

# Urban UAV using vision navigation (Flying Taxi)

Submitted to:

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

By:

June 26, 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.

---

# Preface

## Project Structure

This project is a new chapter of the cooperation between Boeing and Aerospace Department at Cairo University, This project aims to build a (VTOL) flying taxi prototype which can carry one passenger. In order to do that the work was divided in two paths: the design and build of a VTOL vehicle which can satisfy certain ODD, and the second path is related to the visual SLAM using stereo camera and the ROS working frame which we used to build our software architecture.

## Chapters Overview

this Thesis is divided into five chapters each one is related to certain topic and discuss it completely with full details started with Introduction followed by how to implement the module.

- **Chapter 1** : is related to the structural design and the stability analysis as we had to build the model by ourself so we have to analyse it using computer software before starting to build it.
- **Chapter 2** : in a parallel way we had build the mathematical model of the vehicle so we can have a characteristics that we use in the control loops to get the controller gains.
- **Chapter 3** : a comparison between self builded Autopilot and the commercial Autopilot we use.
- **Chapter 4** : a vision navigation and obstacle avoidance is the main core of the software architecture in this chapter we discuss the difference between the vision sensors and the difference between the obstacle avoidance algorithms and the one we used.
- **Chapter 5** : this chapter include complete explain to the ROS and the px4 autopilot and how we used them.

---

# 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 Invariant
PID	Proportional-Integral-Derivative Control
GPU	Graphical Processing Unit
ODD	Operational Design Domain
MC	Multicopter

# Contents

<b>1</b>	<b>Design</b>	<b>6</b>
1.1	Mission . . . . .	6
1.1.1	Payload . . . . .	6
1.1.1.1	Control payload . . . . .	6
1.1.1.2	Passenger payload . . . . .	6
1.2	Aerodynamic Design . . . . .	7
1.2.1	Introduction . . . . .	7
1.2.2	Design Plan . . . . .	7
1.2.2.1	Final results . . . . .	11
1.2.3	Stability Analysis . . . . .	12
1.2.3.1	For longitudinal . . . . .	12
1.2.3.2	For lateral . . . . .	13
1.3	Structural Design . . . . .	15
1.3.1	Building a 3D model . . . . .	15
1.3.1.1	solidWorks outputs: . . . . .	15
1.3.1.2	Steps to draw a 3D model in SolidWorks . . . . .	15
1.3.2	Components . . . . .	15
1.3.3	Wing fixation method . . . . .	17
1.3.3.1	Variable incidence angle fixation method . . . . .	17
1.3.3.2	Constant incidence angle fixation method . . . . .	20
1.3.4	Fuselage . . . . .	20
1.3.5	Propulsion system selection . . . . .	20
1.3.5.1	Vertical takeoff propulsion system . . . . .	20
1.3.5.2	Cruise propulsion system . . . . .	21
1.3.5.3	Power source (battery) . . . . .	22
<b>2</b>	<b>Mathematical Model</b>	<b>23</b>
2.1	Equations of Motion(Rigid-Body dynamics) . . . . .	23
2.1.1	Kinetics ( the study of forces and moments on system in motion) . . . . .	23
2.1.2	Kinematics (the study of motion without regard to forces or moments) . . . . .	24
2.1.3	Resultant Nonlinear model . . . . .	25
2.2	Linearization . . . . .	25
2.3	Parameters . . . . .	26
<b>3</b>	<b>Autopilot</b>	<b>27</b>
3.1	Autopilot design . . . . .	27
3.1.1	Attitude controller . . . . .	27
3.1.2	Altitude Controller . . . . .	32
3.1.3	Attitude & Altitude controllers together in action . . . . .	34
3.1.4	Line-of-sight (LOS) Guidance . . . . .	35
3.1.5	Simulation using LabVIEW . . . . .	42
3.2	Commercial Autopilot . . . . .	44
3.2.1	Commander . . . . .	45
3.2.2	Navigator . . . . .	45
3.2.3	Position controller . . . . .	45
3.2.4	Attitude & Rate Controller . . . . .	47
3.2.5	Estimator . . . . .	49

3.2.6	Output drivers . . . . .	49
3.3	Gains Comparison . . . . .	50
<b>4</b>	<b>Vision navigation and Obstacle Avoidance</b>	<b>51</b>
4.1	Stereo Vision . . . . .	51
4.2	Autonomous robots architecture . . . . .	58
4.3	Obstacle Avoidance . . . . .	58
4.3.1	Local methods . . . . .	58
<b>5</b>	<b>Implementation</b>	<b>60</b>
5.1	ROS . . . . .	60
5.1.1	What is ROS . . . . .	60
5.1.1.1	History of ROS . . . . .	60
5.1.1.2	ROS Philosophy . . . . .	60
5.1.2	Pros and Cons of ROS . . . . .	62
5.1.2.1	Advantages: . . . . .	62
5.1.2.2	Disadvantages: . . . . .	62
5.1.3	Example . . . . .	62
5.1.4	Summary of some important features in ROS . . . . .	64
5.2	Proof of Concept . . . . .	70
5.2.1	State Estimation (Rover) . . . . .	70
5.2.1.1	Software . . . . .	71
5.2.1.2	Localization . . . . .	72
5.2.2	Control (half quad) . . . . .	73
5.2.2.1	Results & Conclusions . . . . .	76
5.3	px4 and Mavros . . . . .	77
5.3.1	Introduction . . . . .	77
5.3.2	User guides to Pixhawk . . . . .	78
<b>A</b>	<b>ROS Examples</b>	<b>83</b>
A.1	publisher.py . . . . .	83
A.2	subscriber.py . . . . .	83
A.3	publisher.cpp . . . . .	83
A.4	subscriber.cpp . . . . .	83
A.5	ROS services using python . . . . .	83
A.5.1	Server . . . . .	83
A.5.2	node uses server . . . . .	83
A.6	ROS actions using python . . . . .	83
A.6.1	Action server . . . . .	83
A.6.2	node uses action . . . . .	83

