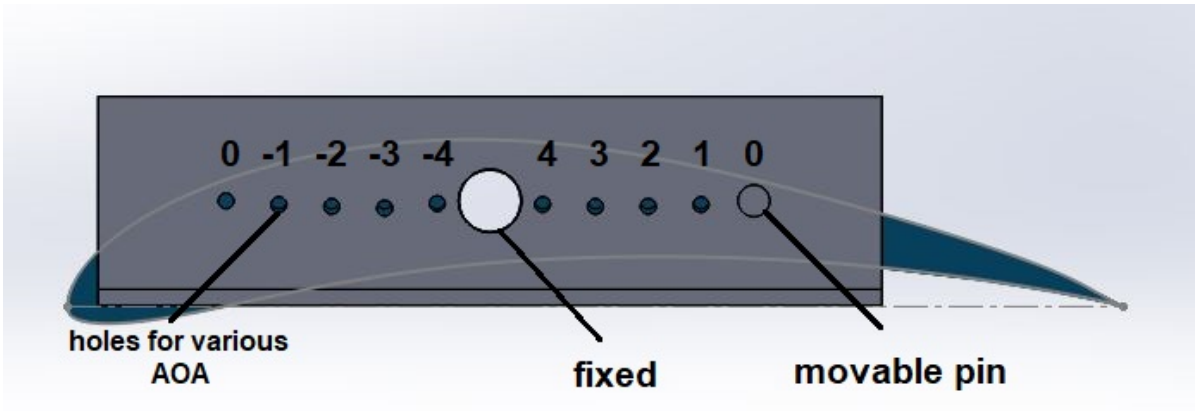


Figure 2.3.15: variable incidence fixation Front view

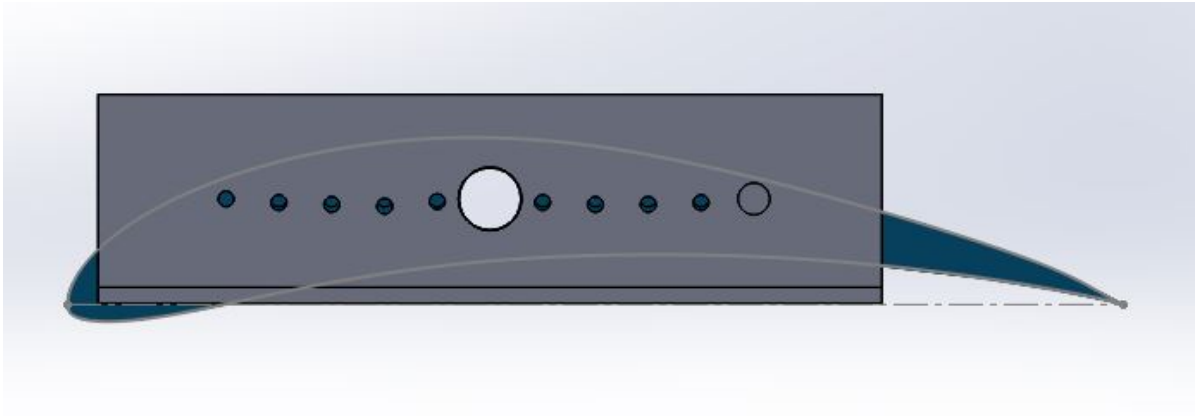


Figure 2.3.16: variable incidence fixation at zero incidence

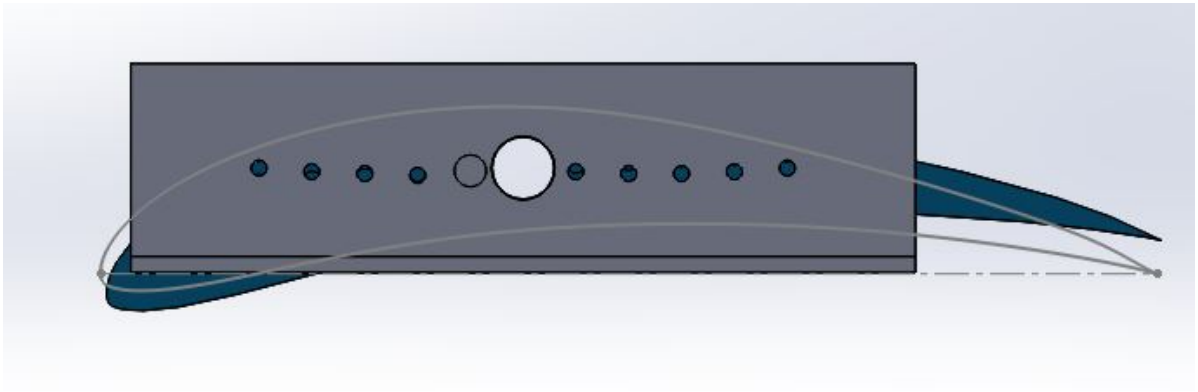


Figure 2.3.17: variable incidence fixation at -4 degrees incidence

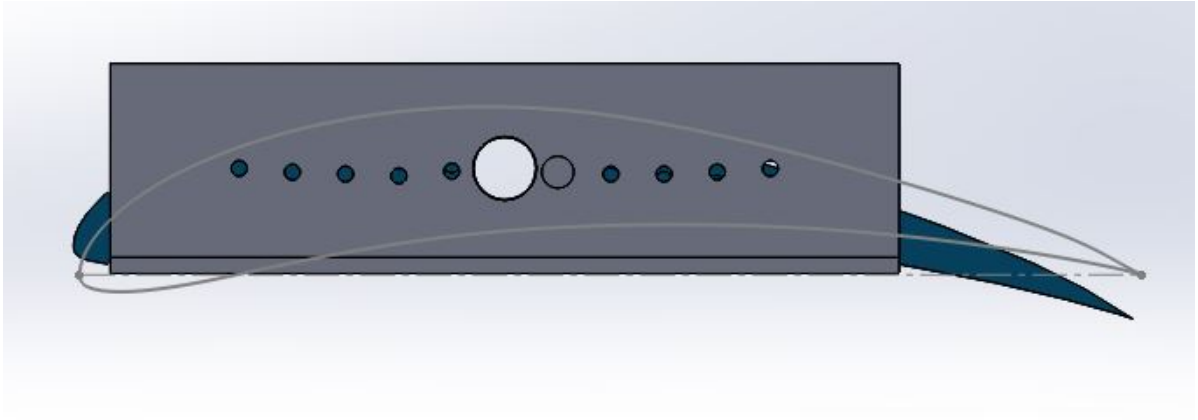


Figure 2.3.18: variable incidence fixation at 4 degrees incidence

2.3.5.2 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:

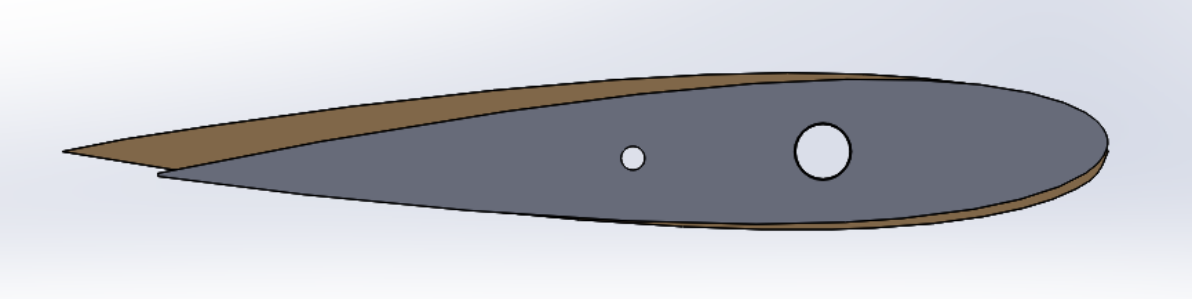


Figure 2.3.19: Constant incidence fixation at 2 degrees incidence

2.3.6 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

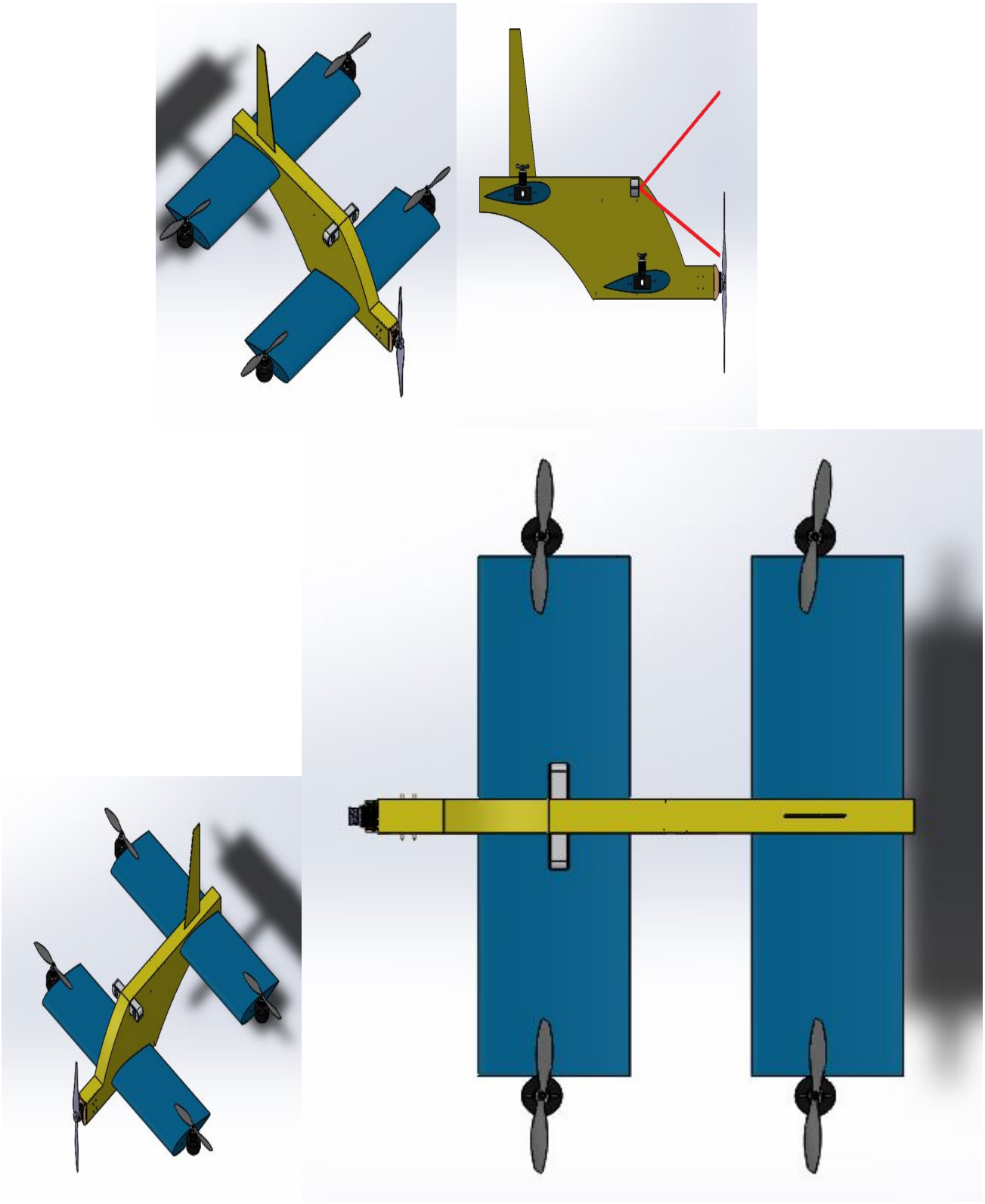


Figure 2.3.20: 1st mechanical design iteration

- Second iteration

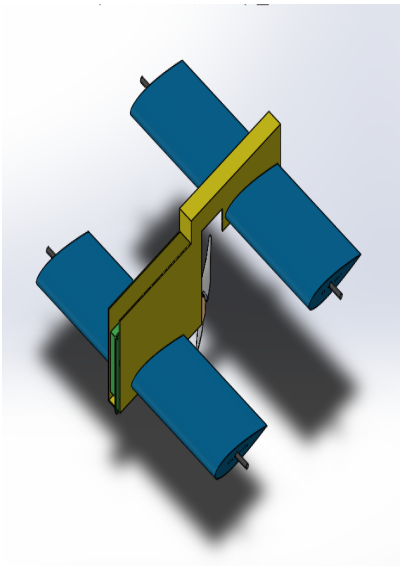


Figure 2.3.21: 2nd mechanical design iteration

- Third iteration

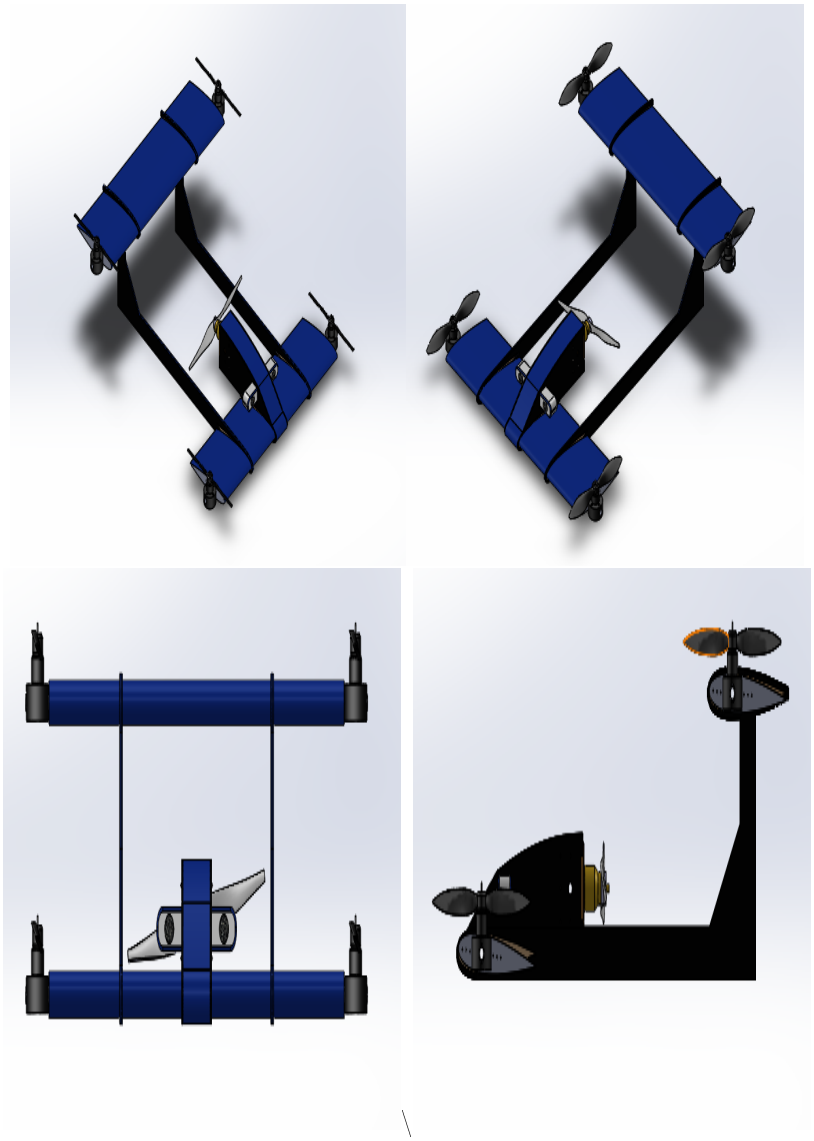


Figure 2.3.22: 3rd mechanical design iteration

- Fourth iteration

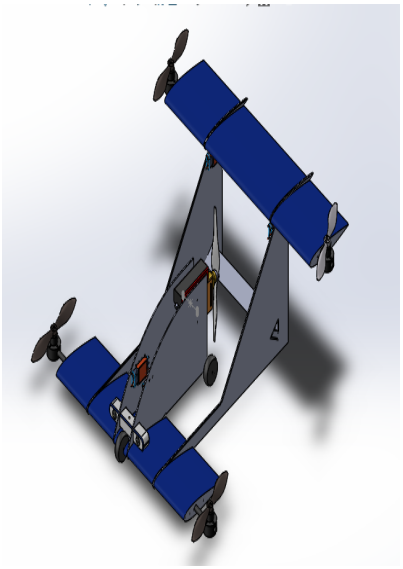


Figure 2.3.23: 4th mechanical design iteration

- Fifth iteration

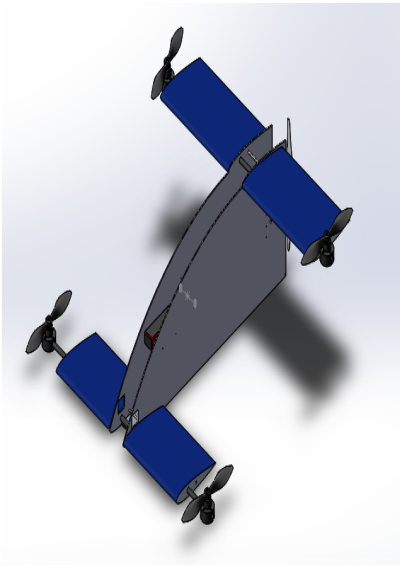


Figure 2.3.24: 5th mechanical design iteration

- Sixth iteration

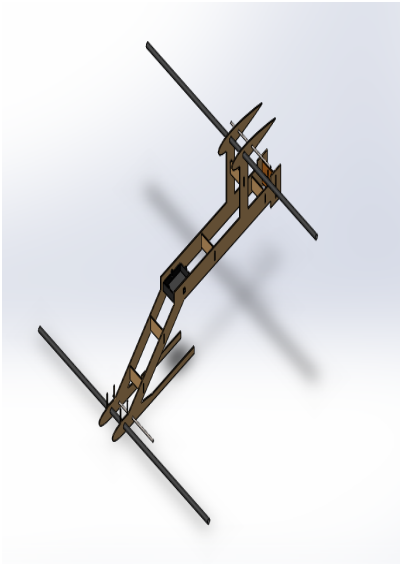


Figure 2.3.25: 6th mechanical design iteration

- Seventh iteration

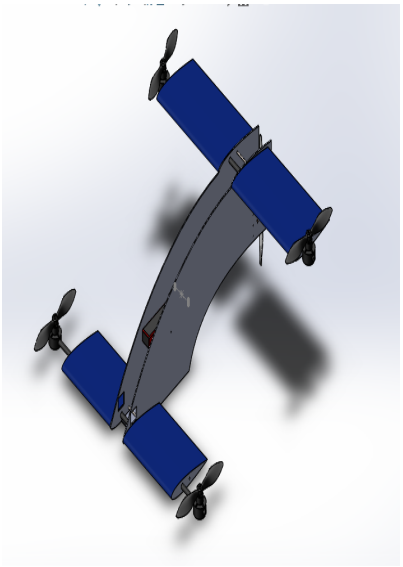


Figure 2.3.26: 7th mechanical design iteration

- Eighth iteration

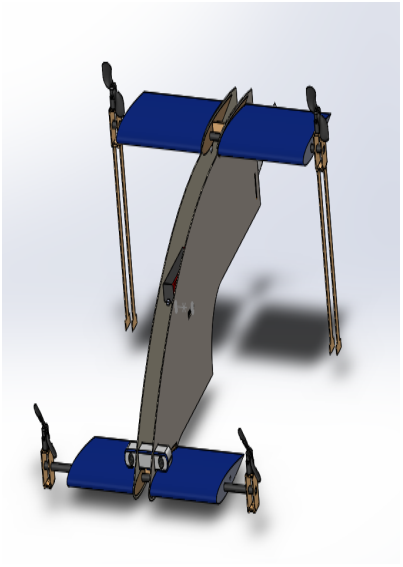


Figure 2.3.27: 8th mechanical design iteration

- Ninth iteration

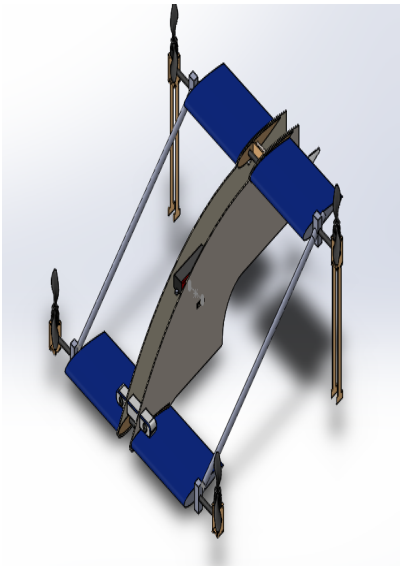


Figure 2.3.28: 9th mechanical design iteration

- Tenth iteration

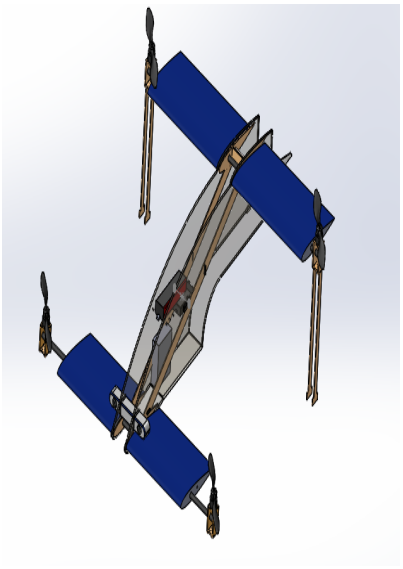


Figure 2.3.29: 10th mechanical design iteration

2.3.7 Propulsion system selection

2.3.7.1 Vertical takeoff propulsion system

- Motors

  the rule of thump 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 2.3.30: 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 2.3.1: Hover motor specifications

MOTOR VERSION	VOLTAGE LiHV [V]	PROPELLER SIZE	THROTTLE RANGE	AMPERAGE [A] (LOWER IS BETTER)	POWER INPUT [W] (LOWER IS BETTER)	[hp]	THRUST OUTPUT [g] (HIGHER IS BETTER)	[N]	[lb]	RPM [rev/min] (HIGHER IS BETTER)	EFFICIENCY [g/W] (HIGHER IS BETTER)	[lb/hp]
KDE2315XF-965 (965Kv) KDEXF-UAS35 S.R. ENABLED	11.6V (3S) 13.1V MAX	9" x 3.0	25.0%	0.6	6	0.01	100	0.98	0.22	3060	16.67	27.40
			37.5%	1.1	12	0.02	160	1.57	0.35	4620	13.33	21.92
			50.0%	2.1	24	0.03	270	2.65	0.60	6000	11.25	18.49
			62.5%	3.5	40	0.05	390	3.82	0.86	7260	9.75	16.03
			75.0%	5.4	62	0.08	520	5.10	1.15	8520	8.39	13.79
			87.5%	8.2	95	0.13	700	6.86	1.54	9780	7.37	12.11
			100.0%	11.0	127	0.17	850	8.34	1.87	10740	6.69	11.00
		9" x 4.5DJI	25.0%	0.7	7	0.01	120	1.18	0.26	2940	17.14	28.18
			37.5%	1.7	19	0.03	220	2.16	0.49	4260	11.58	19.04
			50.0%	3.4	39	0.05	370	3.63	0.82	5520	9.49	15.60
			62.5%	5.6	64	0.09	530	5.20	1.17	6680	8.28	13.61
			75.0%	8.7	100	0.13	720	7.06	1.59	7740	7.20	11.84
			87.5%	13.0	150	0.20	940	9.22	2.07	8820	6.27	10.30
			100.0%	16.9	195	0.26	1120	10.98	2.47	9680	5.74	9.44
		10" x 3.3	25.0%	0.6	6	0.01	110	1.08	0.24	3000	18.33	30.14
			37.5%	1.4	16	0.02	220	2.16	0.49	4440	13.75	22.60
			50.0%	2.7	31	0.04	340	3.33	0.75	5760	10.97	18.03
			62.5%	4.6	53	0.07	490	4.81	1.08	6980	9.25	15.20
			75.0%	7.2	83	0.11	680	6.67	1.50	8100	8.19	13.47
			87.5%	10.7	124	0.17	900	8.83	1.98	9300	7.26	11.93
			100.0%	14.1	163	0.22	1070	10.49	2.36	10140	6.56	10.79
		11" x 3.7	25.0%	0.8	9	0.01	150	1.47	0.33	2880	16.67	27.40
			37.5%	1.8	20	0.03	300	2.94	0.66	4140	15.00	24.66
			50.0%	3.7	42	0.06	470	4.61	1.04	5340	11.19	18.40
			62.5%	6.2	72	0.10	670	6.57	1.48	6480	9.31	15.30
			75.0%	9.7	111	0.15	890	8.73	1.96	7440	8.02	13.18
			87.5%	14.5	168	0.23	1150	11.28	2.54	8460	6.85	11.25
			100.0%	18.5	214	0.29	1340	13.14	2.95	9120	6.26	10.29
		12" x 4.0	25.0%	1.0	11	0.01	190	1.86	0.42	2700	17.27	28.40
			37.5%	2.2	26	0.03	340	3.33	0.75	3900	13.08	21.50
			50.0%	4.7	54	0.07	550	5.39	1.21	4920	10.19	16.74
			62.5%	8.1	93	0.12	790	7.75	1.74	5880	8.49	13.97
			75.0%	12.4	143	0.19	1020	10.00	2.25	6720	7.13	11.73
			87.5%	17.9	208	0.28	1260	12.36	2.78	7440	6.06	9.96
			100.0%	22.7	263	0.35	1450	14.22	3.20	8040	5.51	9.06
	15.4V (4S) 17.4V MAX	9" x 3.0	25.0%	0.7	10	0.01	130	1.27	0.29	4080	13.00	21.37
			37.5%	1.8	27	0.04	250	2.45	0.55	5940	9.26	15.22
			50.0%	3.4	52	0.07	430	4.22	0.95	7740	8.27	13.59
			62.5%	5.5	84	0.11	610	5.98	1.34	9180	7.26	11.94
			75.0%	8.5	130	0.17	840	8.24	1.85	10380	6.46	10.62
			87.5%	13.0	199	0.27	1130	11.08	2.49	12180	5.68	9.34
			100.0%	17.1	262	0.35	1360	13.34	3.00	13320	5.19	8.53
		9" x 4.5DJI	25.0%	1.0	15	0.02	180	1.77	0.40	3840	12.00	19.73
			37.5%	2.6	40	0.05	370	3.63	0.82	5520	9.25	15.21
			50.0%	4.9	75	0.10	580	5.69	1.28	6980	7.73	12.71
			62.5%	8.3	127	0.17	830	8.14	1.83	8280	6.54	10.74
			75.0%	12.7	195	0.26	1120	10.98	2.47	9480	5.74	9.44
			87.5%	18.6	286	0.38	1400	13.73	3.09	10620	4.90	8.05
			100.0%	24.5	376	0.50	1650	16.18	3.64	11700	4.39	7.21
		10" x 3.3	25.0%	0.9	13	0.02	190	1.86	0.42	3900	14.62	24.03
			37.5%	2.1	33	0.04	340	3.33	0.75	5640	10.30	16.94
			50.0%	4.1	63	0.08	540	5.30	1.19	7280	8.57	14.09
			62.5%	6.9	106	0.14	790	7.75	1.74	8780	7.45	12.25
			75.0%	10.7	165	0.22	1060	10.40	2.34	10080	6.42	10.56
			87.5%	15.8	243	0.33	1370	13.44	3.02	11480	5.64	9.27
			100.0%	20.9	321	0.43	1620	15.89	3.57	12360	5.05	8.30
		11" x 3.7	25.0%	1.1	16	0.02	250	2.45	0.55	3720	15.63	25.69
			37.5%	2.6	40	0.05	440	4.31	0.97	5220	11.00	18.08
			50.0%	5.6	85	0.11	720	7.06	1.59	6720	8.47	13.93
			62.5%	9.3	142	0.19	1010	9.90	2.23	7920	7.11	11.69
			75.0%	14.2	219	0.29	1330	13.04	2.93	9060	6.07	9.98
			87.5%	20.8	319	0.43	1650	16.18	3.64	10020	5.17	8.50
			100.0%	27.4	421	0.56	1960	19.22	4.32	11040	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** : KDE 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 2.3.31: Hover motor's preformance data

- ESCs
- Propellers set

2.3.7.2 Cruise propulsion system

- Motors

  According to many references, the rule of thump 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 2.3.2: power of motors to vehicle's weight

Figure 2.3.32: 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 2.3.3: Pusher motor specifications

- ESCs
- Propellers set

2.3.7.3 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 2.3.33: 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 2.3.4: Battery specification

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 [1]

- Translational motion: $\overrightarrow{F} = \frac{d}{dt}(m\overrightarrow{v}) = m(\mathring{\overrightarrow{v}} + \overrightarrow{\omega} \times \overrightarrow{v}) = \overrightarrow{F}_{Aero} + \overrightarrow{F}_{pusher} + \overrightarrow{F}_{motors} + \overrightarrow{g}$
  - 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 $\overrightarrow{v} = u\overrightarrow{i} + v\overrightarrow{j} + w\overrightarrow{k}$ but $\overrightarrow{i}$, $\overrightarrow{j}$, $\overrightarrow{k}$ are unit vector in body axes directions which change direction with time so the absolute change is the sum of change in magnitudes $\mathring{\overrightarrow{v}} = \mathring{u}\overrightarrow{i} + \mathring{v}\overrightarrow{j} + \mathring{w}\overrightarrow{k}$ and directions $\overrightarrow{\omega} \times \overrightarrow{v}$. 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.
  - $\overrightarrow{F}_{Aero}$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.
  - $\overrightarrow{F}_{pusher} = \begin{bmatrix} F_{pusher} \\ 0 \\ 0 \end{bmatrix}$ required to overcome the drag force to achieve required cruise speed.

[1] the study of forces and moments on system in motion

- $\overrightarrow{F}_{motors} = \sum_{i=1}^{4}[-\frac{1}{2}\rho AC(\omega_i R_{propeller})^2]\overrightarrow{k} = \sum_{i=1}^{4}[-b_i\omega_{prop,i}^2]\overrightarrow{k}$
  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 $\overrightarrow{F}_{motors}$
- $\overrightarrow{g} = C_{IB}\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$
  where C_{BI} is the rotation matrix from inertia axes to body axes according to ZYX euler angles(ψ, θ, ϕ) will be derived later in the kinematics study
  $\therefore \overrightarrow{g} = mg\begin{bmatrix} -sin(\theta) \\ cos(\theta)sin(\phi) \\ cos(\phi)cos(\theta) \end{bmatrix}$

- Rotational motion: $\overrightarrow{M} = \frac{d}{dt}(I\overrightarrow{\omega}) = I\overset{\circ}{\overrightarrow{\omega}} + \overrightarrow{\omega} \times I\overrightarrow{\omega} = \overrightarrow{M}_{motors}$
  -
  $$\overrightarrow{M}_{motors} = \overrightarrow{M}_{thrust\,induced} + \overrightarrow{M}_{drag\,induced}$$
  $$= \begin{bmatrix} l(T_3 + T_2 - T_1 - T_4) \\ l(T_1 + T_3 - T_2 - T_4) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -d_1\omega_1^2 - d_2\omega_2^2 + d_3\omega_3^2 + d_4\omega_4^2 \end{bmatrix} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix}$$

  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

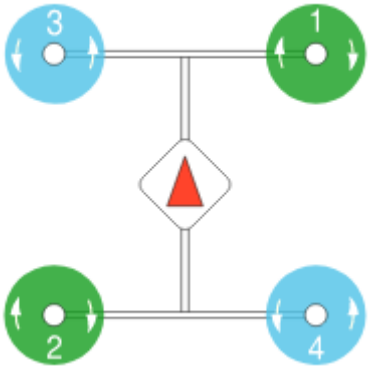


Figure 3.1.1: motor configuration

  - also $T_i = b_i\omega_i^2$
    It's better to design the controller using M_x, M_y, M_zinstead of ω_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
    $\therefore I = \begin{bmatrix} Ixx & 0 & \boldsymbol{-Ixz} \\ 0 & Iyy & 0 \\ \boldsymbol{-Ixz} & 0 & Izz \end{bmatrix}$
  - angular velocity of the flying taxi $\overrightarrow{\omega} = p\overrightarrow{i} + q\overrightarrow{j} + r\overrightarrow{k}$

– absolute change of $I\overrightarrow{\omega}$is the sum of change in magnitudes $I\overset{\circ}{\overrightarrow{\omega}} = I(\mathring{p}\,\overrightarrow{i} + \mathring{q}\,\overrightarrow{j} + \mathring{r}\,\overrightarrow{k})$ and directions $\overrightarrow{\omega} \times I\overrightarrow{\omega}$

3.1.1.2 Kinematics [2]

Using ZYX euler angles (ψ, θ, ϕ), the transformation matrix which can be used to transform vectors from inertial axes to body axes is:

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

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

As ZYX euler angles represents rotation from :

1. body axes around Z to intermediate axes(1) with angle ψ
2. intermediate axes(1) around y1 to intermediate axes(2) with angle θ
3. intermediate axes(2) around x2 to body axes with angle ϕ

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

$\therefore \overrightarrow{\omega} = p\,\overrightarrow{i} + q\,\overrightarrow{j} + r\,\overrightarrow{k} = \mathring{\psi}\overrightarrow{K}_{inertial} + \mathring{\theta}\overrightarrow{j_1} + \mathring{\phi}\overrightarrow{i_2}$

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

, $\overrightarrow{j_1} = \overrightarrow{j_2} = [2nd\,row\,R_x(\phi)]^T = \begin{bmatrix} 0 \\ cos(\phi) \\ -sin(\phi) \end{bmatrix}$

, $\overrightarrow{i_2} = \overrightarrow{i}$

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

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

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

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 , ϕ, θ , ψ

system:

[2] the study of motion without regard to forces or moments

$$\mathring{u} = -gsin\theta - qw + rv + \frac{F_{pusher}}{m} + \frac{X_{aero}}{m} \tag{3.1.1}$$

$$\mathring{v} = gcos\theta sin\phi - ru + pw \tag{3.1.2}$$

$$\mathring{w} = gcos\theta cos\phi - pv + qu - \frac{T_{allmotors}}{m} + \frac{Z_{aero}}{m} \tag{3.1.3}$$

$$\mathring{p} = \frac{I_x I_{xz} pq - I_{xz}^2 qr - I_{xz} I_y pq + I_{xz} I_z pq + I_y I_z qr - I_z^2 qr + I_{xz} M_z + I_z M_x}{I_x I_z - I_{xz}^2} \tag{3.1.4}$$

$$\mathring{q} = \frac{M_y - pr(I_x - I_z) - I_{xz}(p^2 - r^2)}{I_y} = \frac{Iz - Ix}{Iy} pr + \frac{My}{Iy} \tag{3.1.5}$$

$$\mathring{r} = \frac{I_x^2 pq - I_x I_{xz} qr - I_x I_y pq + I_{xz}^2 pq + I_{xz} I_y qr - I_{xz} I_z qr + I_x M_z + I_{xz} M_x}{I_x I_z - I_{xz}^2} \tag{3.1.6}$$

$$\mathring{\phi} = p + qsin\phi tan\theta + rcos\phi tan\theta \tag{3.1.7}$$

$$\mathring{\theta} = qcos\phi - rsin\phi \tag{3.1.8}$$

$$\mathring{\psi} = qsin\phi sec\theta + rcos\phi sec\theta \tag{3.1.9}$$

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

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

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

3.1.2 Linearization

We linearize the dynamics of the flying taxi around equilibrium point at steady flight where :
all rates = 0 , $\phi_o = \theta_o = 0$, $p = q = r = 0$, $\alpha = 0$

$$\therefore \mathring{\Delta u} = -g\theta + \frac{\Delta X_{pusher}}{m} + \frac{X_u^{aero}}{m}\Delta u + \frac{X_w^{aero}}{m}\Delta w \tag{3.1.13}$$

$$\mathring{v} = g\phi - u_o r \tag{3.1.14}$$

$$\mathring{w} = u_o q - \frac{\Delta T_{allmotors}}{m} + \frac{Z_u^{aero}}{m}\Delta u + \frac{Z_w^{aero}}{m}\Delta w + \frac{Z_q^{aero}}{m}\Delta q \tag{3.1.15}$$

$$\mathring{p} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} + \frac{I_{xz} M_z}{I_x I_z - I_{xz}^2} \tag{3.1.16}$$

$$\mathring{q} = \frac{M_y}{Iy} \tag{3.1.17}$$

$$\mathring{r} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} + \frac{I_{zx} M_x}{I_x I_z - I_{xz}^2} \tag{3.1.18}$$

$$\mathring{\phi} = p \tag{3.1.19}$$

$$\mathring{\theta} = q \tag{3.1.20}$$

$$\mathring{\psi} = r \tag{3.1.21}$$

$$\mathring{Z} = w \tag{3.1.22}$$

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	$kg.m^2$
Iy	0.16	$kg.m^2$
Iz	0.23	$kg.m^2$
Ixz	0.05	$kg.m^2$

3.2 Quad-copter phase Autopilot

The modified linearized model for the quad-copter phase:

$$\therefore \mathring{\Delta u} = -g\theta \tag{3.2.1}$$
$$\mathring{v} = g\phi \tag{3.2.2}$$
$$\mathring{w} = -\frac{\Delta T_{allmotors}}{m} \tag{3.2.3}$$
$$\mathring{p} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} + \frac{I_{xz} M_z}{I_x I_z - I_{xz}^2} \tag{3.2.4}$$
$$\mathring{q} = \frac{M_y}{Iy} \tag{3.2.5}$$
$$\mathring{r} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} + \frac{I_{zx} M_x}{I_x I_z - I_{xz}^2} \tag{3.2.6}$$
$$\mathring{\phi} = p \tag{3.2.7}$$
$$\mathring{\theta} = q \tag{3.2.8}$$
$$\mathring{\psi} = r \tag{3.2.9}$$
$$\mathring{Z} = w \tag{3.2.10}$$

3.2.1 Autopilot design

Autopilot block diagram is designed as follows:

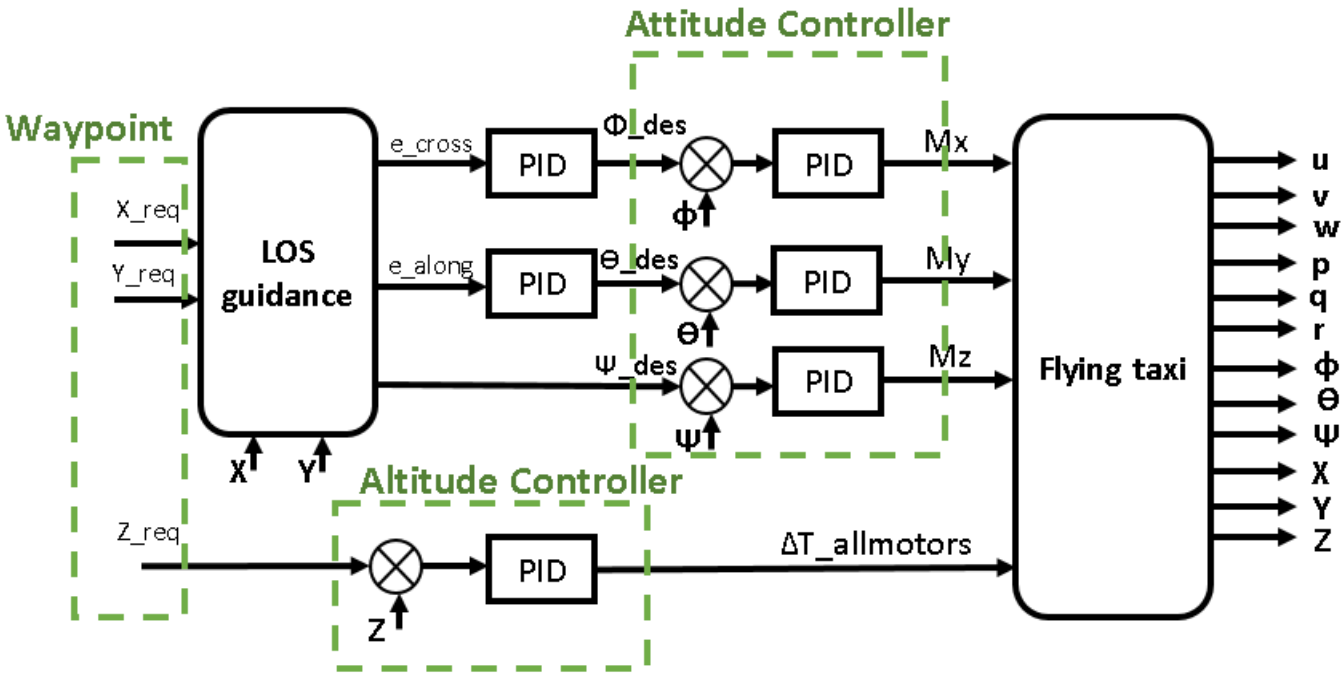


Figure 3.2.1: Autopilot design

3.2.1.1 Attitude controller

Roll Controller:

As shown in parameters table, Ixz is very small compared to Iz so the Mz effect is less than Mx 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): $\therefore \ddot{\phi} = \frac{I_z M_x}{I_x I_z - I_{xz}^2} \Rightarrow G_{phi} = \frac{\phi}{Mx} = \frac{\frac{I_z}{(I_x I_z - I_{xz}^2)\, s^2}}{}$
ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

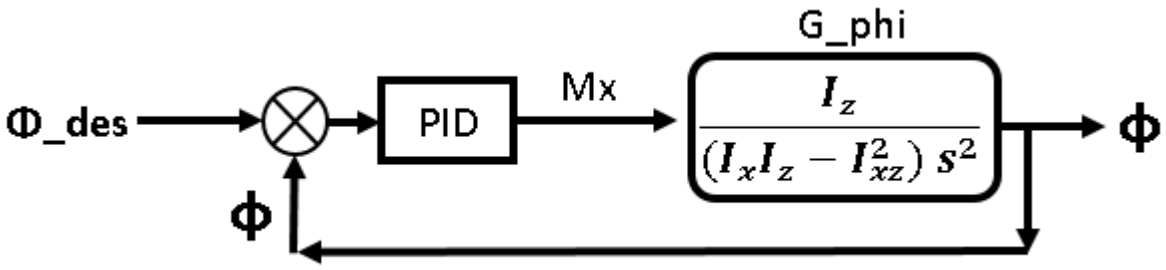


Figure 3.2.2: Roll Controller

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:

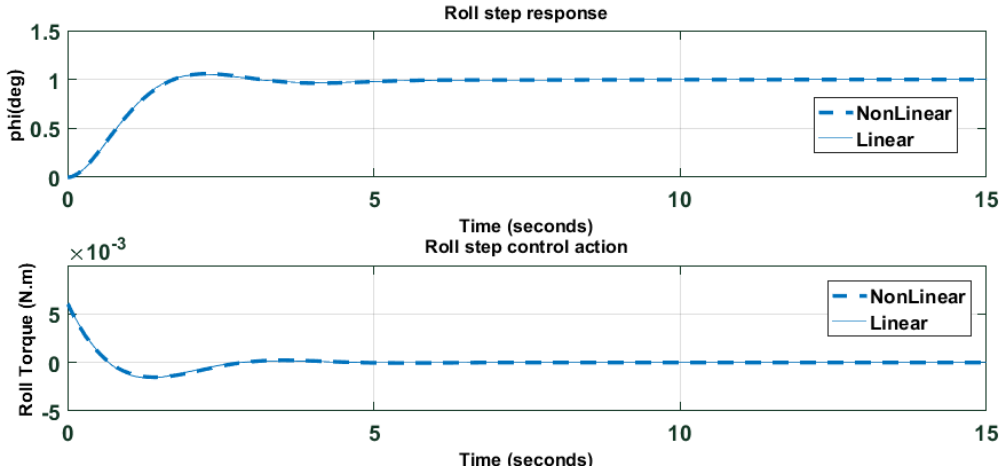


Figure 3.2.3: 1 deg desired Roll input

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

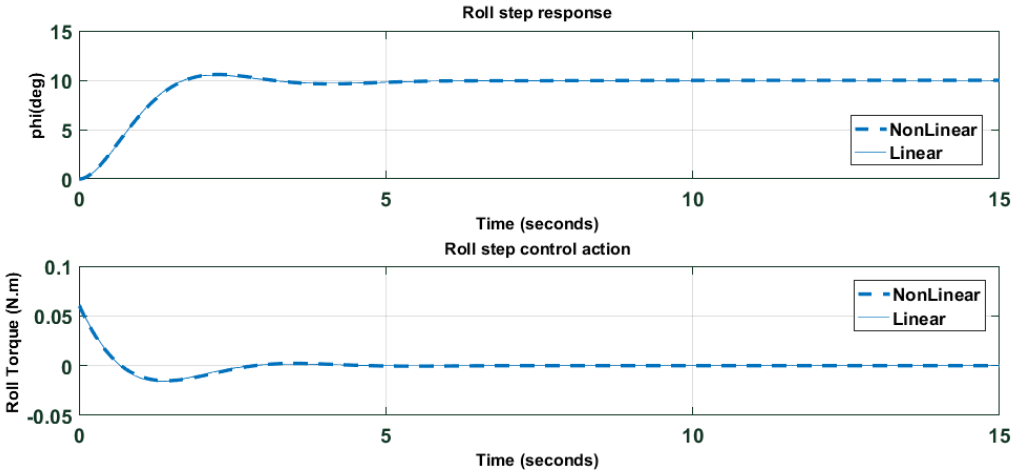


Figure 3.2.4: 10 deg desired Roll input

Pitch Controller:

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

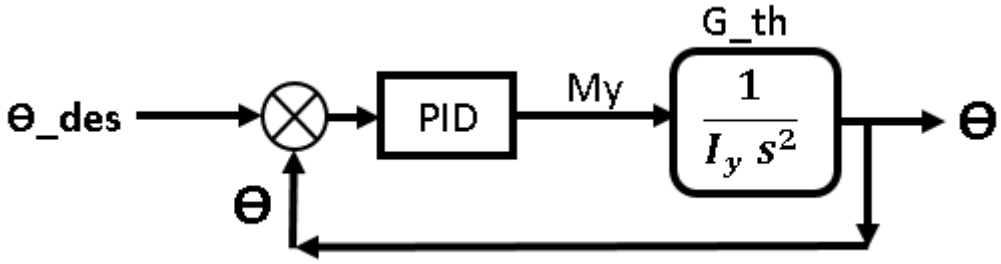


Figure 3.2.5: Pitch Controller

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

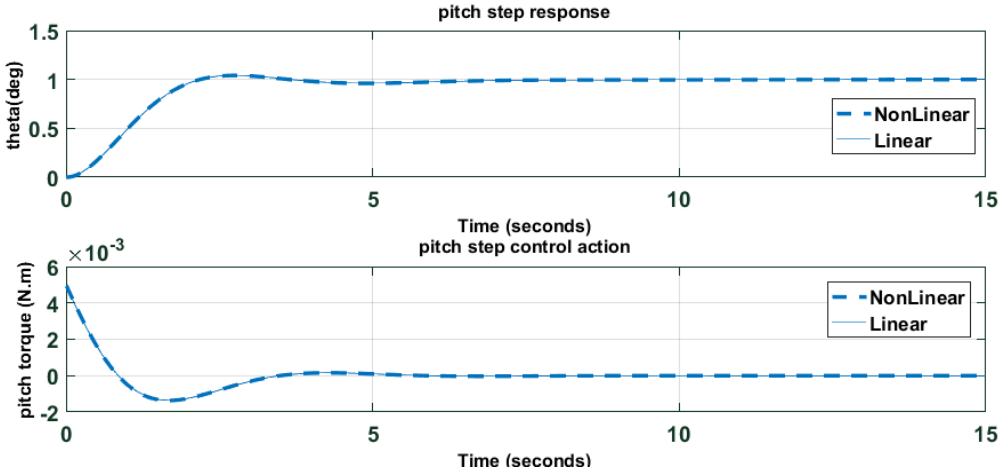


Figure 3.2.6: 1 deg desired pitch input

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

the nonlinear model) :

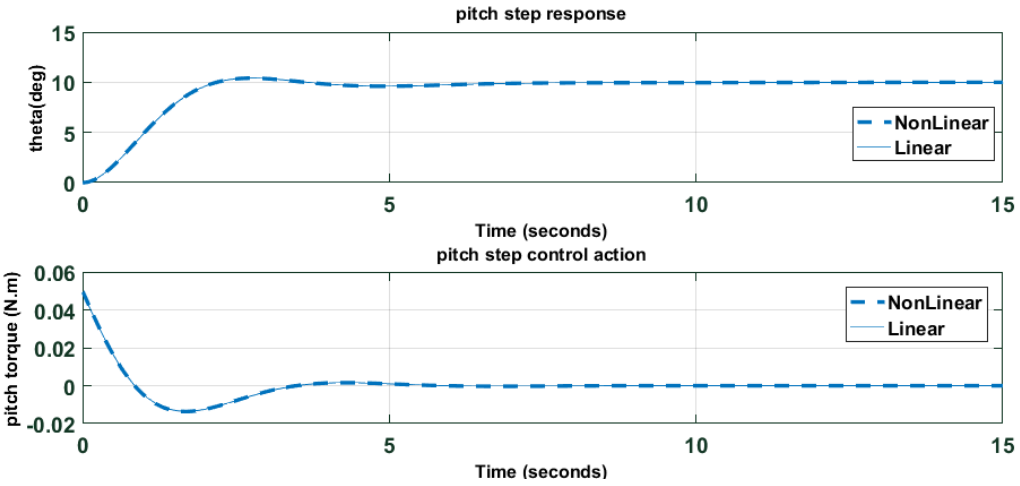


Figure 3.2.7: 10 deg desired pitch input

Yaw Controller:

As shown in parameters table, Ixz is also very small compared to Ix so the Mx effect is less than Mz 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): $\therefore \ddot{\psi} = \frac{I_x M_z}{I_x I_z - I_{xz}^2} \Rightarrow G_\psi = \frac{\psi}{Mz} = \frac{I_x}{(I_x I_z - I_{xz}^2)\, s^2}$

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

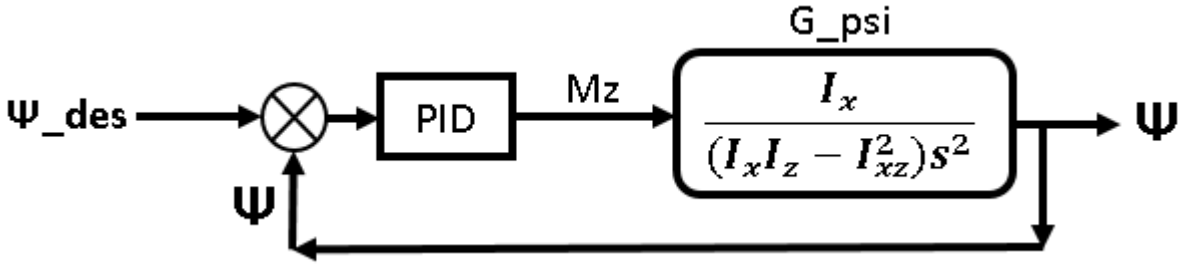


Figure 3.2.8: Yaw Controller

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

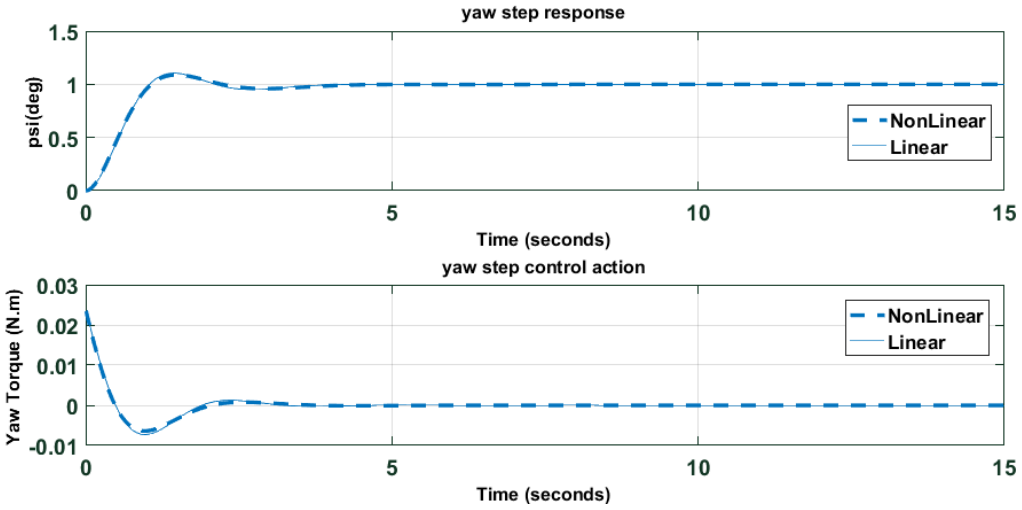


Figure 3.2.9: Yaw Step response and control action

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

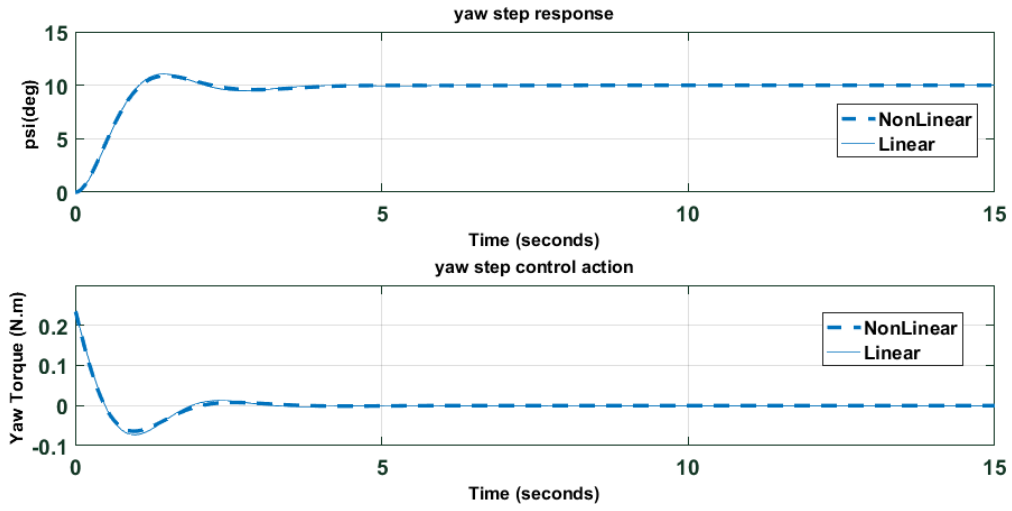


Figure 3.2.10: 10 deg desired yaw input

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.

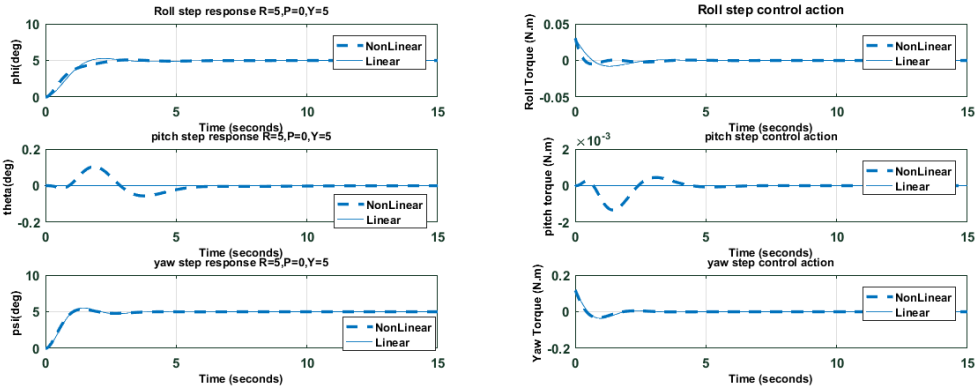


Figure 3.2.11: desired roll = 5 deg , desired pitch = 0 , desired yaw = 5

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

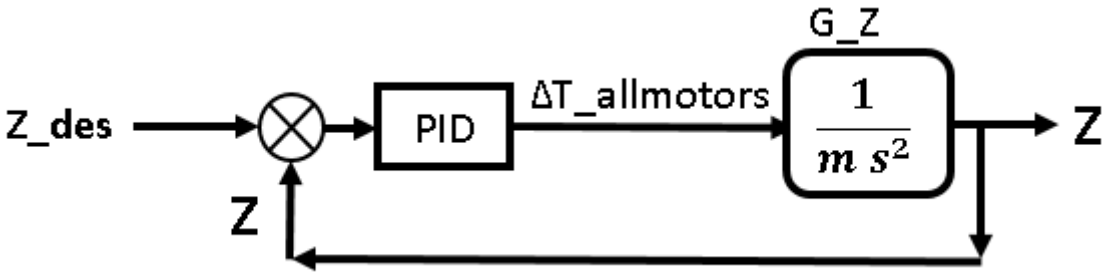


Figure 3.2.12: Altitude Controller

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:

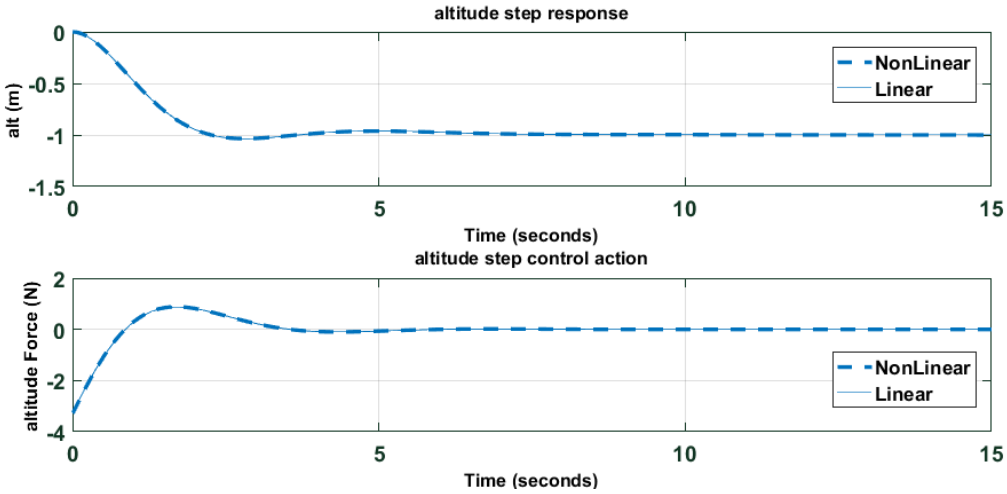


Figure 3.2.13: Altitude Step response and control action

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) :

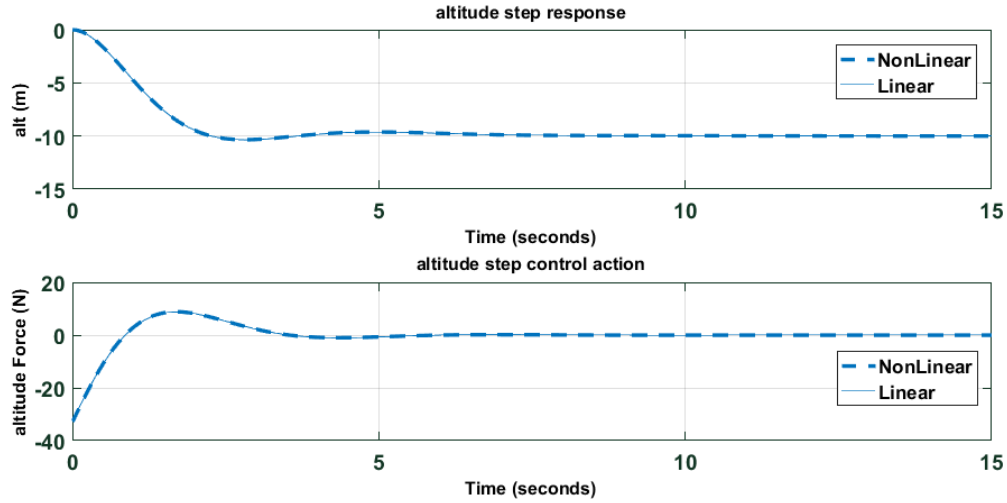


Figure 3.2.14: -4m altitude input

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

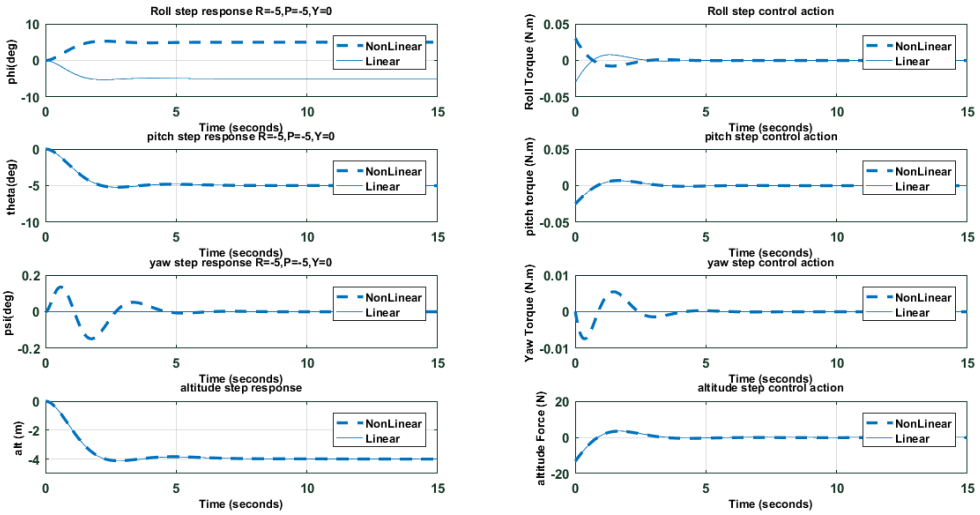


Figure 3.2.15: desired roll = -5 deg , desired pitch = -5 deg , desired altitude = -4 m (Z = 4 m)

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 espically 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.

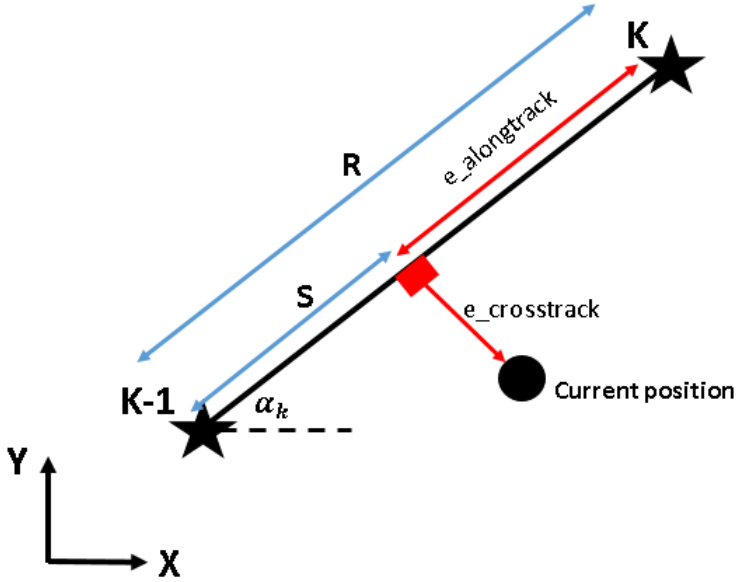


Figure 3.2.16: LOS

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

$$\psi_{des} = \alpha_k \tag{3.2.12}$$

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

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

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

$$e_{along-track} = R - s \tag{3.2.16}$$

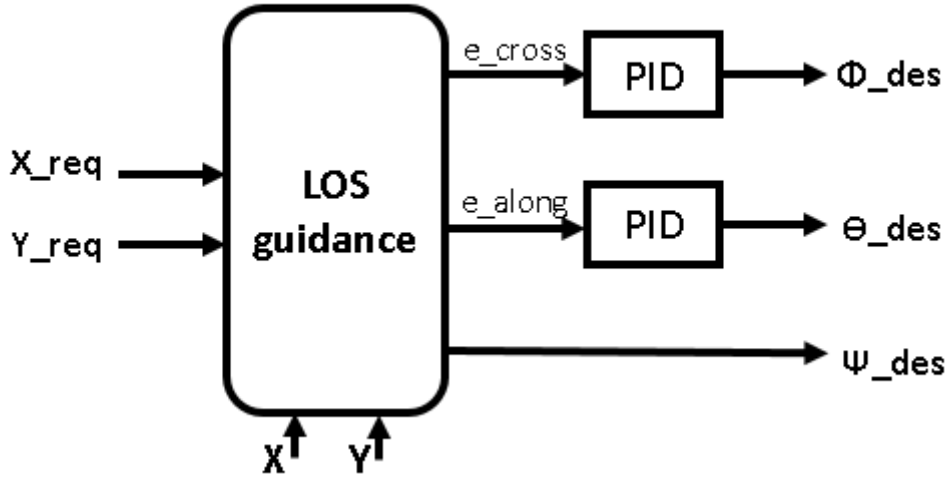


Figure 3.2.17: LOS block diagram

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) : $e_{along-track} = R - s \Rightarrow \mathring{e}_{along-track} = -u \Rightarrow e_{along-track} = \frac{-u}{s}$

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

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.