# List of Figures

0.0.1 BOEING'S flying taxi . . . . .	5
1.2.1 Tandem wing plane . . . . .	7
1.2.2 The 2 airfoils . . . . .	7
1.2.3 Pressure distribution along NACA0015 with difference angle of attack . . . . .	8
1.2.4 Pressure distribution along Eppler423 with difference angle of attack . . . . .	8
1.2.5 Polar graph of NACA0015 . . . . .	8
1.2.6 Polar graph of Eppler423 . . . . .	8
1.2.7 Configuration . . . . .	8
1.2.8 NACA0015 Lift and Cm-alpha . . . . .	9
1.2.9 Eppler423 Lift and Cm . . . . .	9
1.2.10 Lift Vs alpha and Cm Vs alpha . . . . .	11
1.2.11 Cl Vs alpha and Cl/Cd Vs alpha . . . . .	11
1.2.12 $Cl^{3/2}/Cd$ Vs alpha . . . . .	11
1.2.13 Fz vs alpha and Cm vs alpha . . . . .	11
1.2.14 zero pole map for Longitudinal mode . . . . .	12
1.2.15 zero pole map for Lateral mode . . . . .	13
1.2.16 Time Response . . . . .	14
1.3.1 Pixhawk 3D model . . . . .	15
1.3.2 Pixhawk mount 3D model . . . . .	16
1.3.3 ZED stereo camera 3D model . . . . .	16
1.3.4 Hovering quad motor 3D model . . . . .	17
1.3.5 Pusher motor 3D model . . . . .	17
1.3.6 Battery 3D model . . . . .	17
1.3.7 variable incidence fixation assembly . . . . .	18
1.3.8 variable incidence fixation Front view . . . . .	18
1.3.9 variable incidence fixation at zero incidence . . . . .	19
1.3.10 variable incidence fixation at -4 degrees incidence . . . . .	19
1.3.11 variable incidence fixation at 4 degrees incidence . . . . .	20
1.3.12 Constant incidence fixation at 2 degrees incidence . . . . .	20
1.3.13 KDE2315XF-965 . . . . .	21
1.3.14 Hover motor's preformance data . . . . .	21
1.3.15 Rimfire .10 35-30-1250 Outrunner Brushless . . . . .	21
1.3.16 Lithium Polymer Battery (11.1 V, 5200 mAH- 35C) . . . . .	22
3.1.1 Autopilot design . . . . .	27
3.1.2 Roll Controller . . . . .	28
3.1.3 Roll Step response and control action . . . . .	28
3.1.4 5 deg desired Roll input . . . . .	29
3.1.5 Pitch Controller . . . . .	29
3.1.6 Pitch Step response and control action . . . . .	30
3.1.7 5 deg desired pitch input . . . . .	30
3.1.8 Yaw Controller . . . . .	31
3.1.9 Yaw Step response and control action . . . . .	31
3.1.10 5 deg desired yaw input . . . . .	32
3.1.11 desired roll = 5 deg , desired pitch = 0 , desired yaw = 5 . . . . .	32
3.1.12 Altitude Controller . . . . .	33

3.1.13Altitude Step response and control action . . . . .	33
3.1.14 deg desired yaw input . . . . .	34
3.1.15desired roll = -5 deg , desired pitch = -5 deg , desired altitude = -4 m ( Z = 4 m) . . . .	34
3.1.16LOS . . . . .	35
3.1.17LOS block diagram . . . . .	36
3.1.18 guidance . . . . .	36
3.1.19long-track control (trial : 1) . . . . .	37
3.1.20LabVIEW front panel . . . . .	37
3.1.21long-track control (trial : 2 increasing P) : high steady-state error . . . . .	38
3.1.22long-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) . . . . .	38
3.1.23long-track control (trial : 3 adding small I term to eliminate steady-state error) : settling time is highly increased . . . . .	39
3.1.24long-track error control (final trial) : increasing P and D . . . . .	39
3.1.25 guidance . . . . .	40
3.1.26X,Y guidance waypoint(X=0 , Y=1 , Z=-1) . . . . .	40
3.1.27X,Y Guidance waypoint(X=1 , Y=1 , Z=-1) . . . . .	41
3.1.28LabVIEW front panel . . . . .	42
3.1.29LabVIEW blockdiagram . . . . .	42
3.1.30LabVIEW Guidance blockdiagram . . . . .	43
3.1.31LabVIEW altitude BD . . . . .	43
3.1.32LabVIEW attitude controller BD . . . . .	44
3.2.1 PX4 architecture from PX4 website . . . . .	45
3.2.2 cascaded control architecture from “Nonlinear Quadcopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello,2013” . . . . .	46
3.2.3 Position Controller . . . . .	47
3.2.4 cascaded control architecture from “Nonlinear Quadcopter Attitude Control Technical Report,Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello,2013” . . . . .	47
3.2.5 Attitude&rate controller . . . . .	49
4.1.1 Ref[?] MAV uses 4 cameras to get pose estimate real-time on-board . . . . .	55
4.1.2 2 forward-facing stereo configuration . . . . .	55
4.1.3 final configuration . . . . .	57
5.1.1 ROS architecture . . . . .	60
5.1.2 Why ROS? . . . . .	61
5.1.3 ROS graph of Stanford’s STAIR robot nodes. . . . .	61
5.1.4 ROS example . . . . .	63
5.1.5 ROS msgs . . . . .	67
5.1.6 ROS actions . . . . .	69
5.1.7 comparison between ROS parameters, Dynamic Reconfigure , Topics , Services , and Actions	69
5.2.1 Differential Drive Robot (DD robot) . . . . .	70
5.2.2 differential drive model . . . . .	70
5.2.3 differential_drive package . . . . .	70
5.2.4 1D Bi-rotor conf1 Block Diagram . . . . .	73
5.2.5 1D Bi-rotor conf2 Block Diagram . . . . .	75



# List of Tables

1.1.1 components masses . . . . .	6
1.2.1 <b>comparison between 2 airfoils</b> . . . . .	10
1.3.1 Hover motor specifications . . . . .	21
1.3.2 power of motors to vehicle's weight . . . . .	21
1.3.3 Pusher motor specifications . . . . .	22
1.3.4 Battery specification . . . . .	22
4.1.1 different sensors for vision . . . . .	52
4.1.2 ZED stereo camera vs Kinect . . . . .	54

# Motivation

A vertical take-off and landing (VTOL) aircraft is one that can hover, take off, and land vertically. This classification can include a variety of types of aircraft including fixed-wing aircraft as well as helicopters and other aircraft with powered rotors.

Some VTOL aircraft can operate in other modes as well, such as CTOL (conventional take-off and landing), STOL (short take-off and landing), and/or STOVL (short take-off and vertical landing). Others, such as some helicopters, can only operate by VTOL, due to the aircraft lacking landing gear that can handle horizontal motion. VTOL is a subset of V/STOL (vertical and/or short take-off and landing). Some lighter-than-air aircraft also qualify as VTOL aircraft, as they can hover, takeoff, and land with vertical approach/departure profiles.

Electric and hybrid-electric vertical takeoff and landing aircraft eVTOL is a technology being used in the quest for fully autonomous passenger air vehicles (PAV).

So why VTOL ?

vertical take off and landing, enables a traveler to travel directly from point A to point B, instead of going from point A to an airport in a car say, then fly from the airport to another airport, and then drive with a car from the other airport to point B. Not only in travel is VTOL very useful, in rescue operations as well as in military and law enforcement applications it is often

## • Advantages of VTOL

VTOL is challenging. It is far more technically difficult than conventional winged flight, but in exchange, it has several advantages.