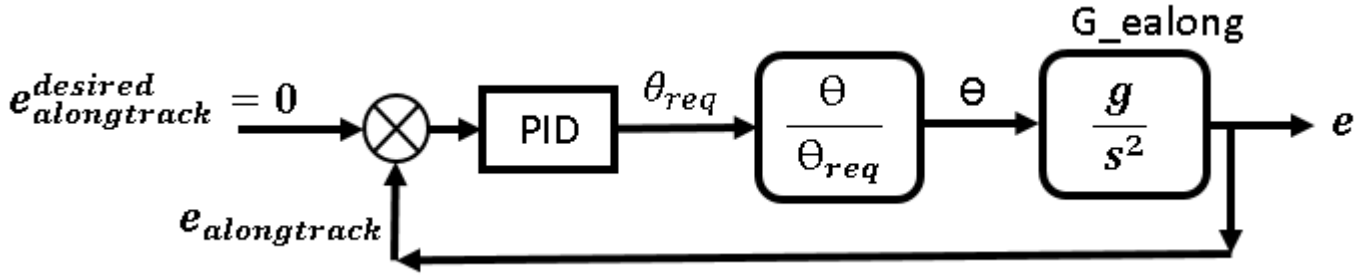


Figure 3.2.18: X guidance

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

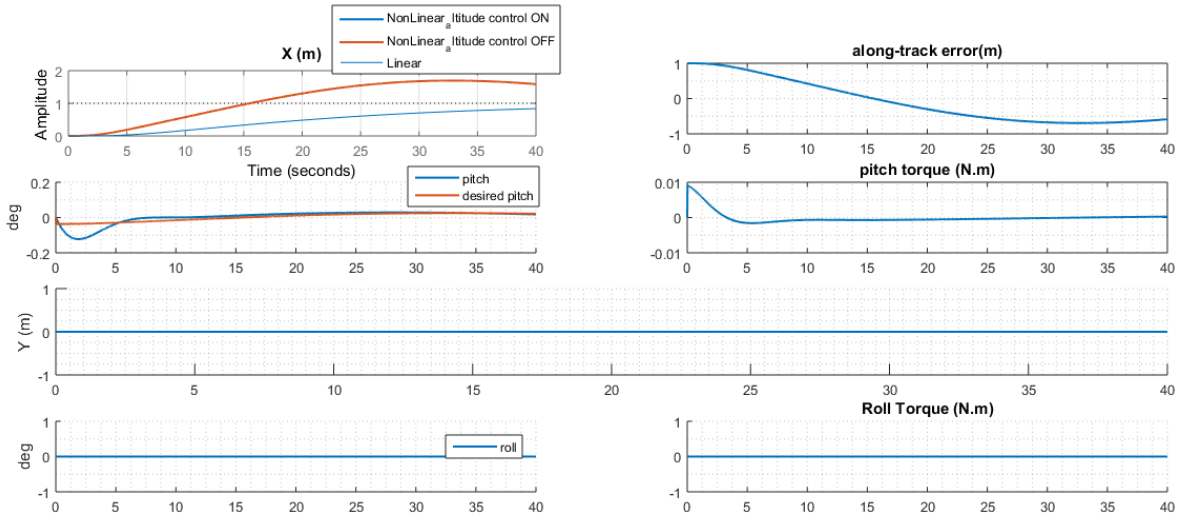


Figure 3.2.19: along-track control (trial : 1)

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

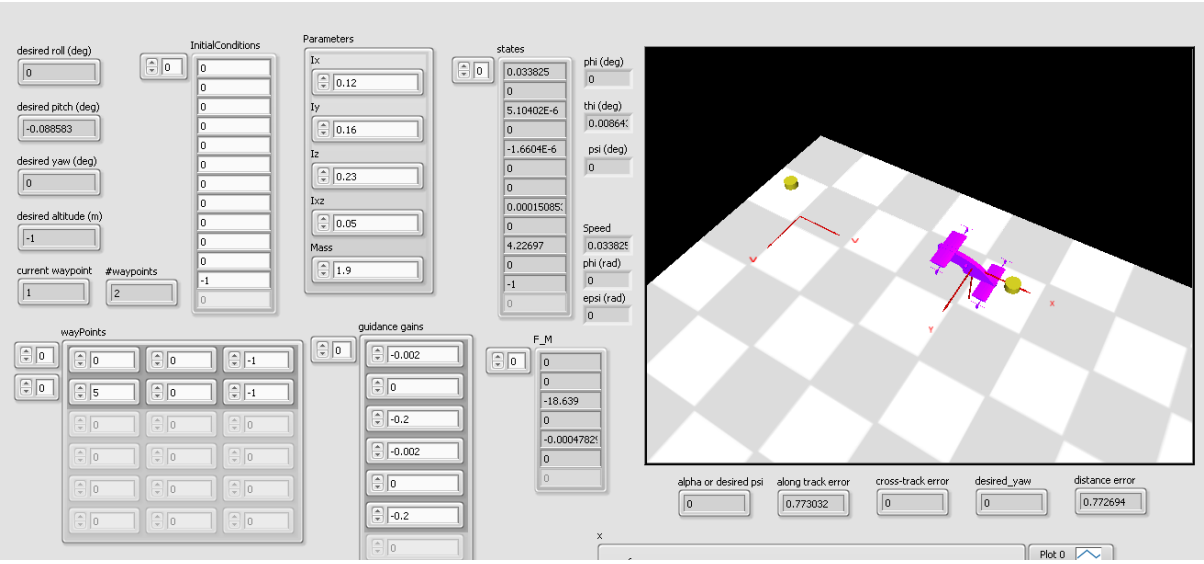


Figure 3.2.20: LabVIEW front panel

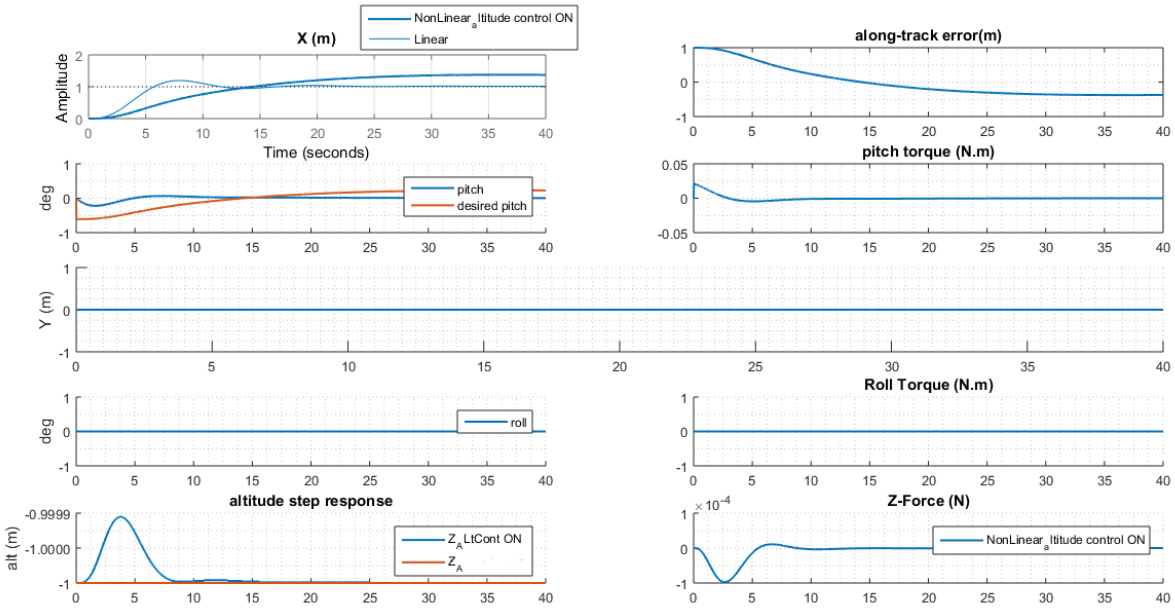


Figure 3.2.21: along-track control (trial : 2 increasing P) : high steady-state error

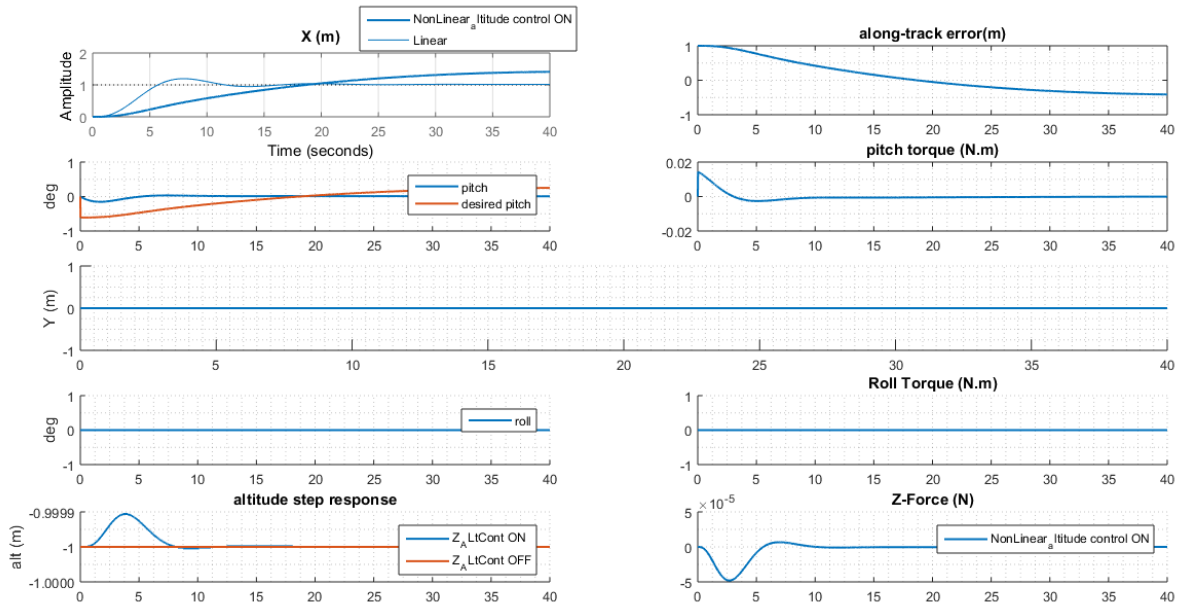


Figure 3.2.22: along-track control (trial : 2 decreasing D) : high steady-state error and the settling time is increased (as expected but we only wanted see the effect)

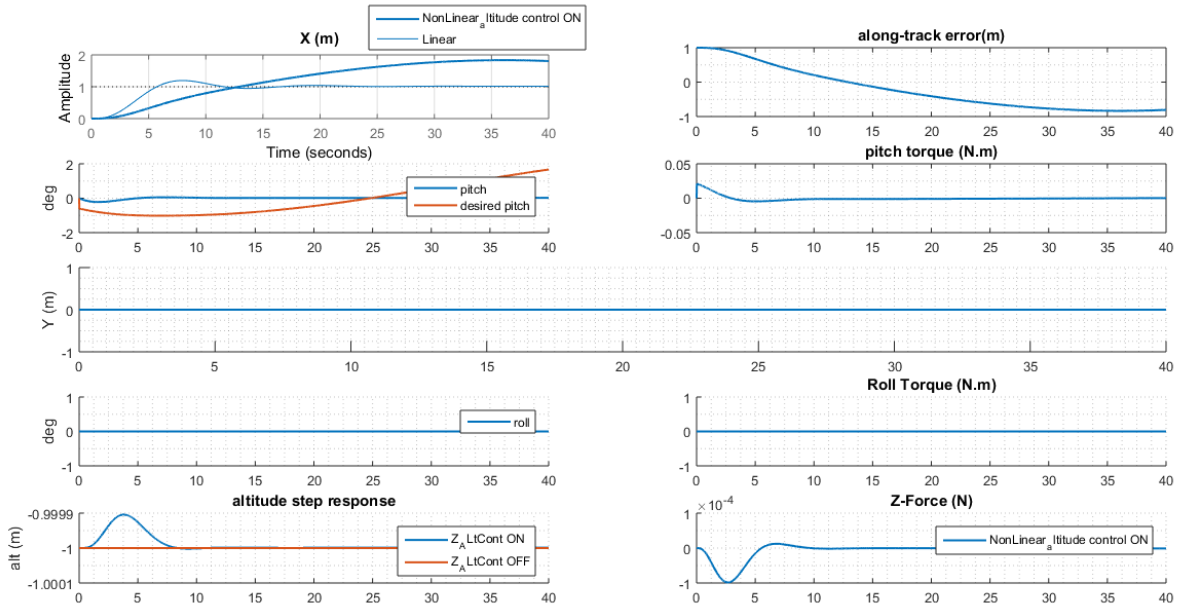


Figure 3.2.23: along-track control (trial : 3 adding small I term to eliminate steady-state error) : settling time is highly increased

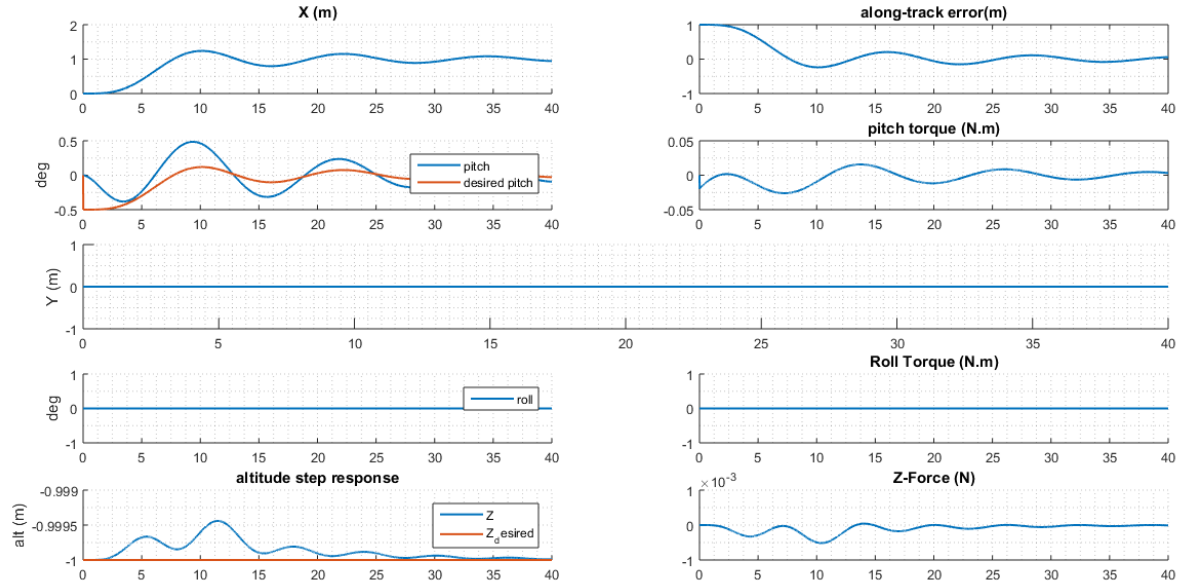


Figure 3.2.24: along-track error control (final trial) : increasing P and D

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

new gains after tunning 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: $\mathring{v} = g\phi \Rightarrow \mathring{e}_{cross-track} = v = \frac{g\phi}{s} \Rightarrow e_{cross-track} = \frac{g\phi}{s^2}$

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

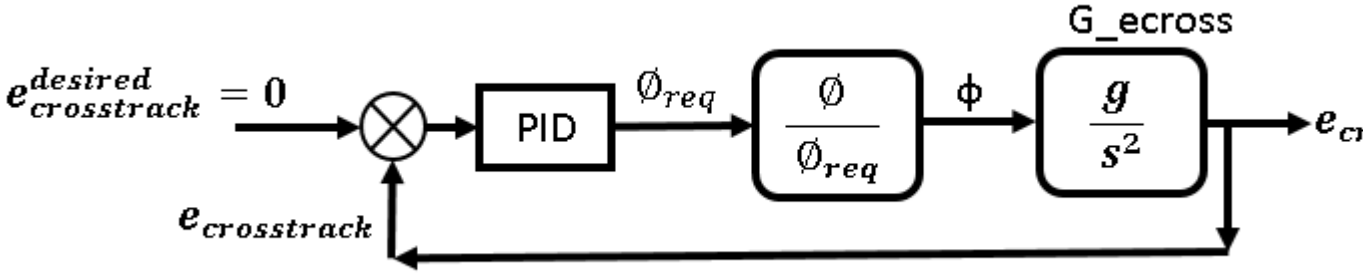


Figure 3.2.25: Y guidance

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 $\frac{\phi}{\phi_{req}}$ 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

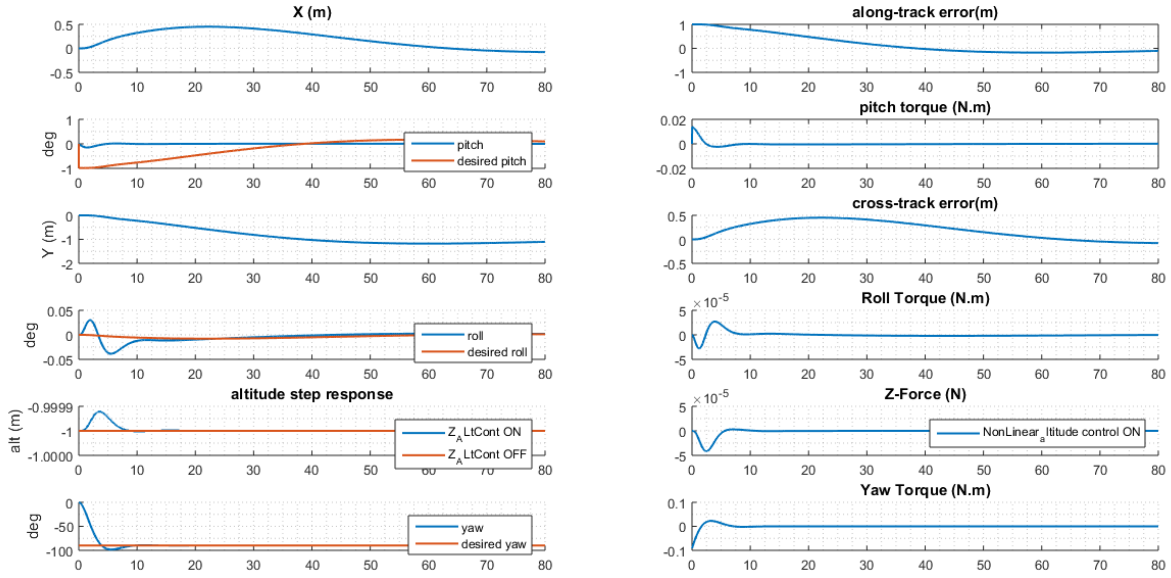


Figure 3.2.26: X,Y guidance waypoint(X=0 , Y=1 , Z=-1)

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

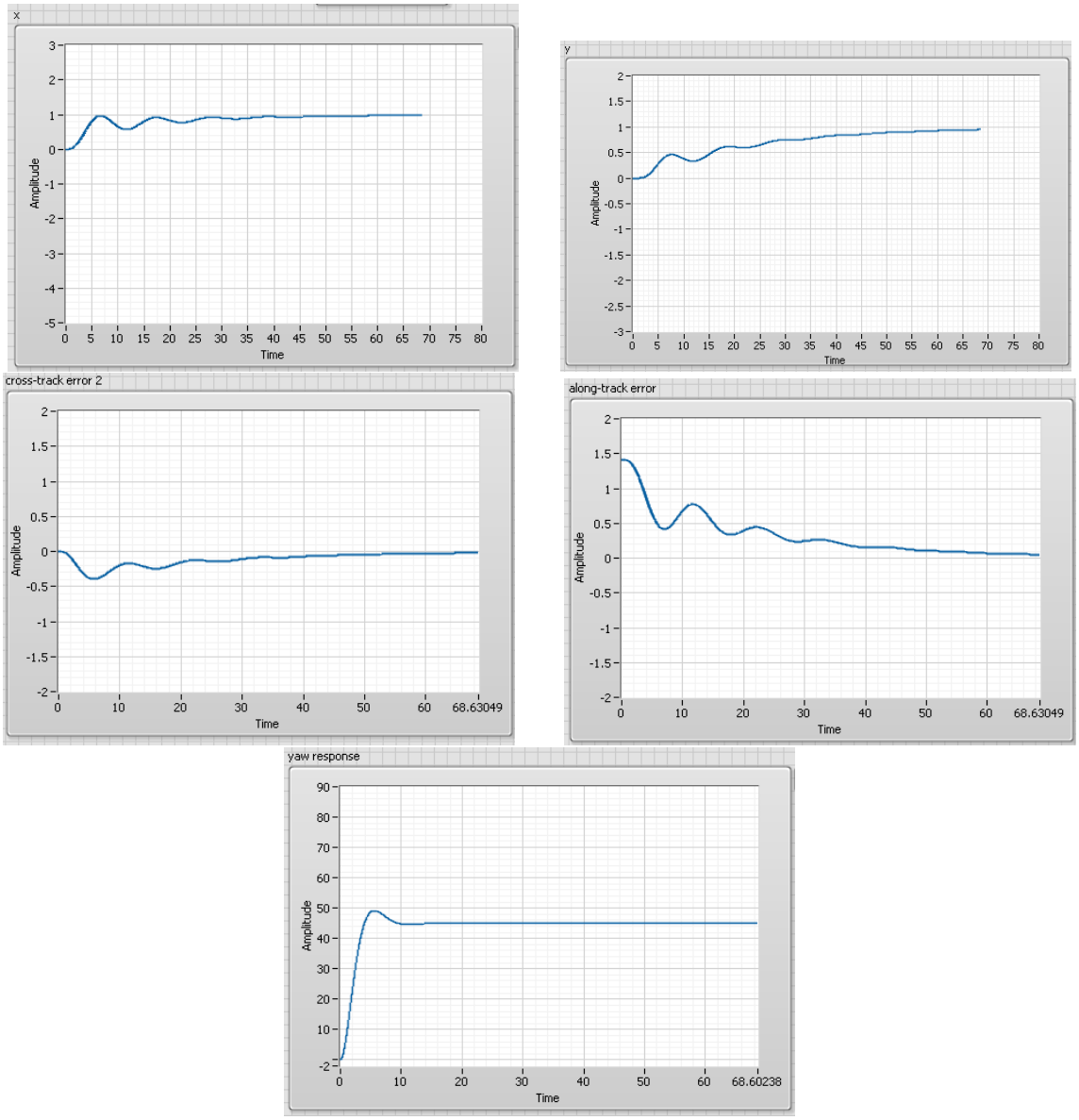


Figure 3.2.27: X,Y Guidance waypoint(X=1 , Y=1 , Z=-1)

3.2.2 Simulation using LabVIEW

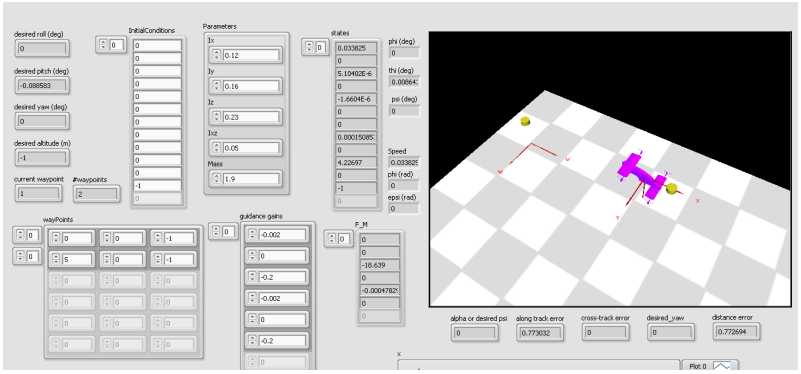


Figure 3.2.28: LabVIEW front panel

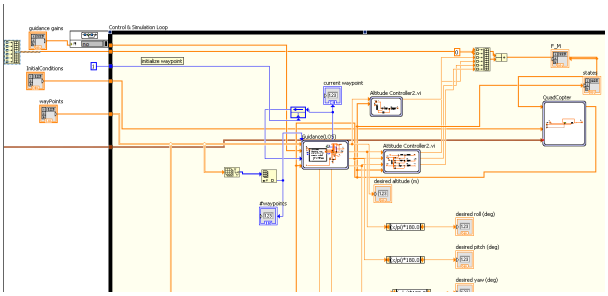


Figure 3.2.29: LabVIEW blockdiagram

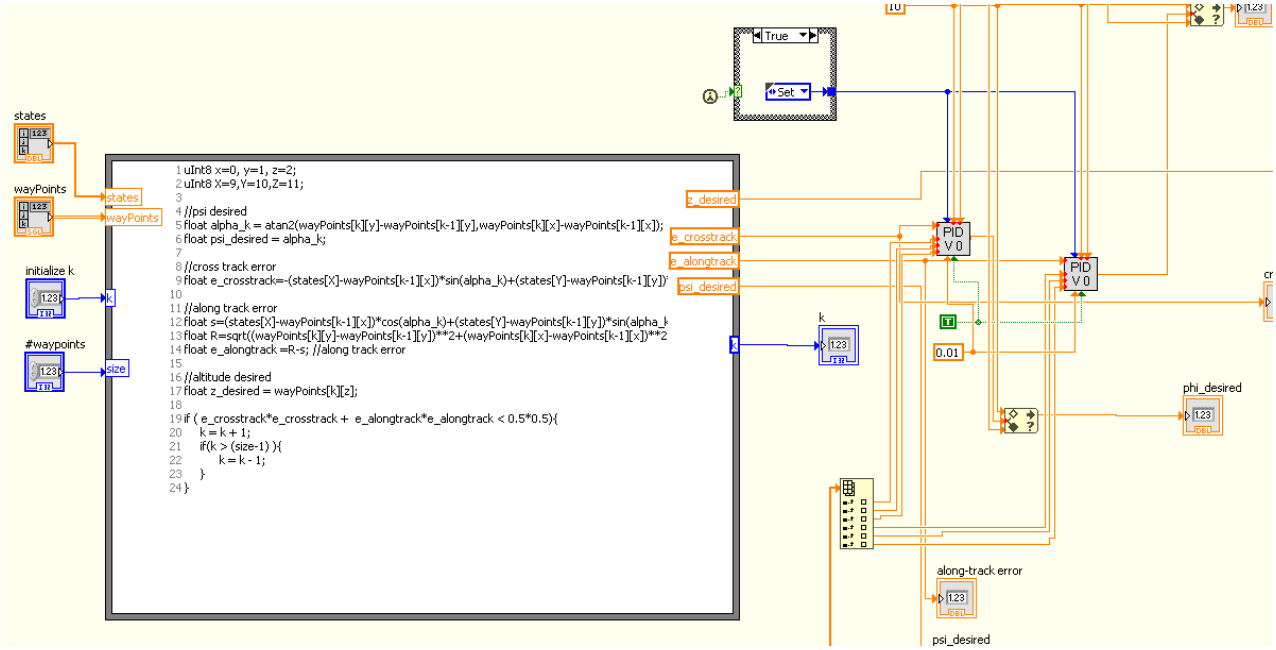


Figure 3.2.30: LabVIEW Guidance blockdiagram

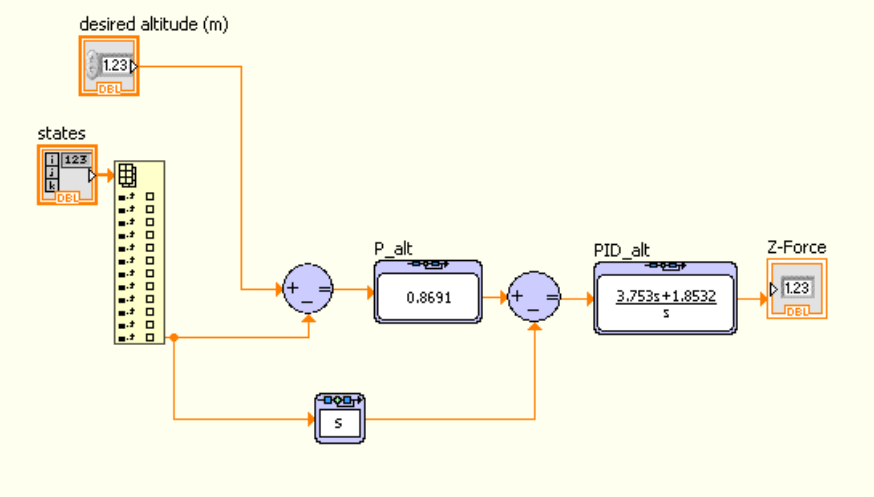


Figure 3.2.31: LabVIEW altitude BD

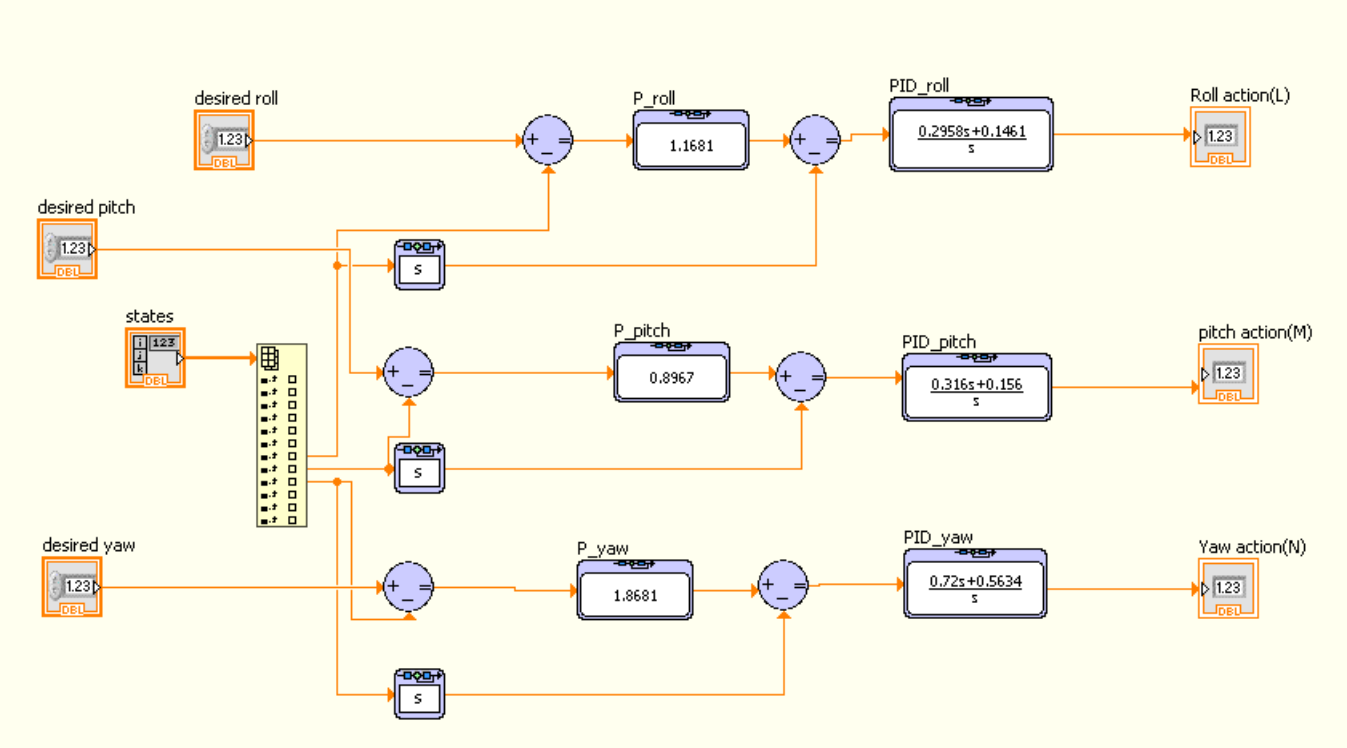


Figure 3.2.32: LabVIEW attitude controller BD

3.2.3 Commercial Autopilot

We used PIXHAWK as a commercial autopilot. PIXHAWK is an autopilot hardware for academic, hobby and developer communitites 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.

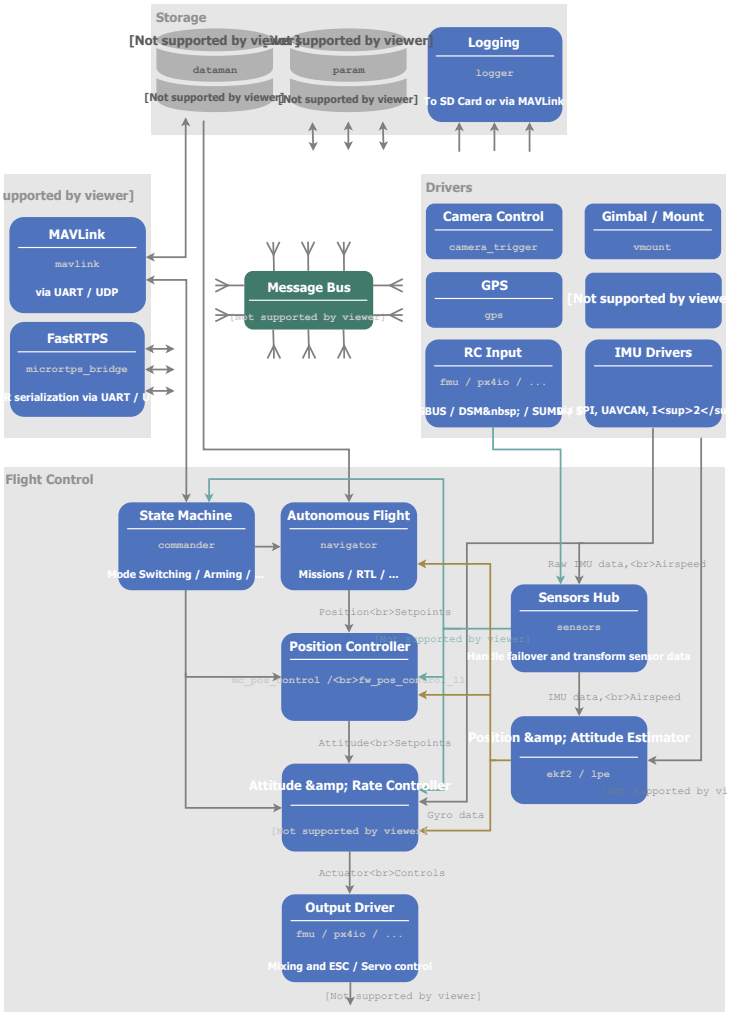


Figure 3.2.33: PX4 architecture from PX4 website

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

3.2.4 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 muticopter. 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.

3.2.5 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.

3.2.6 Position controller

inputs: Position setpoints & heading

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

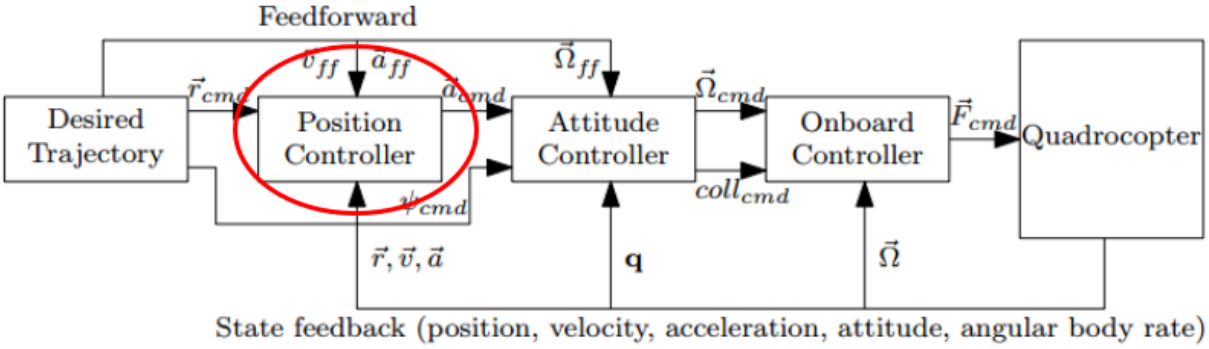


Figure 3.2.34: cascaded control architecture from "Nonlinear Quadrocopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D'Andrea, Raffaello,2013"

The feedforward signal isn't used for multirotors.
The position controller is implemented such that it consists of 2 loops :

1. 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);
}
```

2. 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 PIDs , 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.

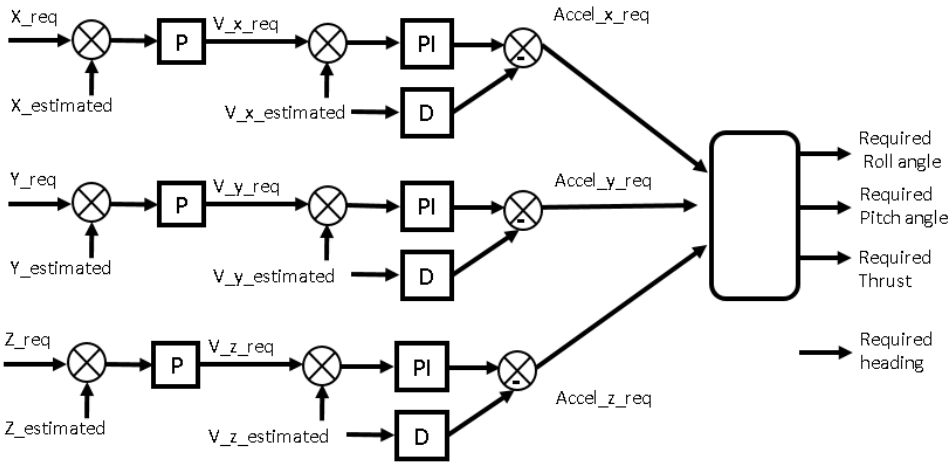


Figure 3.2.35: Position Controller

3.2.7 Attitude & Rate Controller

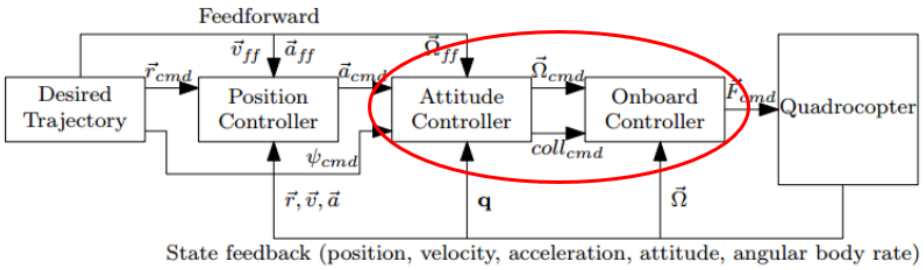
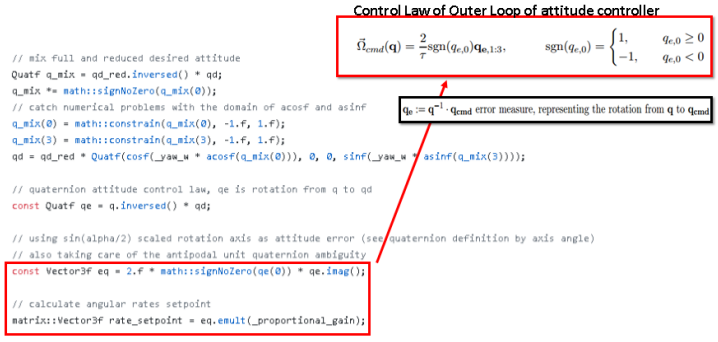


Figure 3.2.36: cascaded control architecture from "Nonlinear Quadrocopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D'Andrea, Raffaello,2013"

inputs: attitude setpoints(Roll-Pitch-Yaw)
outputs: **Required Forces and Moments**
The Attitude controller consists also of 2 loops :

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



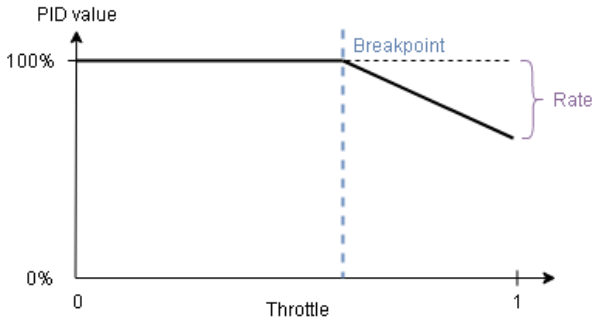
2. 2nd loop inputs: angular rates setpoint , outputs: required forces and moments , controller: PID-controller

```
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.



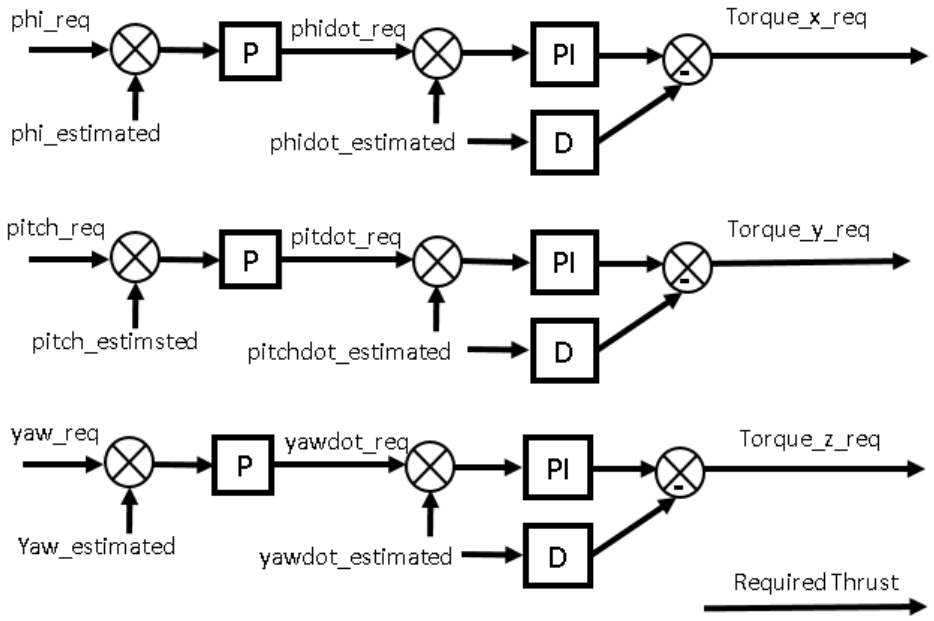


Figure 3.2.37: Attitude&rate controller

3.2.8 Estimator

In PX4 flightstack, there is 4 types of estimators are 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 as an **EKF(Extended Kalman filter)** for estimating **3D postion , 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** .

3.2.9 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 incude 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".

3.2.10 Gains Comparison

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 Iinertial 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[1] 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

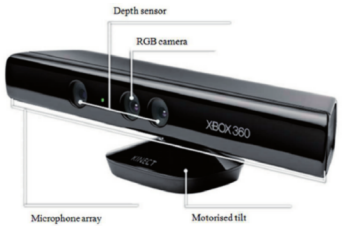
[1] Point cloud is a set of data points in space

<table>
<tr><th>Stereo Camera</th><th>Depth Cameras</th><th>Laser Scanner</th></tr>
<tr>
<td>

2 cameras are rigidily mounted to a common mechanical structure.
Its performance depends on : mechanical design , resolution , lens type & quality ,....
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.
ROS message: sensor_msgs/Image and sensor_msgs/CameraInfo
Image processing packages can be used to handle the outputs of the cameras to achieve the stereo vision

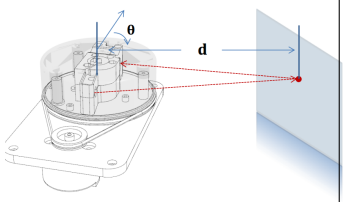
ZED stereo Camera : from 0.5 to 20m at 100FPS, indoors and outdoors.
</td>
<td>

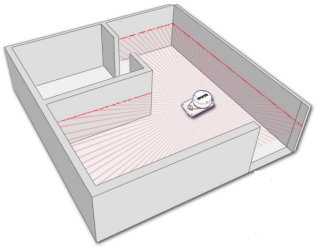
Unlike the passive stereo camera, they are active cameras that shine a texture pattern on the surfaces and from the deformation of the pattern thay can get the informations of the scene.
They greatly improves the system performance.
typically operate near-infrared wavelength to reduce system sensitivity to the colors of the objects.
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.
the output of those cameras is point clouds which are estimated 3D points of the scene
ROS message: sensor_msgs/PointCloud2



</td>
<td>

superior accuracy (about 14 mm error)
longer sensing range than cameras (0.2 - 6 m)





ROS message: sensor_msgs/LaserScan

</td>
</tr>
</table>

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 [2]	ZED stereo[3]	Kinect[4]	
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

[2] https://www.stereolabs.com/zed-mini/
[3] https://www.stereolabs.com/zed/
[4] 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[5] 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

[5]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*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 synchronoused to capture data ar 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 bond) then do the sparse stereo matching
  - SLAM employed to get pose from the successfully matched features

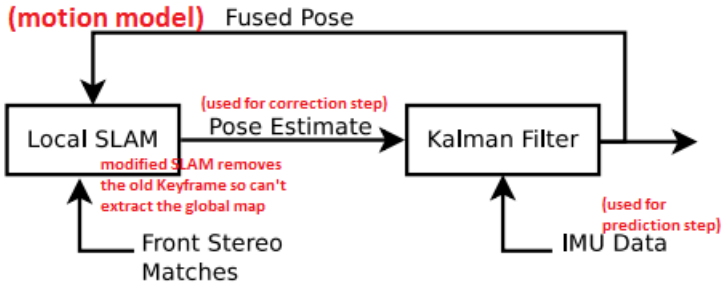


Figure 4.1.2: 2 forward-facing stereo configuration

  - 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 Configurarion

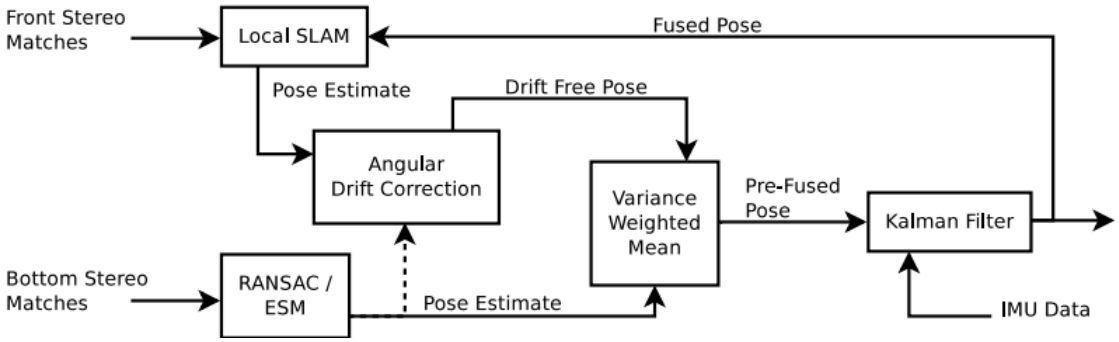
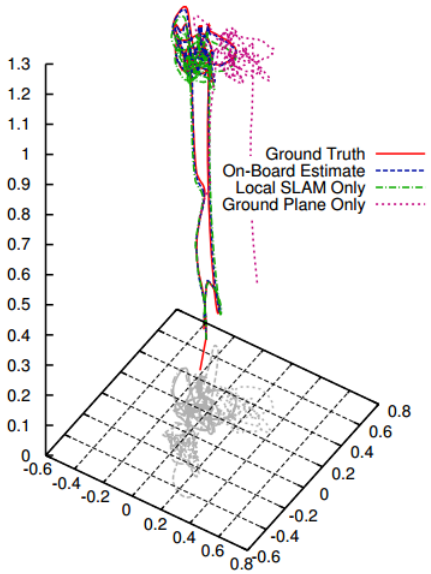


Figure 4.1.3: final configuration

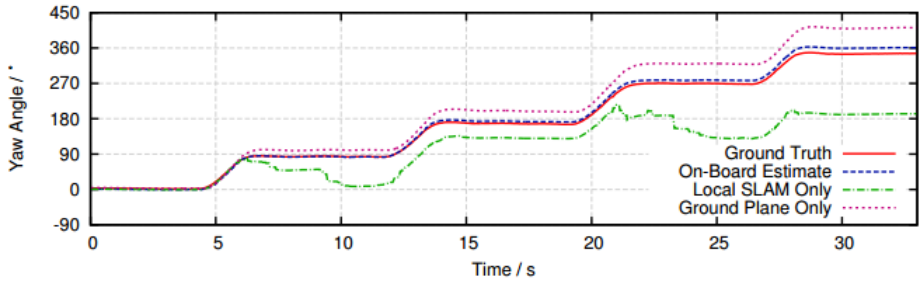
- Final Results



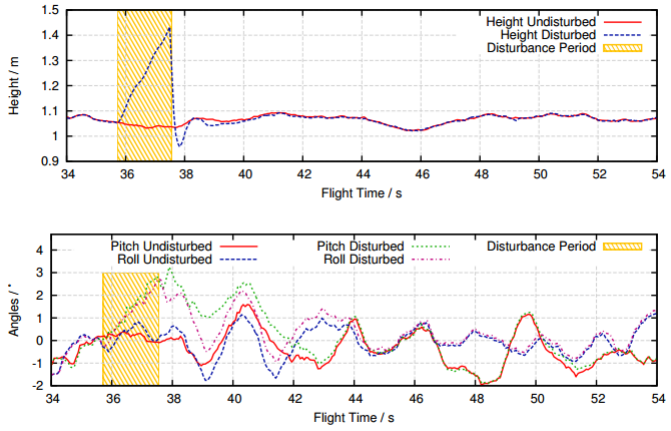
Ground truth and estimated position during autonomous take-off, hovering and landing. Scale is in meters.