1. **Time and fuel savings:** Vertical take off and landing enables an aerial vehicle to take off from many more locations than an aircraft that requires an airport. This means that the journey to and from the airport can be eliminated, saving time and fuel, and the researchers in US found out that A person in the US lives an average of 17 miles away to the next airport. But he lives on average less than 1 mile away from a location that is suitable for vertical take-off and landing of a small civilian VTOL.
2. **Efficiency and speed:** For personal aircraft the the wing is typically constructed with a compromise in mind. Typically it is desired that personal airplanes can take off and land at fairly low speeds such as 100kph , so that they can use small regional airfields, for example. On the other hand speed and efficiency are desired, but an overly large wing is a hindrance to this. Since lift (like drag) increases with speed, a very large wing at cruise speed generates more lift than needed, requiring the pilot to trim the elevator down, causing drag. The large wing itself, is also causing drag due to skin friction, and the larger it is, the more 'skin' is in contact with the air, the more drag is caused, the slower the airplane can fly and the more fuel it burns. A VTOL plane can be built such that its wings are optimized strictly for speed and efficiency since take off and landing are not handled by the wing. If the VTOL components in turn do not cause much additional drag, then real net drag reductions can be achieved making the plane faster and more efficient than even a pure aircraft.
3. **Safety:** More than 85% of all airplane accidents (and fatalities) happen during their conventional take off and landing. The initial climb and the final descent are the most dangerous segments of a flight. An aircraft is safest when cruising at high altitude. It is most vulnerable when flying close to the ground or rolling on the ground at high speeds because the margin for error and error recovery is small to non-existent. This is why take off and landings are comparatively dangerous, even on airports with their vastly long and empty runways. Vertical take off and landing greatly reduces this risk because the plane is accelerated and decelerated while high in the air adding margin for error and recovery. This makes already safe air travel even safer.

- **Different types of VTOL**

1. **Rotorcraft** , or rotary wing aircraft, are those that use lift generated by rotor blades spinning around a central mast, so helicopters, quadcopters and gyrocopters.
2. **Powered-lift vehicles** are those that take off and land vertically but perform differently from rotorcraft when in flight. They typically have a more conventional fixed wing plane design. Examples include convertiplanes such as the Bell Boeing V-22 Osprey, which takes off and lands vertically but uses fixed wing lift in normal flight.

- **The future of VTOL**

the VTOL aircraft has the potential to change the course of aviation history and create a new chapter in modern day travel.

- A number of firms are developing aircraft that use a VTOL system. Despite a flood of recent controversies, ride-sharing app firm Uber has pledged to launch its first flying taxis in 2020 using VTOL.
- Munich-based aviation startup Lilium aims to offer an on-demand flying taxi service that it claims will be five times faster than traveling in a car. The current prototype is a two-seater aircraft shaped like a conventional plane that uses a VTOL system.
- at the 2017 Geneva Motor Show, Airbus showcased a prototype flying hybrid car. The modular vehicle can disconnect from its wheels, after which it is picked up by a flying set of rotors.
- Nasa has developed the battery-powered GL-10, which take off and land vertically but flies efficiently like a conventional plane.

**FLYING TAXI** An air taxi is a small commercial aircraft which makes short flights on demand, In 2001 air taxi operations were promoted in the United States by a NASA and aerospace industry study on the potential Small Aircraft Transportation System (SATS) and the rise of light-jet aircraft manufacturing, the Flying taxi is a direct application of the VTOL which will have a great impact in the future, many companies now like: Google, Uber and Audi ,..etc invest much money in the self-driving cars and they achieved good steps, On the other hand part of these companies in addition to the air transport companies think in another way to build a flying vehicle which can carry a person from point A and reach point B ,Actually by the start of 2019 Boeing released their first self-flying taxi which completed its first flight successfully.

So it's obvious now that we are a step far from a new technology in the Air transport that will make air transportation not limited for the Traveling between countries, it's a new fast growing path which means that we need more aerospace engineers who are aware about many concepts like path planning, state estimation, computer vision ,..etc.

Of course they are many difficulties face this technology , technical difficulties for example the test experiments as it cost too much and other difficulties related to the laws and how much people will trust an autonomous flying vehicle.

../ch1 Introduction/190123140308-02-boeing-vtol-exlarge-169.jp

Figure 0.0.1: BOEING'S flying taxi

# Chapter 1

## Design

### 1.1 Mission

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

#### 1.1.1 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

##### 1.1.1.1 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

##### 1.1.1.2 Passenger payload

comfortable chair

- components weight estimation

components	masses (gm)	quantity
Pixhawk	37	1
Mini Jetson KIT	150	1
RP Lidar A1	190	1
ZED camera	159	1
Quad motors		4
Quad propellers		4
Quad ESC		4
Pusher motor		1
Pusher propeller		1
Pusher ESC		1
Battery	380	1

Table 1.1.1: components masses

## 1.2 Aerodynamic Design

### 1.2.1 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, Reynold'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 ( $\delta = \alpha^w - \alpha^p$ ): The relative angle of attack between two angles of attack for each wing.

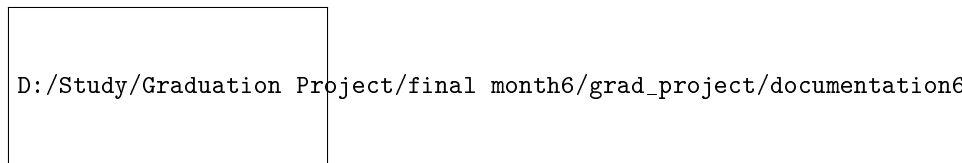


Figure 1.2.1: Tandem wing plane

### 1.2.2 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.

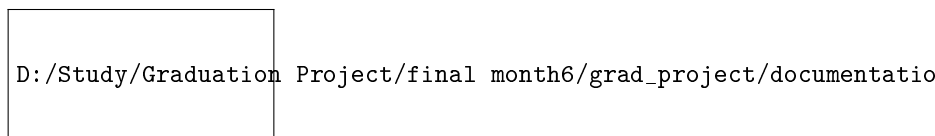
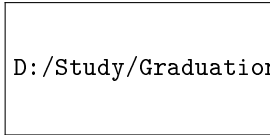


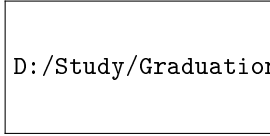
Figure 1.2.2: The 2 airfoils

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:



D:/Study/Graduation Project/final month6/grad\_project/documentation

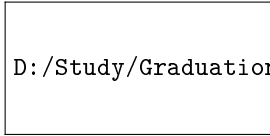
Figure 1.2.3: Pressure distribution along NACA0015 with difference angle of attack



D:/Study/Graduation Project/final month6/grad\_project/documentation

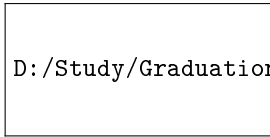
Figure 1.2.4: Pressure distribution along Eppler423 with difference angle of attack

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



D:/Study/Graduation Project/final month6/grad\_project/documentation

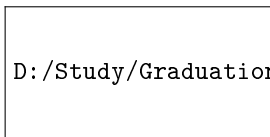
Figure 1.2.5: Polar graph of NACA0015



D:/Study/Graduation Project/final month6/grad\_project/documentation

Figure 1.2.6: Polar graph of Eppler423

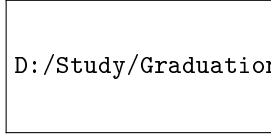
Now we ready to build design after prepared airfoils, after trying different dimensions we get the best configuration then check stability and lift produced by vehicle we make analysis by different stream speed and found there isn't any effect on stability but affected at lift produced over Airfoil but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



D:/Study/Graduation Project/final month6/grad\_project/documentation

Figure 1.2.7: Configuration

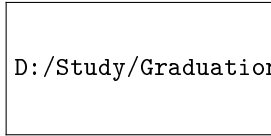
- NACA0015



D:/Study/Graduation Project/final month6/grad\_project/documentation

Figure 1.2.8: NACA0015 Lift and Cm-alpha

- Eppler423



D:/Study/Graduation Project/final month6/grad\_project/documentation

Figure 1.2.9: 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 1.2.1: **comparison between 2 airfoils**

- Conclusion of results:

1. 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.
2. 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%.
3. For Lift: We find trim angle of attack at 4.5 degree, and we go to  $F_z$  VS  $\alpha$  graph get lift that carry aircraft we get  $F_z=28(N)$  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) , 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:





### 1.2.3 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

#### 1.2.3.1 For longitudinal

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

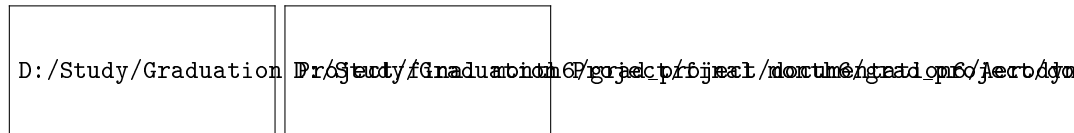


Figure 1.2.14: zero pole map for Longitudinal mode

- 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	

**1.2.3.2 For lateral**

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

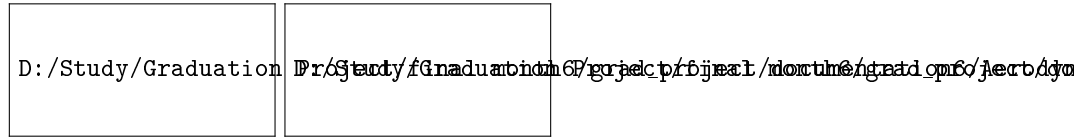


Figure 1.2.15: zero pole map for Lateral mode

- **Lateral derivatives from Xflr5**

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

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

CXu = -0.05768	CYb = -0.45694
CLu = 0.00031	Clb = -0.05641
Cmu = 0.00000	Cnb = 0.27372
CXa = 0.10753	CYp = -0.05404
CLa = 5.46422	Clp = -0.51794
Cma = -1.09910	Cnp = -0.05093
CXq = -0.49272	CYr = 0.62552
CLq = 12.54598	Clr = 0.21810
Cmq = -16.59853	Cnr = -0.37974

- **Time response characteristics**

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

D:/Study/Graduation Project/final month6/grad\_project/documentation

Figure 1.2.16: Time Response

## 1.3 Structural Design

### 1.3.1 Building a 3D model

Solid Works is used to create a 3D model and AutoCAD is used to edit 2D sketches for machining purposes

#### 1.3.1.1 SolidWorks outputs:

1. Files used in machining
2. Accurate weight estimation
3. Gravity center location control
4. Some analysis (flow and structural)

#### 1.3.1.2 Steps to draw a 3D model in SolidWorks

1. Define important dimensions created through aerodynamic analysis process such as: arm, span, chord ...
2. Define CG position needed for aerodynamic stability
3. Define components of the payload, their dimensions, weight and fixation points
4. define the components that needs mounts
5. Clearance needed for wiring and connections