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

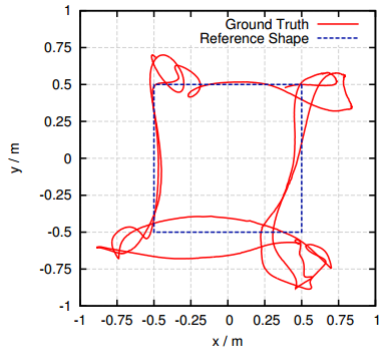
Position estimation errors for the examined processing methods.



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 plannig)	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)

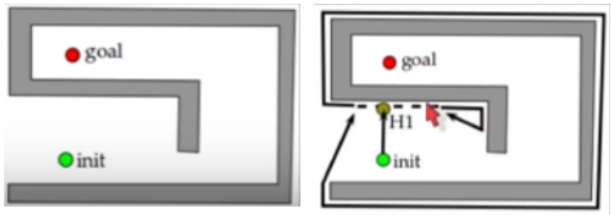
obstacle avoidance methods can be concluded as follows:

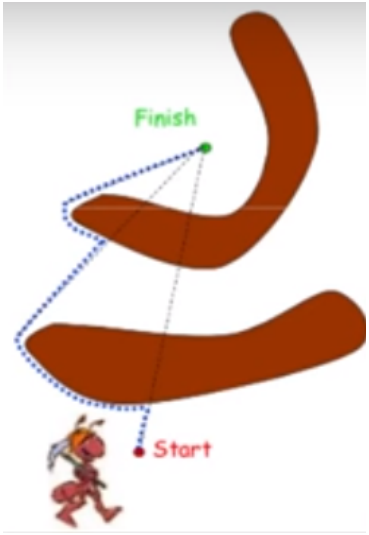
Global Obstacle avoidance	Local Obstacle avoidance
Grassfire algorithm - Dijkstra's algorithm - A* search	Bug algorithms - Vector field histogram

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.

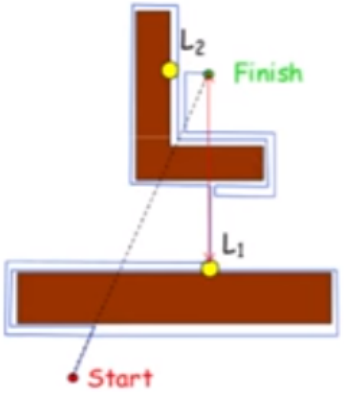
- **Bug Algorithms**
  inspired by insects which follow the obstacle wall to bypass it. There is many variants of Bug algorithms. They differ in sensors and amount of the memory used.
  - Bug 0
    - follow obstacle boundary until goal is clear
    - No memory used
    - It's either right or left wall following
    - Don't guarantee **Completeness**[a]



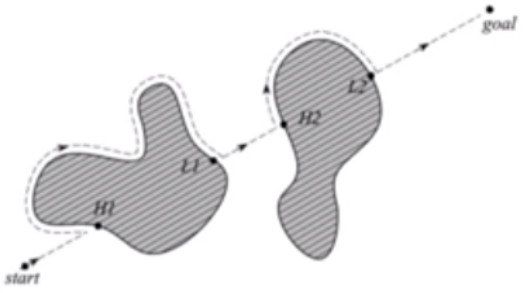


[a] Assuming there is a path to the goal, Bug 0 don't guarantee finding it.

- 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

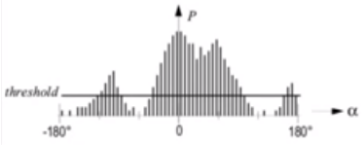


- Bug 2
  - wall follow the obstacle and leave when meets the goal line such that the second point on the goal line is closest to the goal than the first point at which we left the goal line and started to follow the wall.
  - This method is complete.[a]
  - Faster than Bug.



[a] 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[6] and the previous direction [7]
  - 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.

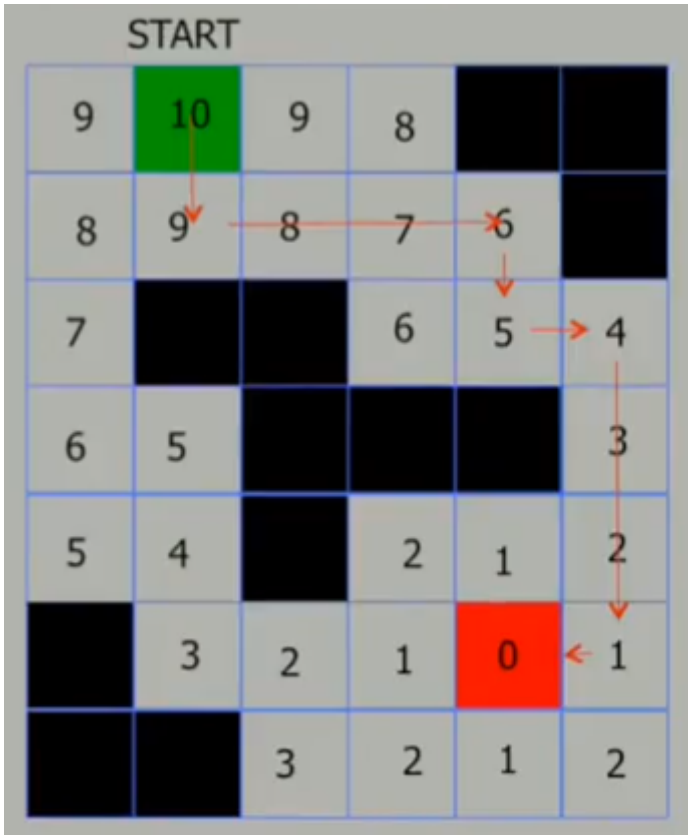
[6]
* difference between the new direction and the goal direction

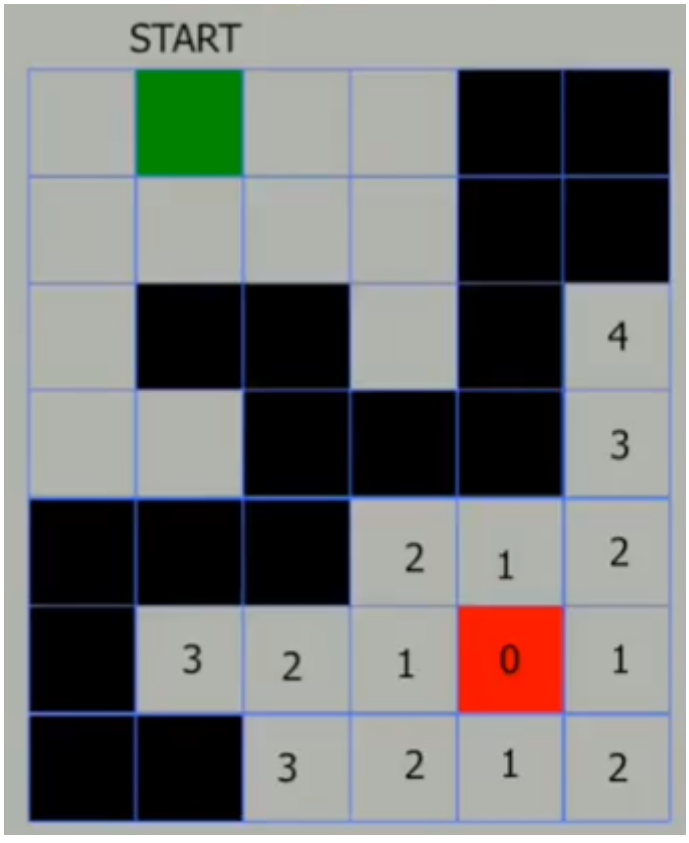
[7] difference between the previously selected direction and the new direction

4.2.2 Global methods

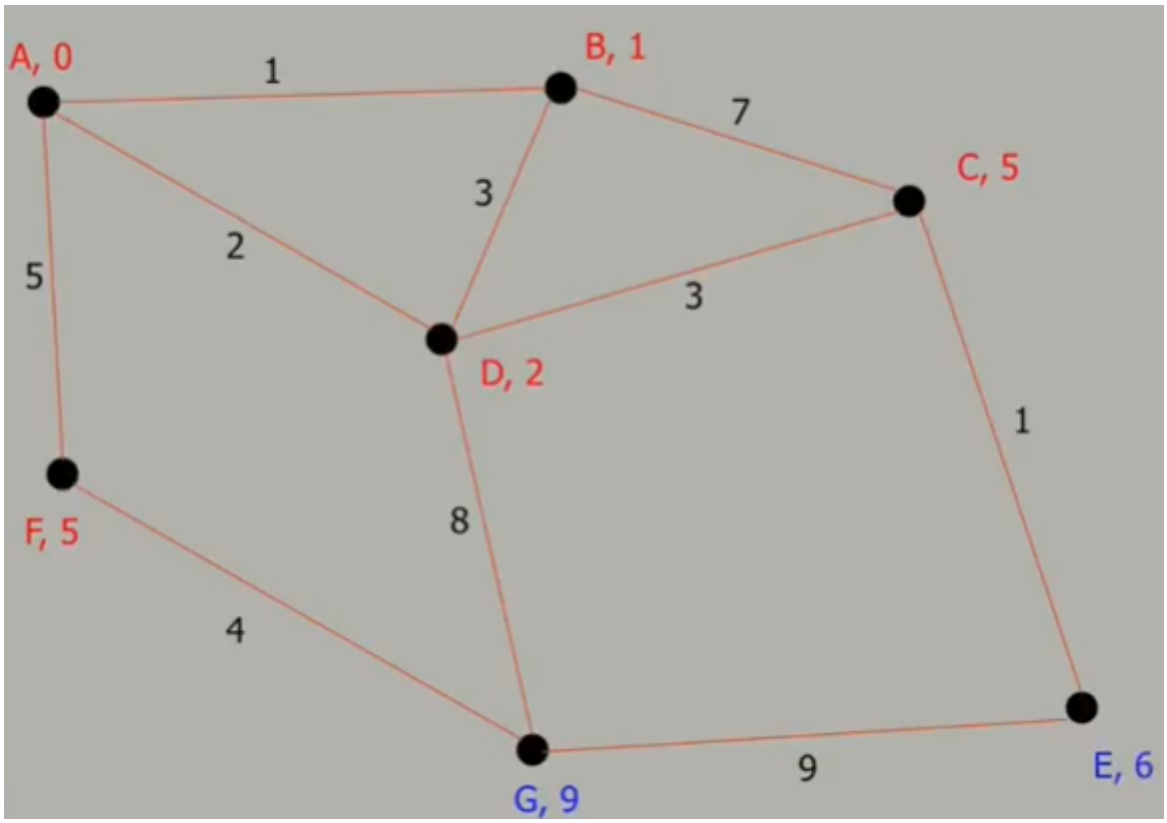
The vehicle has a map of the environment or builds it at run-time then computational motion planning methods, it finds the optimal path to the goal which avoids any obstacles.

- Grassfire algorithm

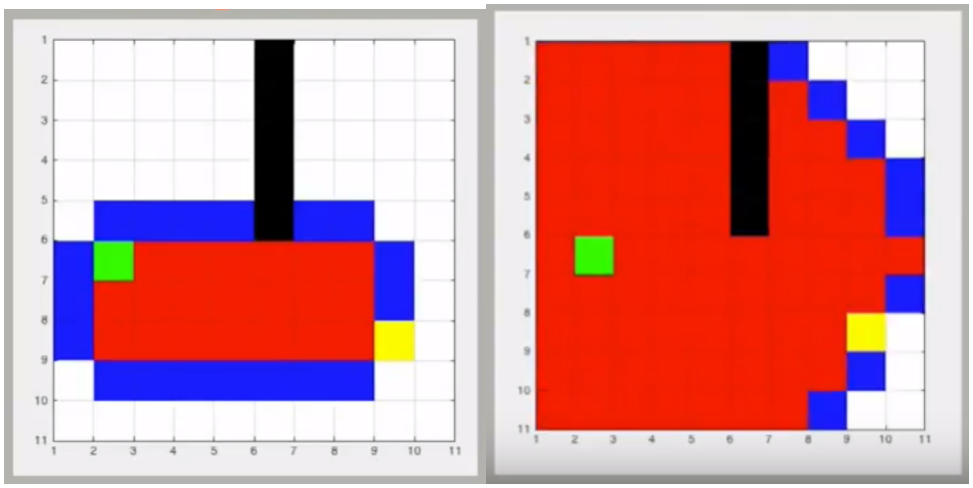




- Dijkstra's algorithm



- 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.

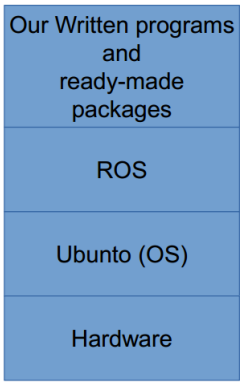


Figure 5.1.1: ROS architecture

5.1.1.1 History of ROS

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

[1] Open Source Robotics Foundation

5.1.1.2 ROS Philosophy

Peer-to-Peer-communication: individual programs communicate over defined API[2],(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.

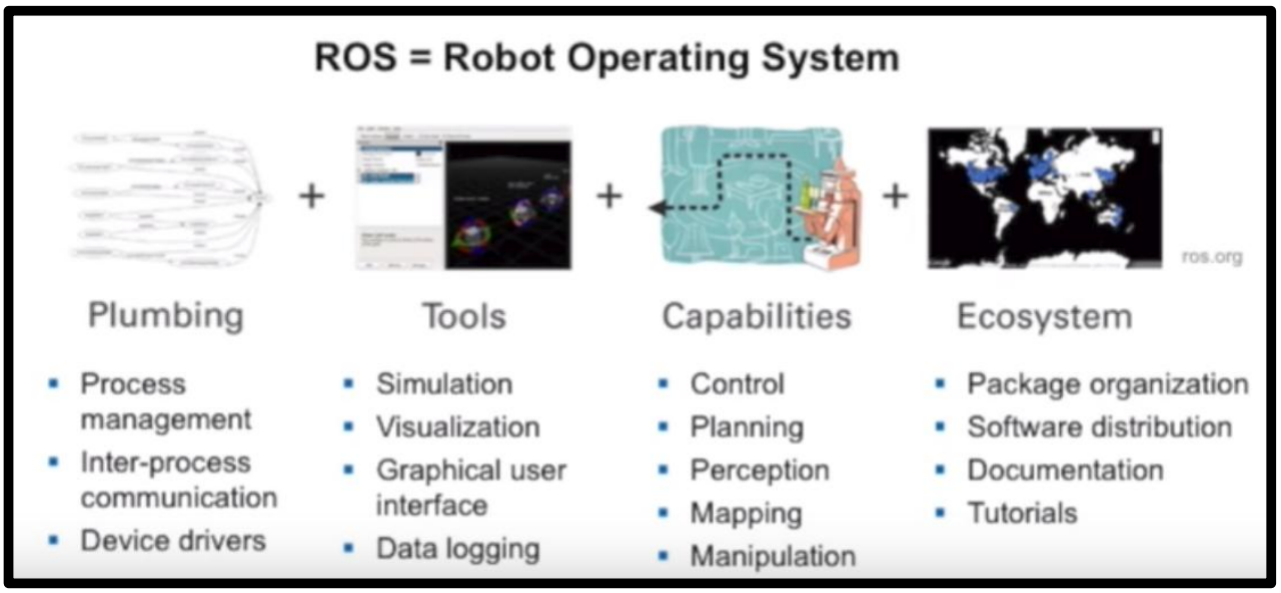


Figure 5.1.2: Why ROS?

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[3]to reduce the cost (MyRIO is expenssive).

[2]Application programming interface : it is a set of methods of communication among various components
[3]If that will achieve the real-time requirements which is a must in Aeronutical applications

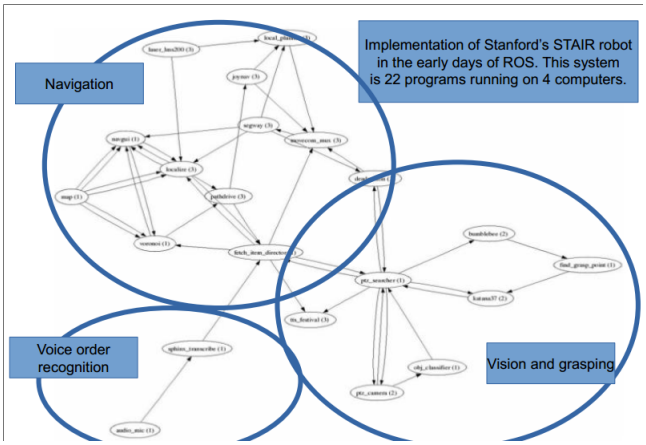


Figure 5.1.3: ROS graph of Stanford's STAIR robot nodes.

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:

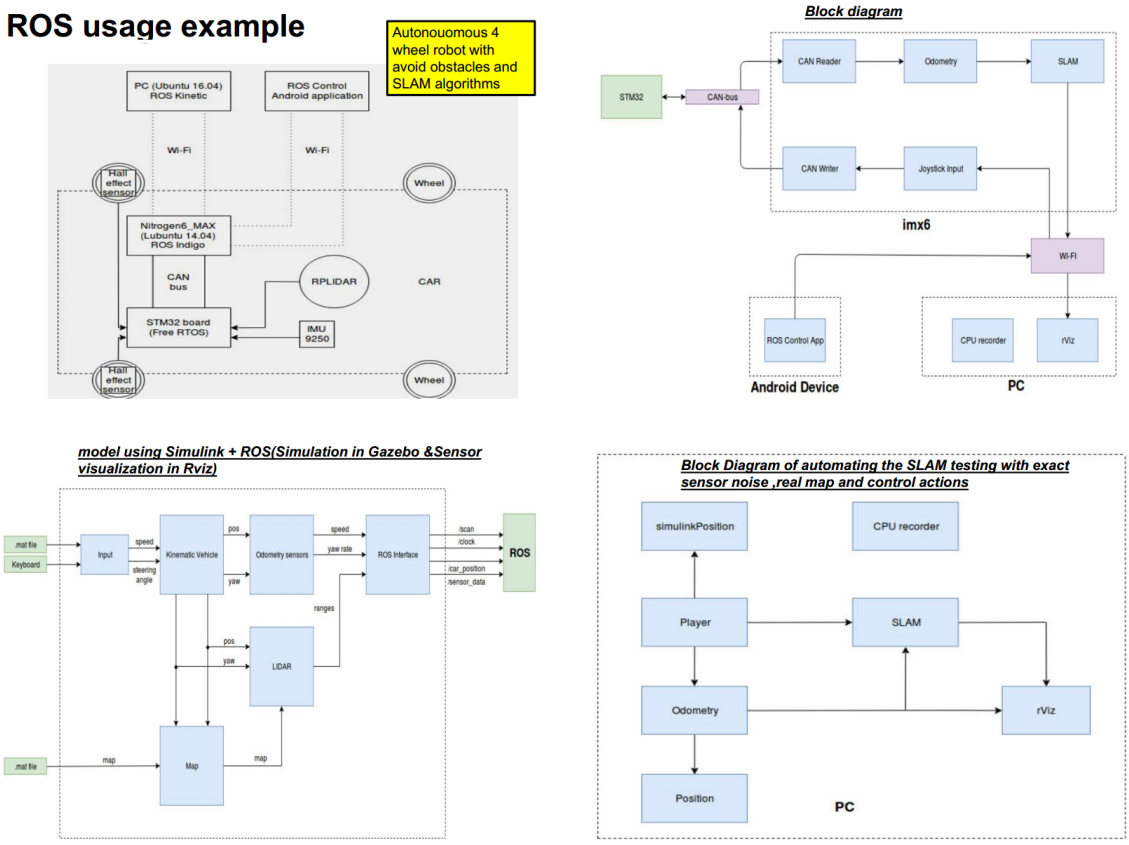
There's a fair amount of complexity in ROS such as : distributed computation, multi-threading , event-driven programming, and other concepts lie at the heart of the system. If you're not already familiar with at least some of these, ROS can have a hard learning curve[4]

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.