### 1.3.2 Components

- 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

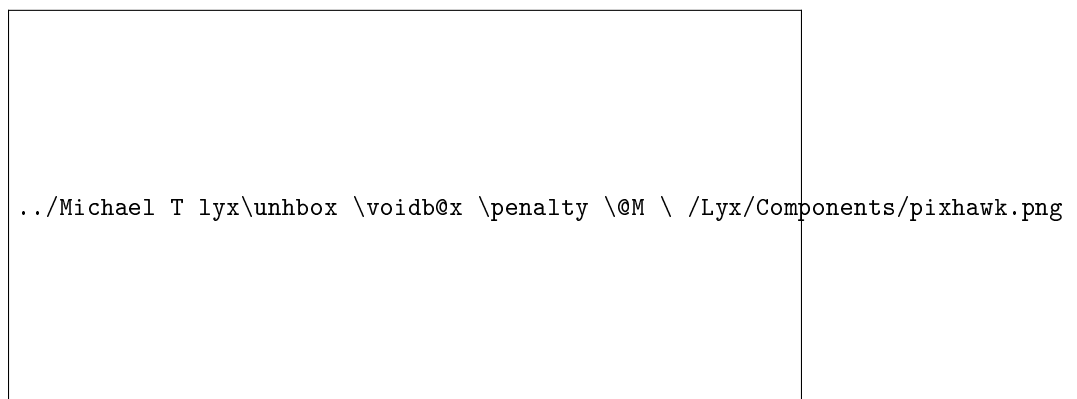


Figure 1.3.1: Pixhawk 3D model

- Pixhawk mount

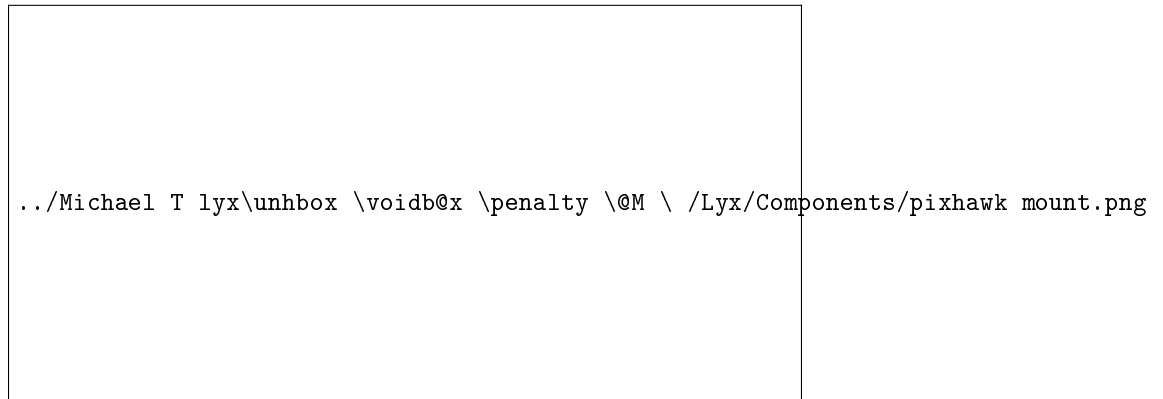


Figure 1.3.2: Pixhawk mount 3D model

- ZED camera

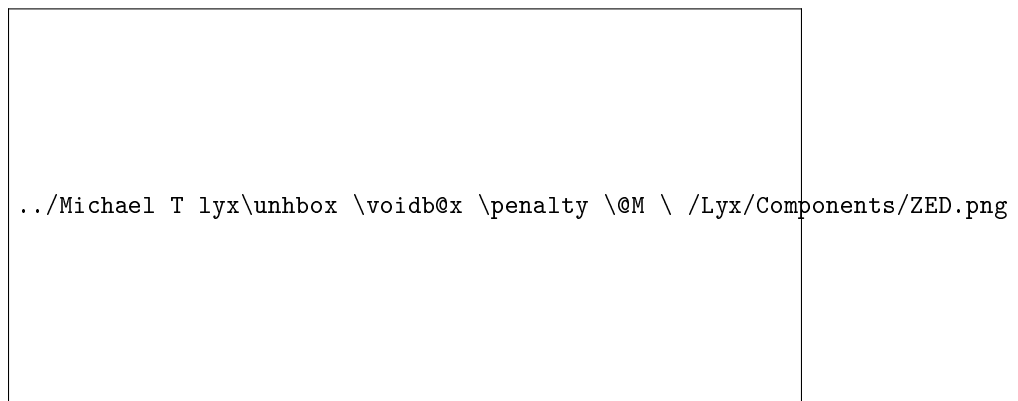


Figure 1.3.3: ZED stereo camera 3D model

- Jetson nano

- Quad Motor

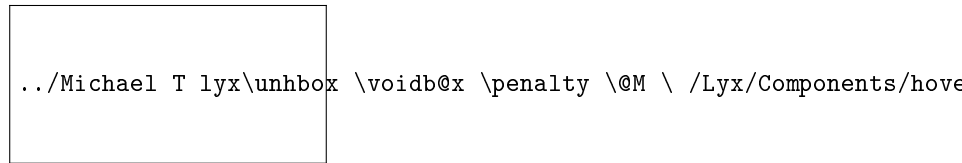


Figure 1.3.4: Hovering quad motor 3D model

- Pusher motor

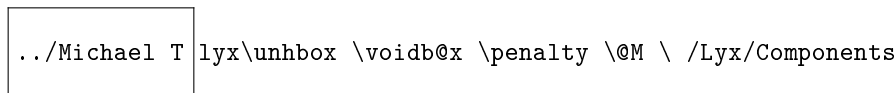


Figure 1.3.5: Pusher motor 3D model

- Battery

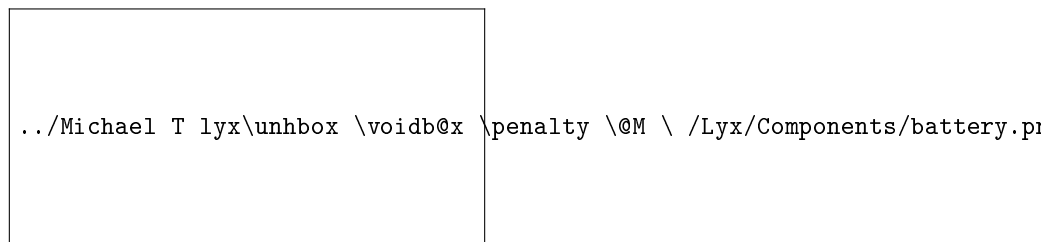


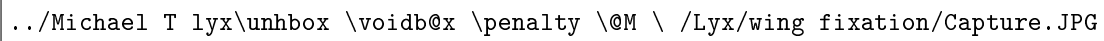
Figure 1.3.6: Battery 3D model

### 1.3.3 Wing fixation method

#### 1.3.3.1 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:



../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/Capture.JPG

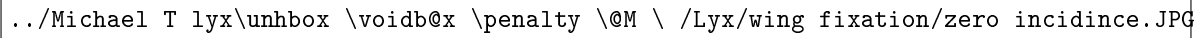
Figure 1.3.7: variable incidence fixation assembly



../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/explanition incidence.jpg

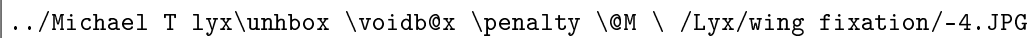
Figure 1.3.8: variable incidence fixation Front view





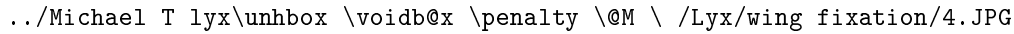
../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/zero incidence.JPG

Figure 1.3.9: variable incidence fixation at zero incidence



../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/-4.JPG

Figure 1.3.10: variable incidence fixation at -4 degrees incidence

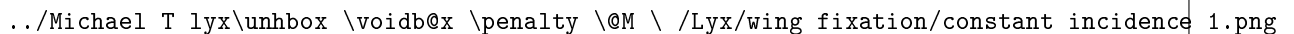


../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/4.JPG

Figure 1.3.11: variable incidence fixation at 4 degrees incidence

### 1.3.3.2 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:



../Michael T lyx\unhbox \voidb@x \penalty \@M \ /Lyx/wing fixation/constant incidence 1.png

Figure 1.3.12: Constant incidence fixation at 2 degrees incidence

## 1.3.4 Fuselage

### 1.3.5 Propulsion system selection

#### 1.3.5.1 Vertical takeoff propulsion system

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

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

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

../Michael T lyx\unhbox \voidb{x \penalty \@M \ /Lyx/propulsion sy

Figure 1.3.13: 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 1.3.1: Hover motor specifications

../Michael T lyx\unhbox \voidb{x \penalty \@M \ /Lyx/propulsion system

Figure 1.3.14: Hover motor's preformance data

- ESCs
- Propellers set

### 1.3.5.2 Cruise propulsion system

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

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

Table 1.3.2: power of motors to vehicle's weight

../Michael T lyx\unhbox \voidb{x \penalty \@M \ /Lyx/propulsion system/gpr

Figure 1.3.15: 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 1.3.3: Pusher motor specifications

- ESCs
- Propellers set

### 1.3.5.3 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:

The working time of the hovering motors is 20% of the mission endurance

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

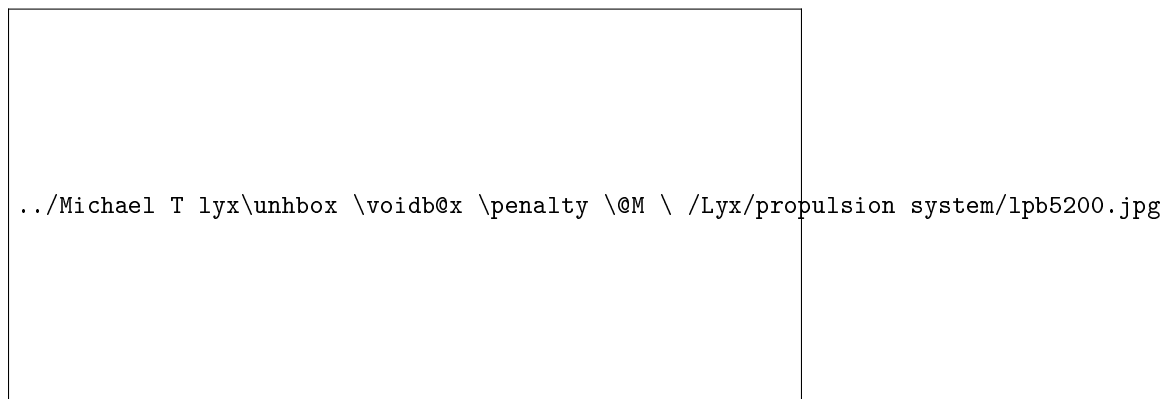


Figure 1.3.16: Lithium Polymer Battery (11.1 V, 5200 mAh- 35C)

- Battery specification

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

Table 1.3.4: Battery specification

## Chapter 2

# Mathematical Model

The mathematical model will be derived mainly to design the controller during phase in which the flying taxi acts as multicopter.

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

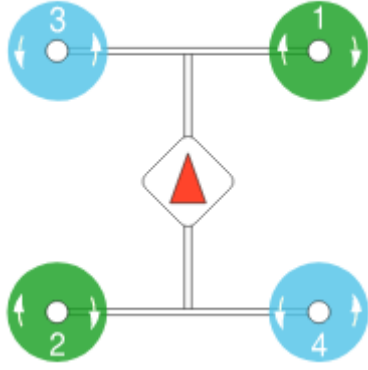
#### 2.1.1 Kinetics ( the study of forces and moments on system in motion)

- Translational motion:  $\vec{F} = \frac{d}{dt}(m\vec{v}) = m(\dot{\vec{v}} + \vec{\omega} \times \vec{v}) = \vec{F}_{Aero} + \vec{F}_{motors} + \vec{g}$ 
  - This relation represents the absolute resultant force acting on the CG ,which is the mass multiplied by the absolute change in the velocity vector of the CG of the flying taxi  $\vec{v} = u\vec{i} + v\vec{j} + w\vec{k}$  but  $\vec{i}, \vec{j}, \vec{k}$  are unit vector in body axes directions which change direction with time so the absolute change is the sum of change in magnitudes  $\dot{\vec{v}} = \dot{u}\vec{i} + \dot{v}\vec{j} + \dot{w}\vec{k}$  and directions  $\vec{\omega} \times \vec{v}$  . Using body axes directions in translational motion is very beneficial as the inertial sensors gives its measurements in body axes so it's more suitable.
  - $\vec{F}_{Aero}$  can be ignored because during multicopter phase, the flying taxi encounters negligible aerodynamic force due to low speed
  - $\vec{F}_{motors} = \sum_{i=1}^4 [-\frac{1}{2}\rho AC(\omega_i R_{propeller})^2] \vec{k} = \sum_{i=1}^4 [-b_i \omega_{prop,i}^2] \vec{k}$   
where  $b_i$  is constant can be obtained for every motor relates the square of the angular velocity of the motor  $\omega_{prop,i}^2$  to the generated thrust and the summation of the thrust of each motor holds the total thrust  $\vec{F}_{motors}$
  - $\vec{g} = C_{IB} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$   
where  $C_{BI}$  is the rotation matrix from inertia axes to body axes according to ZYX euler angles( $\psi, \theta, \phi$ ) will be derived later in the kinematics study  
 $\therefore \vec{g} = mg \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\theta) \end{bmatrix}$
- Rotational motion:  $\vec{M} = \frac{d}{dt}(I\vec{\omega}) = I\dot{\vec{\omega}} + \vec{\omega} \times I\vec{\omega} = \vec{M}_{motors}$

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

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

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



also  $T_i = b_i \omega_i^2$

It's better to design the controller using  $M_x, M_y, M_z$  instead of  $\omega_i^2$  to make the controller independent of the configuration so we need a control allocation (or mixing) law to map the controller output to a required angular velocity for each motor.

- This relation represents the absolute resultant moment around the CG , represented in the body axes directions which is very beneficial also in derivation because inertia is usually constant in body axes in contrast to fixed axes beside the previous reason stated in the translational motion.
- $I$  is the inertia tensor in body axes. For our flying taxi it's symmetric about xz plane  $\therefore I = \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix}$
- angular velocity of the flying taxi  $\vec{\omega} = p \vec{i} + q \vec{j} + r \vec{k}$
- absolute change of  $I\vec{\omega}$  is the sum of change in magnitudes  $I\dot{\vec{\omega}} = I(\dot{p} \vec{i} + \dot{q} \vec{j} + \dot{r} \vec{k})$  and directions  $\dot{\vec{\omega}} \times I\vec{\omega}$

### 2.1.2 Kinematics (the study of motion without regard to forces or moments)

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

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

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

As ZYX euler angles represents rotation from :

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

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

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

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

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

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

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

### 2.1.3 Resultant Nonlinear model

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

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

system:

$$\dot{u} = -g\sin\theta - qw + rv \quad (2.1.1)$$

$$\dot{v} = g\cos\theta\sin\phi - ru + pw \quad (2.1.2)$$

$$\dot{w} = g\cos\theta\cos\phi - pv + qu - \frac{T_{allmotors}}{m} \quad (2.1.3)$$

$$\dot{p} = \frac{I_y - I_z}{I_x}qr + \frac{M_x}{I_x} \quad (2.1.4)$$

$$\dot{q} = \frac{I_z - I_x}{I_y}pr + \frac{M_y}{I_y} \quad (2.1.5)$$

$$\dot{r} = \frac{I_y - I_x}{I_z}pq + \frac{M_z}{I_z} \quad (2.1.6)$$

$$\dot{\phi} = p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \quad (2.1.7)$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \quad (2.1.8)$$

$$\dot{\psi} = q\sin\phi\sec\theta + r\cos\phi\sec\theta \quad (2.1.9)$$

$$\dot{X} = [\cos(\theta)\cos(\psi)]u + [\sin(\phi)\cos(\psi) - \cos(\phi)\sin(\psi)]v + [\cos(\phi)\cos(\psi) + \sin(\phi)\sin(\psi)]w \quad (2.1.10)$$

$$\dot{Y} = [\cos(\theta)\sin(\psi)]u + [\sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi)]v + [\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi)]w \quad (2.1.11)$$

$$\dot{Z} = [-\sin(\theta)]u + [\sin(\phi)\cos(\theta)]v + [\cos(\phi)\cos(\theta)]w \quad (2.1.12)$$

Notes:

- from equ.s (4,5,6,7,8,9) , we have full control over all attitude angles  $(\phi, \theta, \psi)$  using  $M_x, M_y, M_z$
- from equ. (3,12) , we have direct control over Z (so altitude) using  $T_{allmotors}$
- from equ.s (1,2,10,11) , we don't have direct control over positions (X,Y) but we can using attitude, so if we want to control the position we can't independently set specific attitude as it will be determined based on the required position.