[4]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.



Rviz used to visualize the sensors outputs

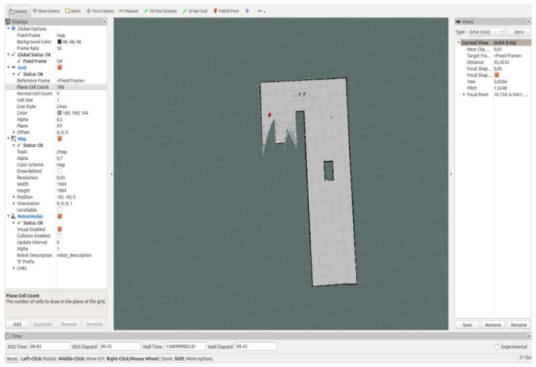


Figure 5.1.4: ROS example

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 /opy/ros/indigo[a]/setup.bash
```

[a] 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

```
>roscd
```

  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-purpuse executable program
- individually compiled, executed and manged
- 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 generate 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 recieves msgs). They can communicate with each others 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 avid 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 an linear velocities in 3D . It holds a msg called "Twist" which defined in a package called "geometry_msgs" and this msg contains Linear and Angular velocities in x,y,and z deirections

- remapping is a very useful feature ROS can do on topics. For example if I have a node called "image_view" reads from a topic called "image" which published in by the "camera" node but we have two cameras (left and right) so we can remap the topic to be exist in a two separate name spaces each hold different data for each camera but definitely the same msg so now we can have "/right/image" topic and "/left/image" topic and makes "image_view" node to read from both.

- **ROS messages**
- Data structure defining the type of a topic.
- comromised 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 commuinity besides there is a varity of messages that we may never need to define new one.
- ```
  >rostopic type /topic_name
  >rostopic pub /topic_name msg_type data
  ```

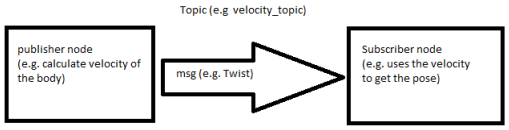


Figure 5.1.5: ROS msgs

- **ROS Launch**
- launch is a tool for launching multiple nodes (as well as setting parameters) because it isn't practicle 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 naem
  - find other CMake or catkin packages needed for build: find_package()
  - for adding msgs / actions / sevices: 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 exicte 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

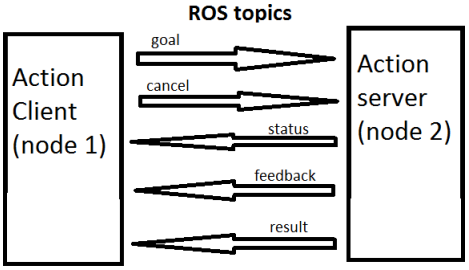


Figure 5.1.6: ROS actions

- 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).

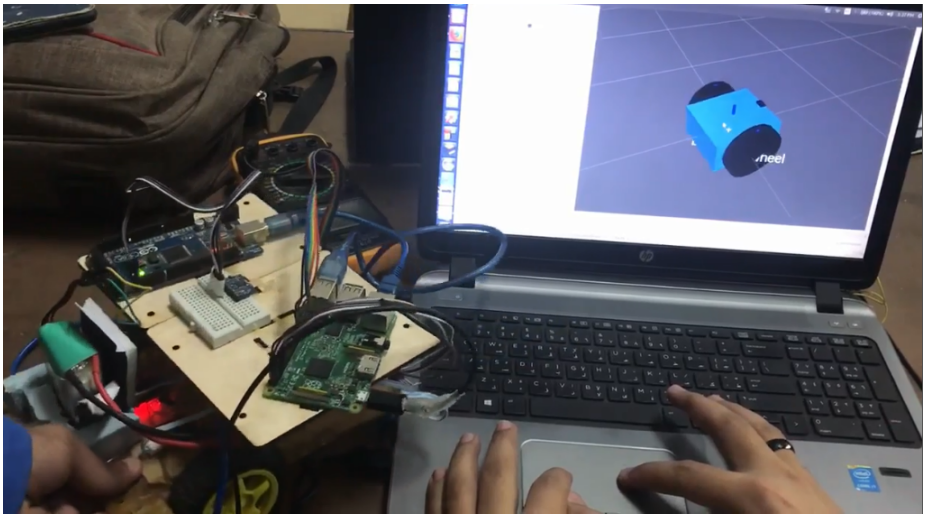


Figure 5.2.1: Differential Drive Robot (DD robot)

- **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**

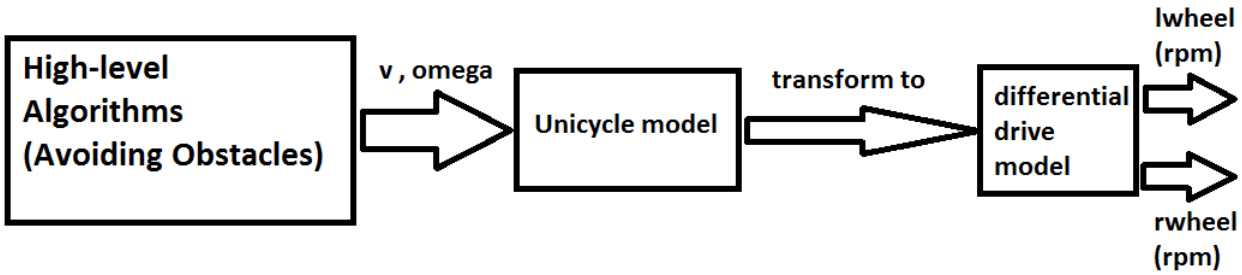


Figure 5.2.2: differential drive model

- **Block Diagram**

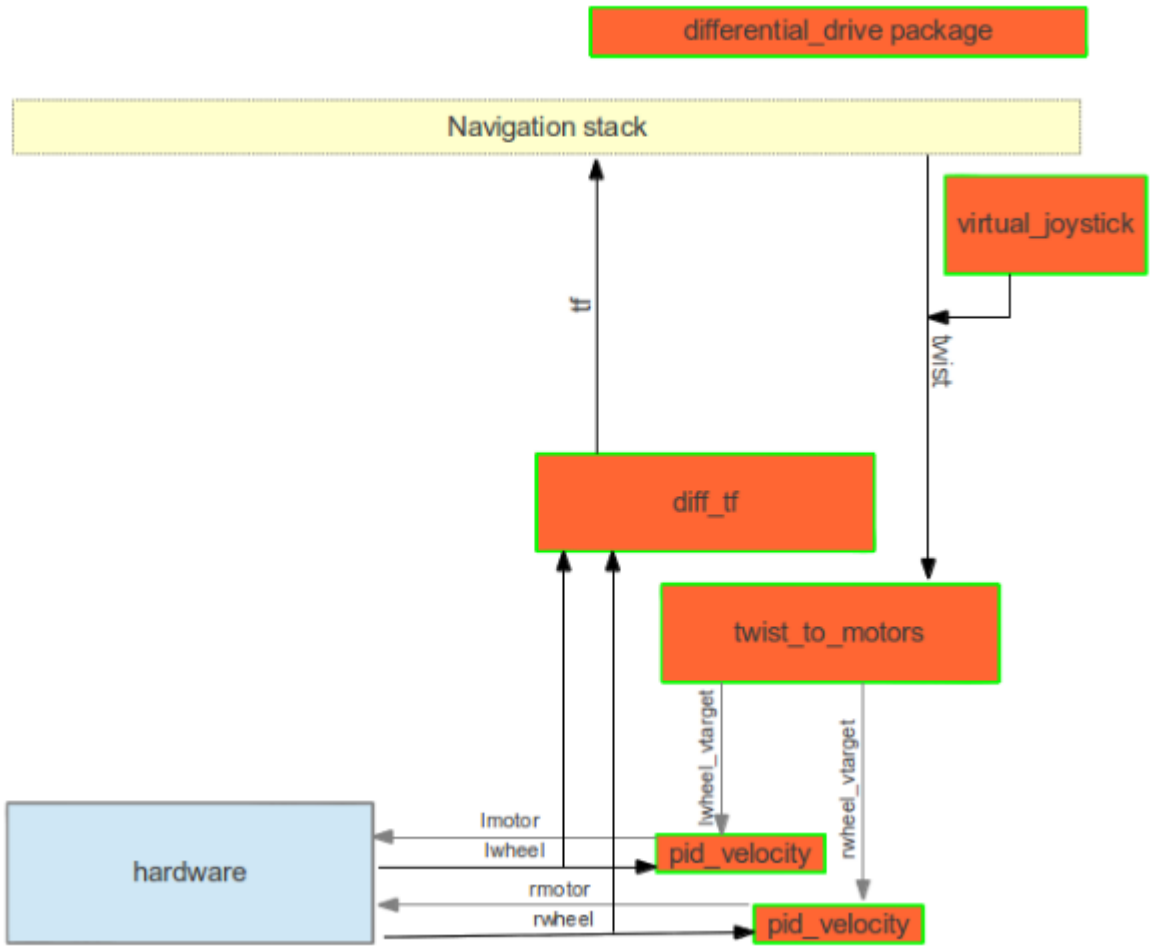
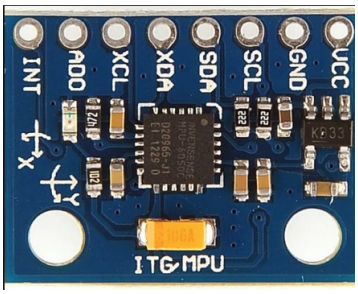


Figure 5.2.3: differential_drive package

- **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 linearvelocity and angular velocity(Twist) of the DD robot torequired 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
	pid_velocity	The controller using feedbackfrom encoder and set pointfrom twist_to_motors andoutputs the PWM to the motor	**~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 axestransformations of the DDrobot due to the movementwhich 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
			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	**_speed** is the required linear velocity
			_turn is the required angular velocity

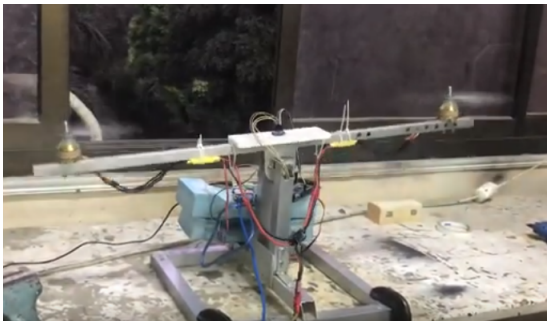
- topics

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.

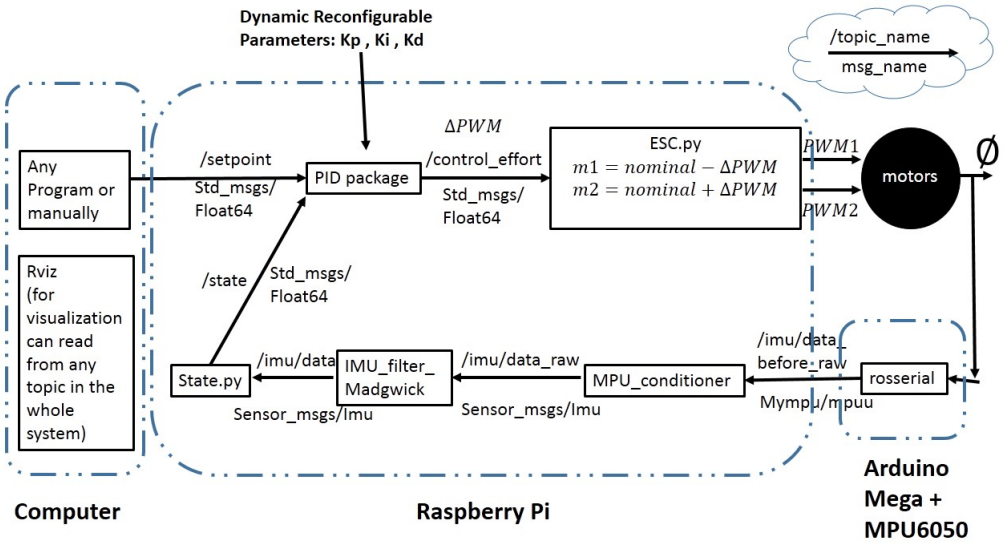


Figure 5.2.4: 1D Bi-rotor conf1 Block Diagram

- **Configuration 2 (myPIDA)**

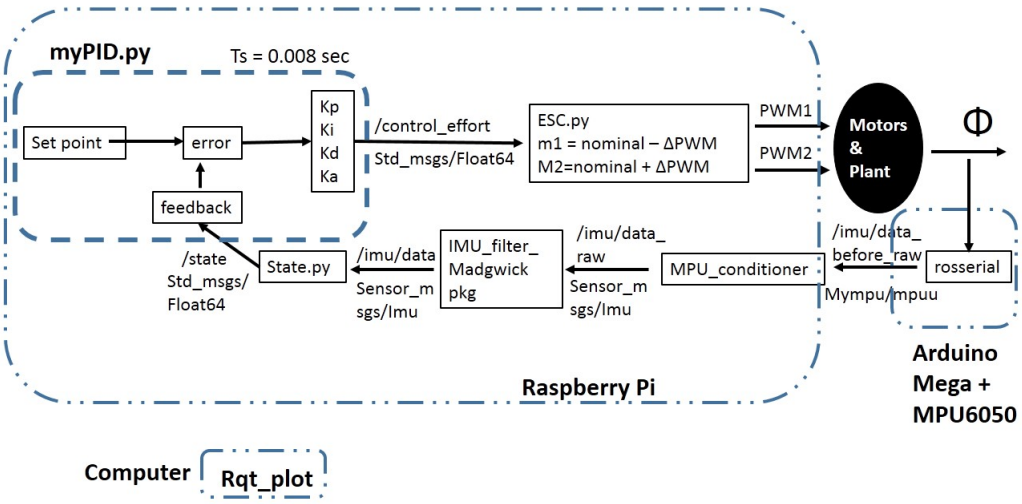


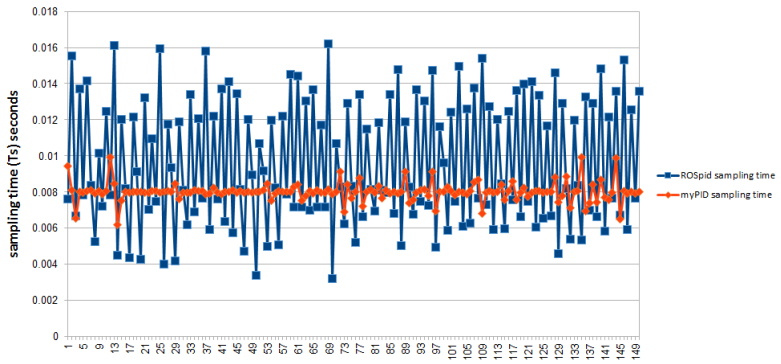
Figure 5.2.5: 1D Bi-rotor conf2 Block Diagram

Package	Node	Parameters