## 2.2 Linearization

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

all rates = 0 ,  $\phi = \theta = 0, p = q = r = 0$

$$\therefore \dot{u} = -g\theta \quad (2.2.1)$$

$$\dot{v} = -g\phi \quad (2.2.2)$$

$$\dot{w} = -\frac{\Delta T_{allmotors}}{m} \quad (2.2.3)$$

$$\dot{p} = \frac{M_x}{I_x} \quad (2.2.4)$$

$$\dot{q} = \frac{M_y}{I_y} \quad (2.2.5)$$

$$\dot{r} = \frac{M_z}{I_z} \quad (2.2.6)$$

$$\dot{\phi} = p \quad (2.2.7)$$

$$\dot{\theta} = q \quad (2.2.8)$$

$$\dot{\psi} = r \quad (2.2.9)$$

$$\dot{X} = u \quad (2.2.10)$$

$$\dot{Y} = v \quad (2.2.11)$$

$$\dot{Z} = w \quad (2.2.12)$$

$\dot{u}, \dot{v}, \dot{X}, \dot{Y}$  considered not so good approximations as will be proved later by comparing linear and nonlinear models but we will use them only to estimate initial gains for guidance system.

## 2.3 Parameters

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

m	1.4	Kg
I <sub>x</sub>	0.050595	kg.m <sup>2</sup>
I <sub>y</sub>	0.0817086	kg.m <sup>2</sup>
I <sub>z</sub>	0.117281	kg.m <sup>2</sup>



## Chapter 3

# Autopilot

### 3.1 Autopilot design

From previous notes we stated after deriving the nonlinear model, we designed our control block diagram to be as follows:

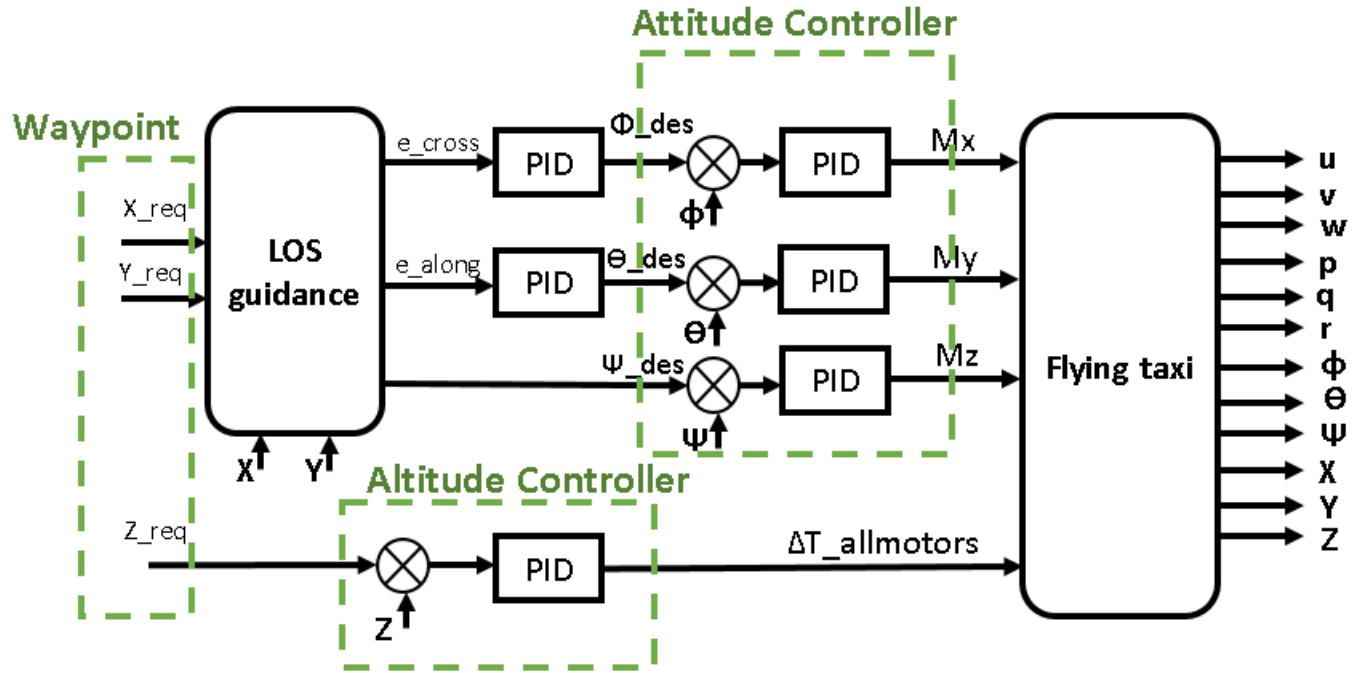


Figure 3.1.1: Autopilot design

#### 3.1.1 Attitude controller

##### Roll Controller:

From Linearization section using equations (2.2.4), (2.2.7):  $\therefore \ddot{\phi} = \frac{M_x}{I_x} \Rightarrow G_{phi} = \frac{\phi}{M_x} = \frac{1}{I_x s^2}$   
 ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

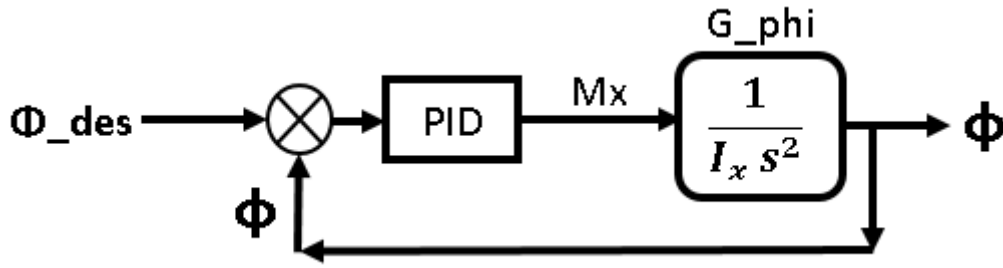


Figure 3.1.2: Roll Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.025298$  ,  $D = 0.043$  ,  $I = 0$

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

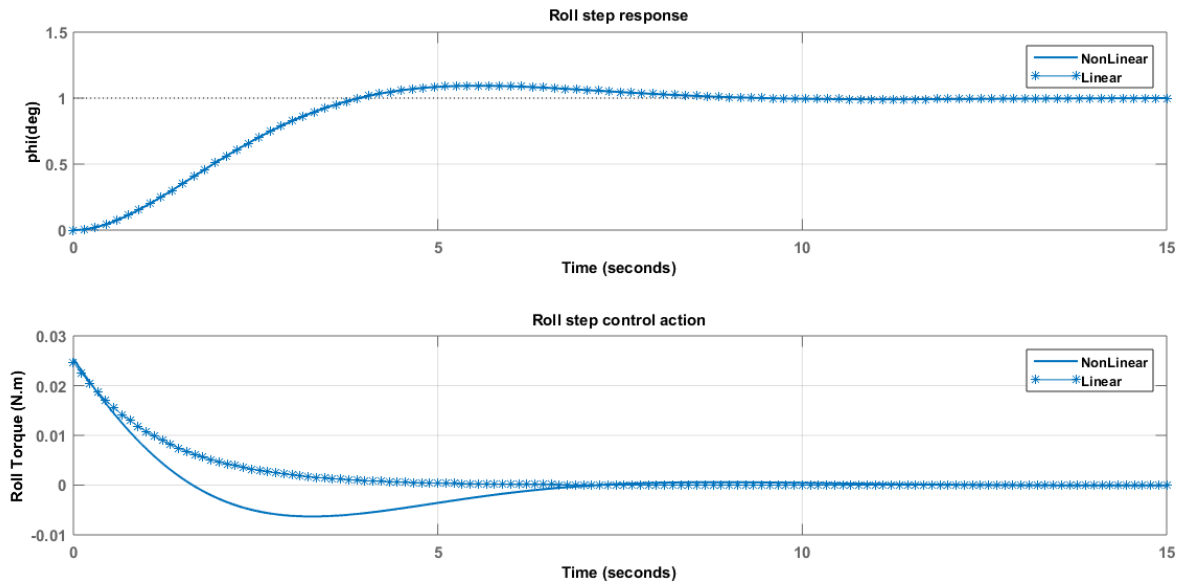


Figure 3.1.3: Roll Step response and control action

The response for 5 deg desired roll input and control action (to show the effect of larger input on the nonlinear model) :