myPID	PID_node	Ts 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 Kp = some value , Ki = Kd = Ka = 0 , so the control should be = Kp * error = - Kp * 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, hiqh-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 .

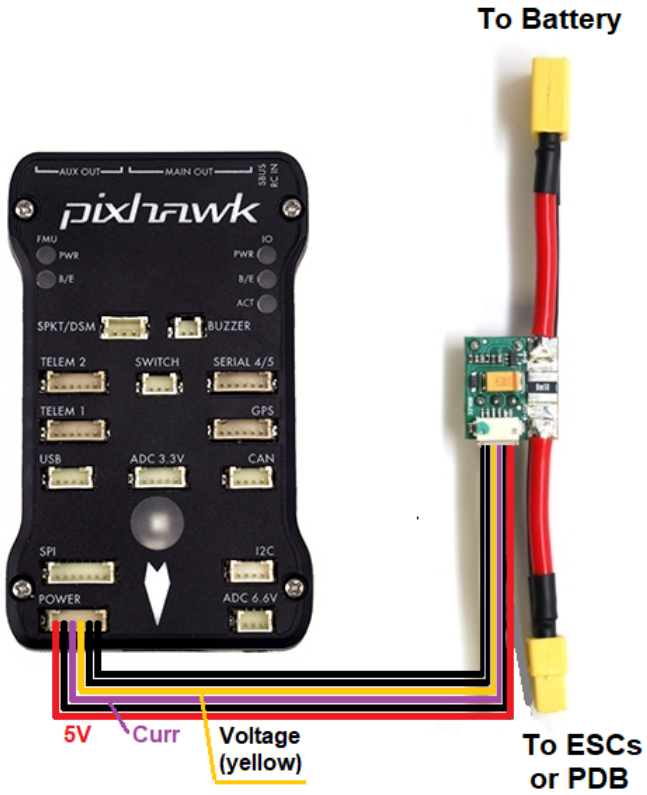
5.3.2 User guides to Pixhawk

- **Step 1: Wiring and Connections**

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

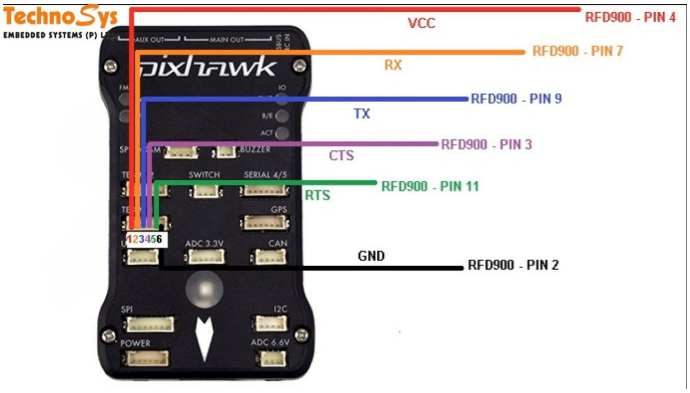
- Power module
- Telemetry
- PPM
- Flight controller and receiver
- Power Distribution Board (PDB)
- Jumper wires
- Android USB cable

- **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



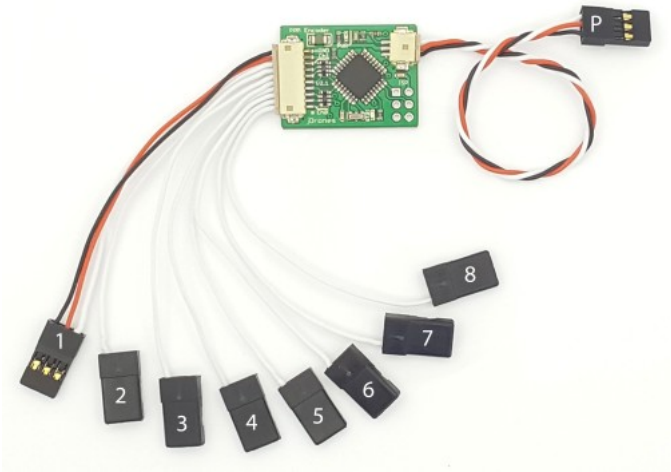
- **Telemetry:** The telemetry is divided into two parts , the first part is connected to the ground station to receive and send signals to the other part which is onboard to receive and send signals from and to the ground station .

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 he flight controller receiver to the Pixhawk.

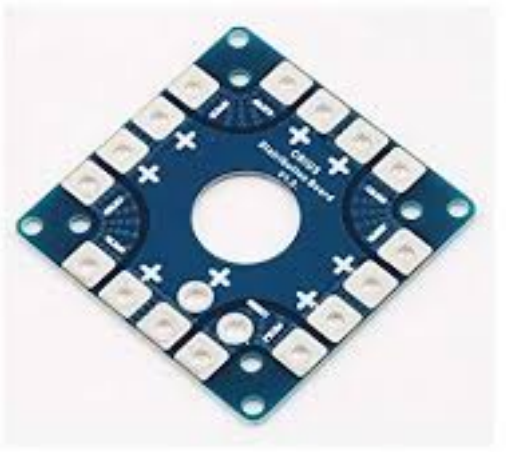


- **Flight controller and receiver:**

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 Qgroundconrol app to connect the Pixhawk to the ground station.

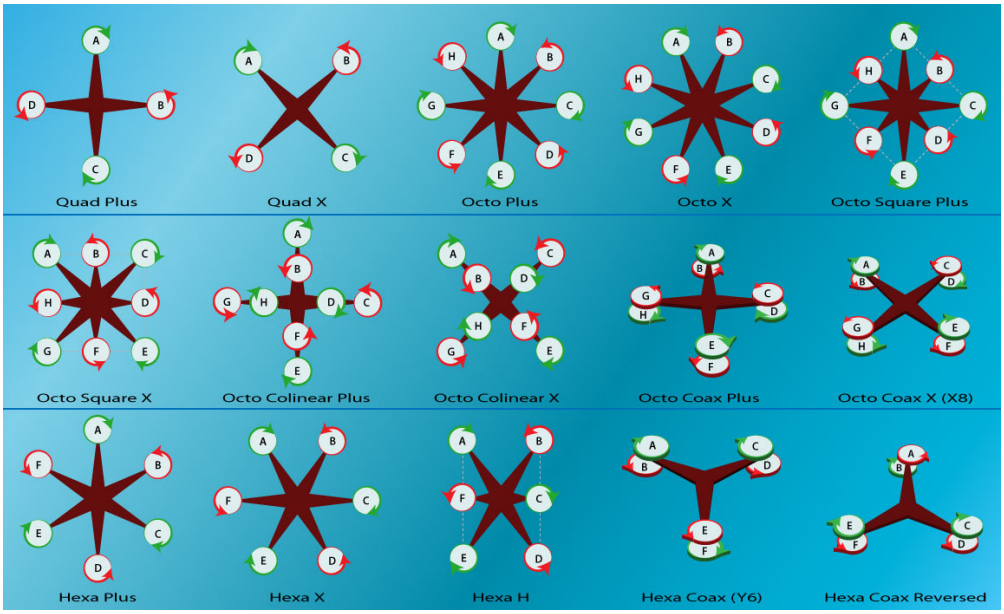
- **Step 4: Firmware selection**

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.

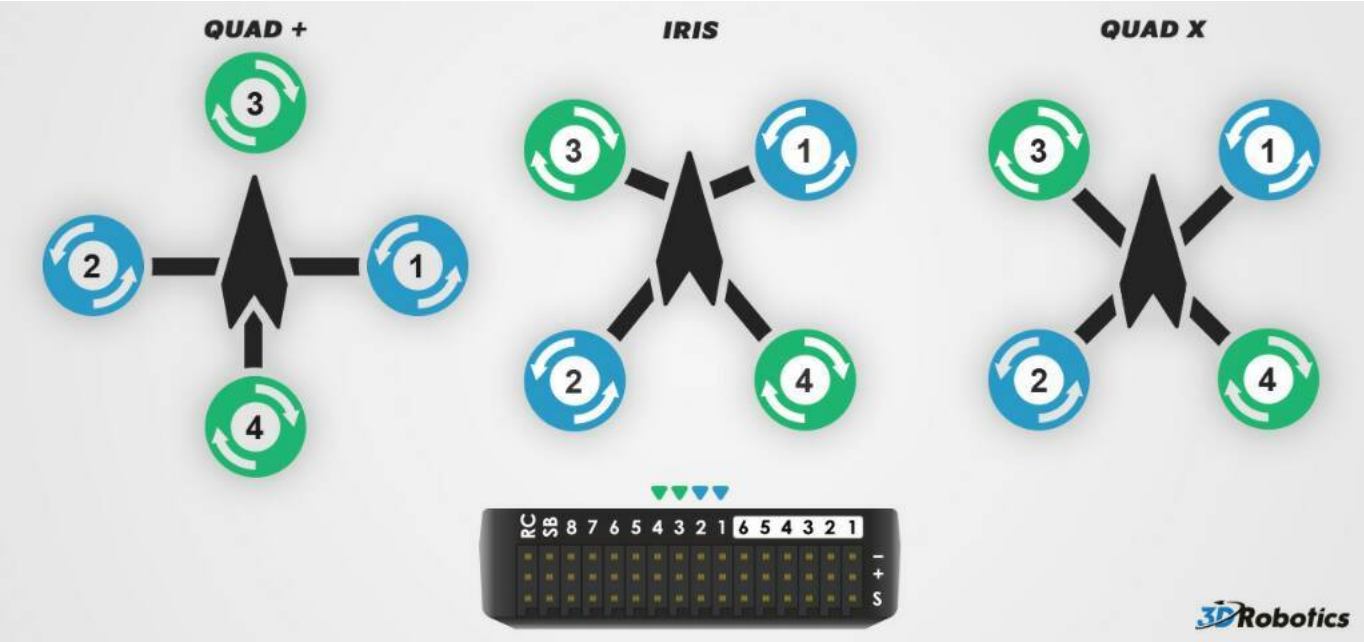
We will be using stabilize mode during testing and flight.

- **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.

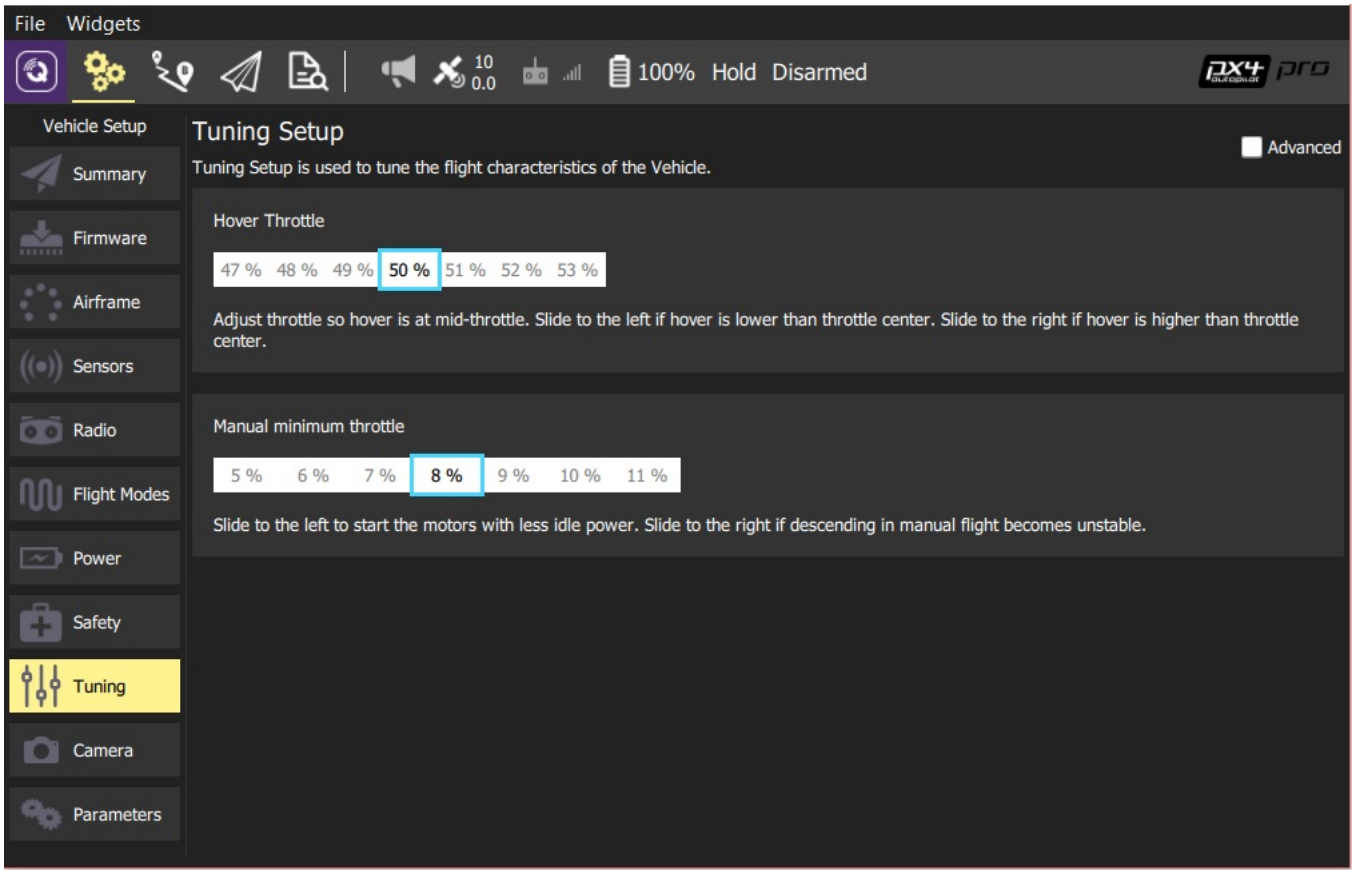
- Connect the PDB to the Power module (DO NOT CONNECT THE BATTERY OR POWER SUPPLY).
- Connect the signal pins of the ESCs to the Pixhawk output channels according to the order of the selected frame before.



- 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.

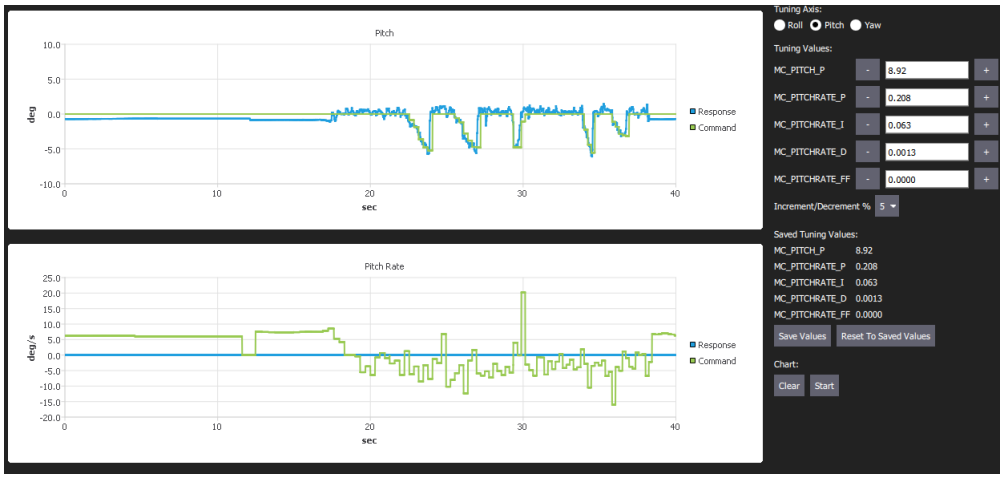
  - A low stabilize P will cause the copter to rotate very slowly and may cause the copter to feel unresponsive and could cause a crash if the wind disturbs it. Try lowering the RC_Feel parameter before lowering Stability P if smoother flight is desired.

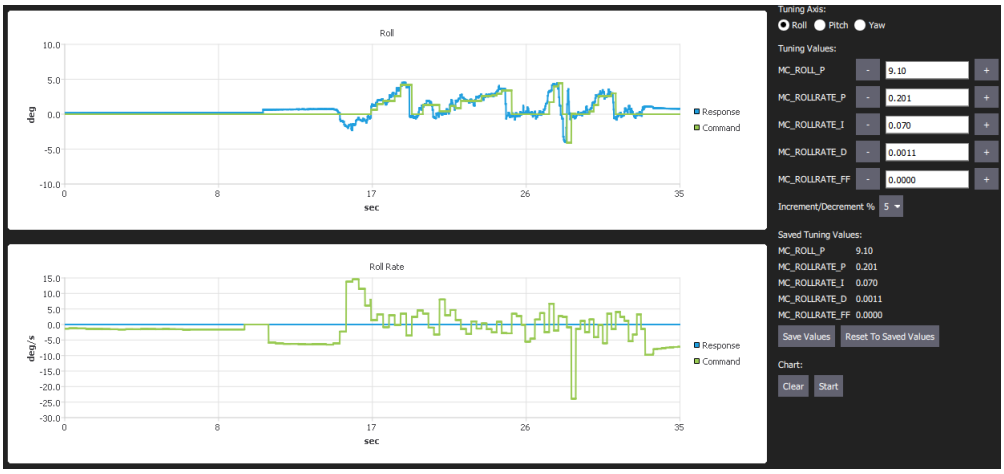
- Rate Roll/Pitch P, I and D terms control the output to the motors based on the desired rotation rate from the upper Stabilize (i.e. angular) controller. These terms are generally related to the power-to-weight ratio of the copter with more powerful copters requiring lower rate PID values. For example 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.

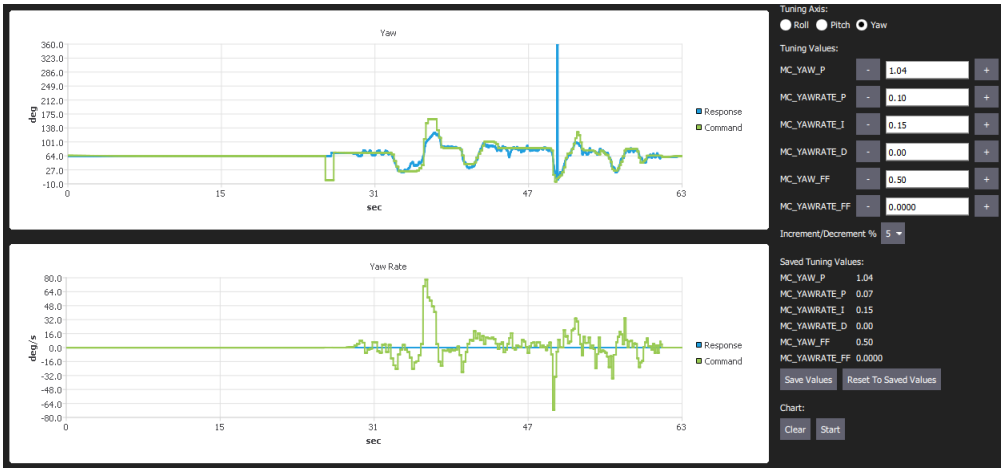
  - Rate Roll/Pitch P is the single most important value to tune correctly for your copter.
  - The higher the P the higher the motor response to achieve the desired turn rate.
  - Default is P = 0.15 for standard Copter.
  - Rate Roll/Pitch I is used to compensate for outside forces that would make your 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

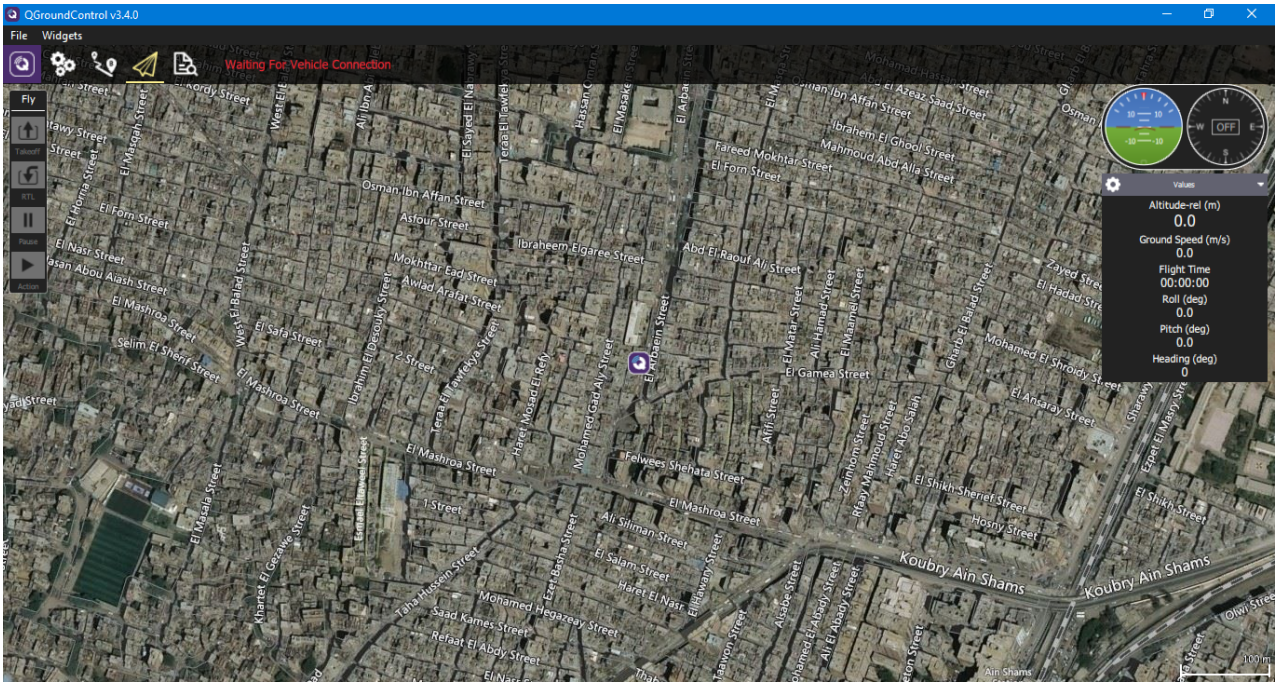
  - USB cable
  - Telemetry

  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

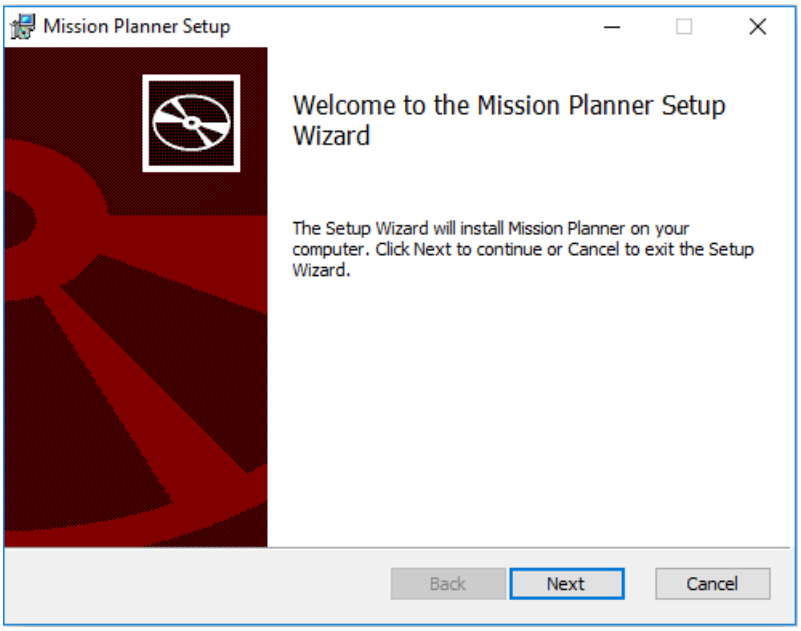
Mission Planner is a ground control station for Plane, Copter and Rover. It is compatible with Windows only. Mission Planner can be used as a configuration utility or as a dynamic control supplement for your autonomous vehicle. Here are just a few things you can do with Mission Planner:

- 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 you 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 the 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 it's 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.



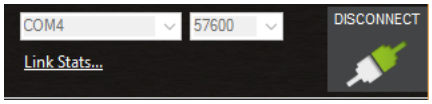
- Step 2: Connect to Pixhawk

  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:

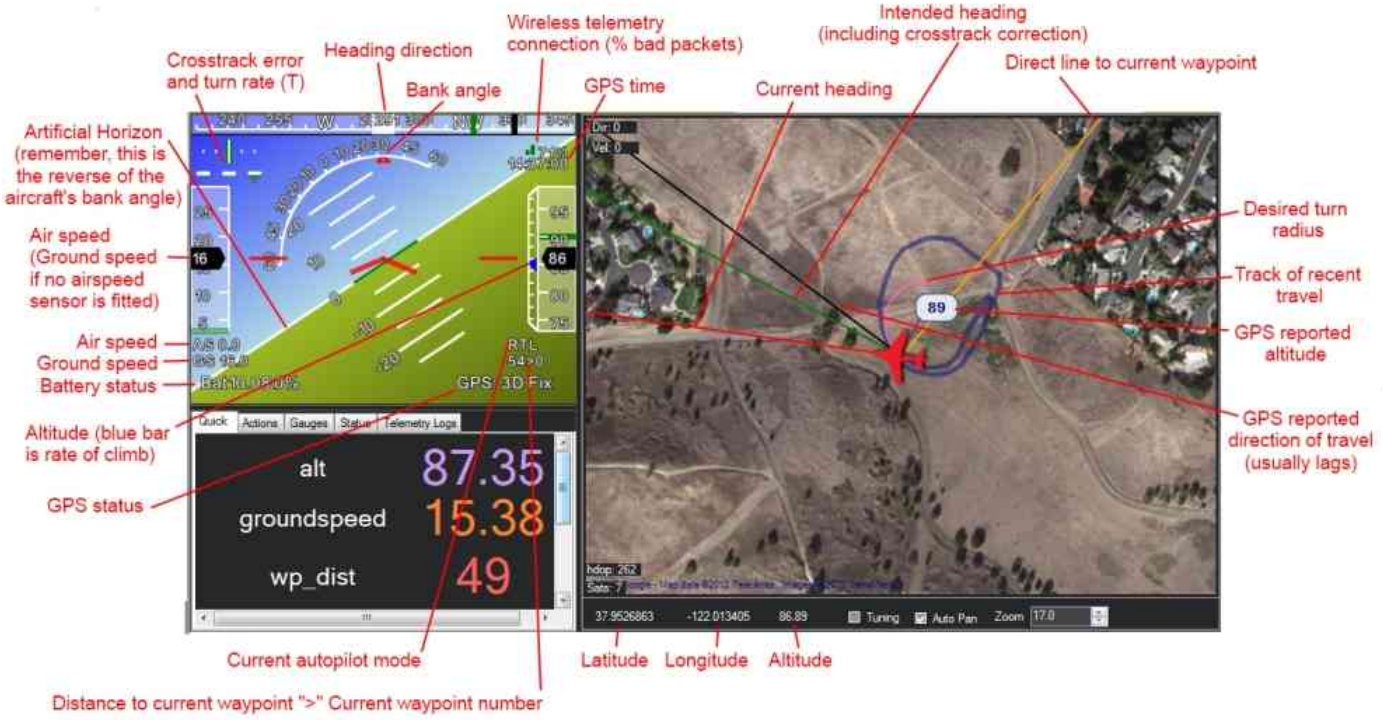


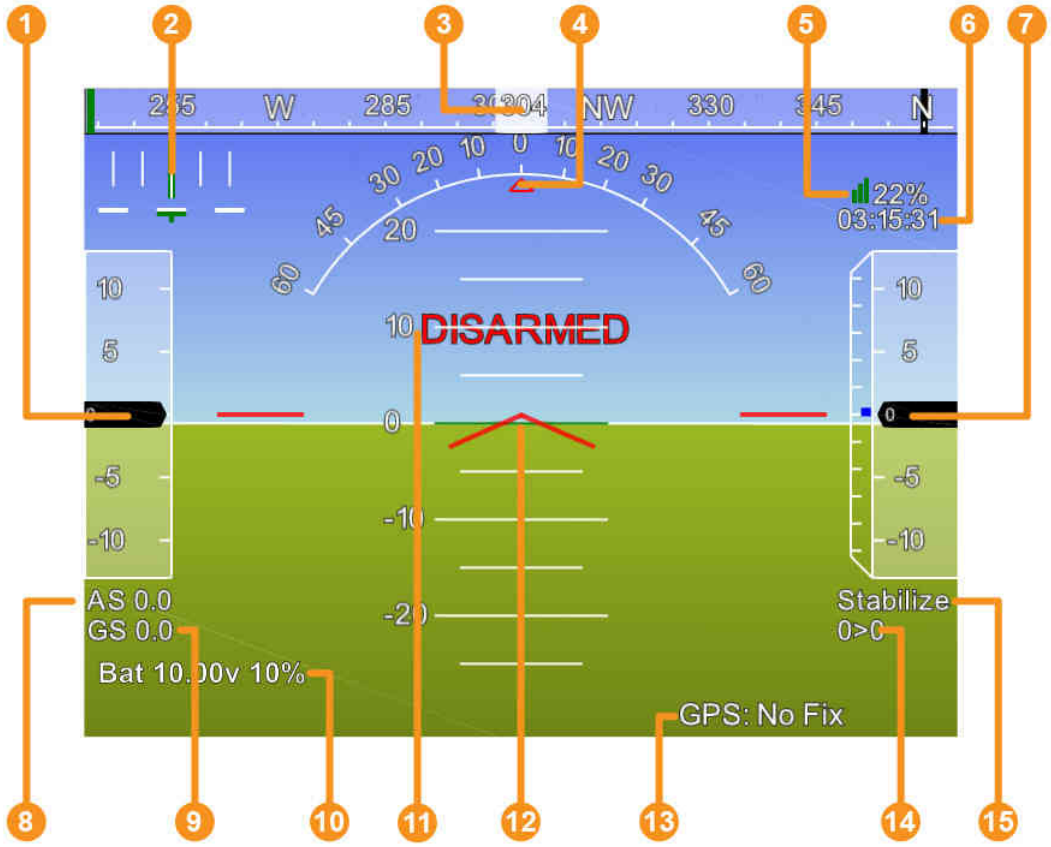
5.4.5 User interface of Mission Planner

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