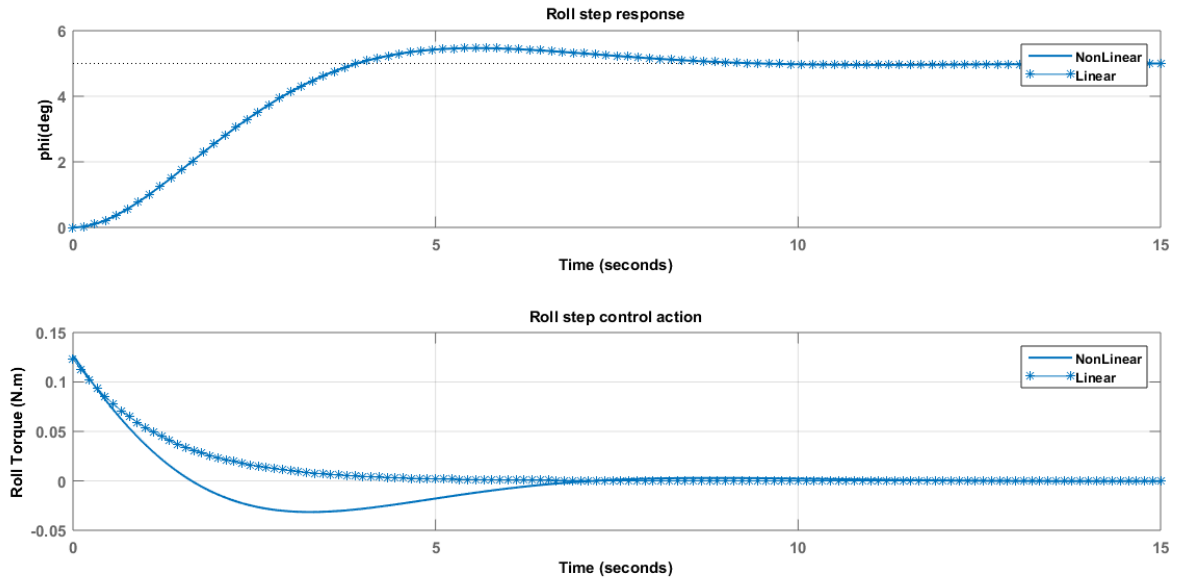


Figure 3.1.4: 5 deg desired Roll input

**Pitch Controller:**

From Linearization section using equations (2.2.5), (2.2.8):  $\therefore \ddot{\theta} = \frac{My}{I_y} \Rightarrow G_\theta = \frac{\theta}{My} = \frac{1}{I_y s^2}$   
 ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

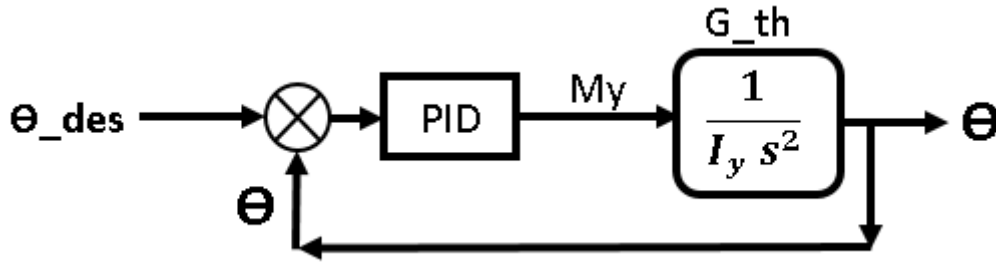


Figure 3.1.5: Pitch Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.040854$  ,  $D = 0.06945$  ,  $I = 0$

Note: Higher gains for pitch than roll due to higher  $I_y$  than  $I_x$  so the controller needs to exert more effort to move the vehicle in pitch direction

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

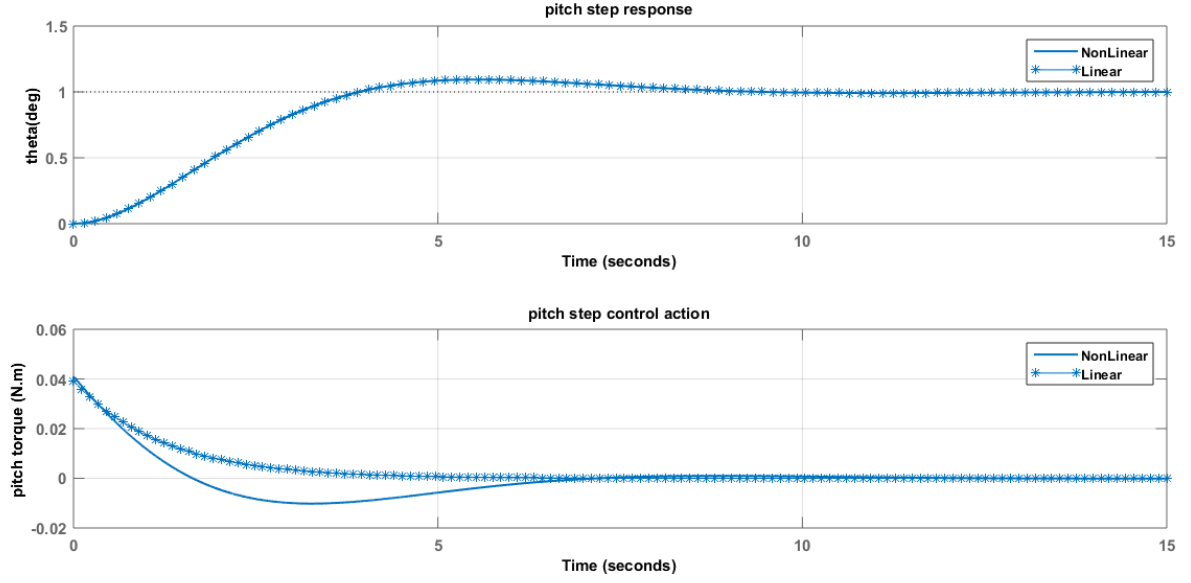


Figure 3.1.6: Pitch Step response and control action

The response for 5 deg desired pitch input and control action (to show the effect of larger input on the nonlinear model) :

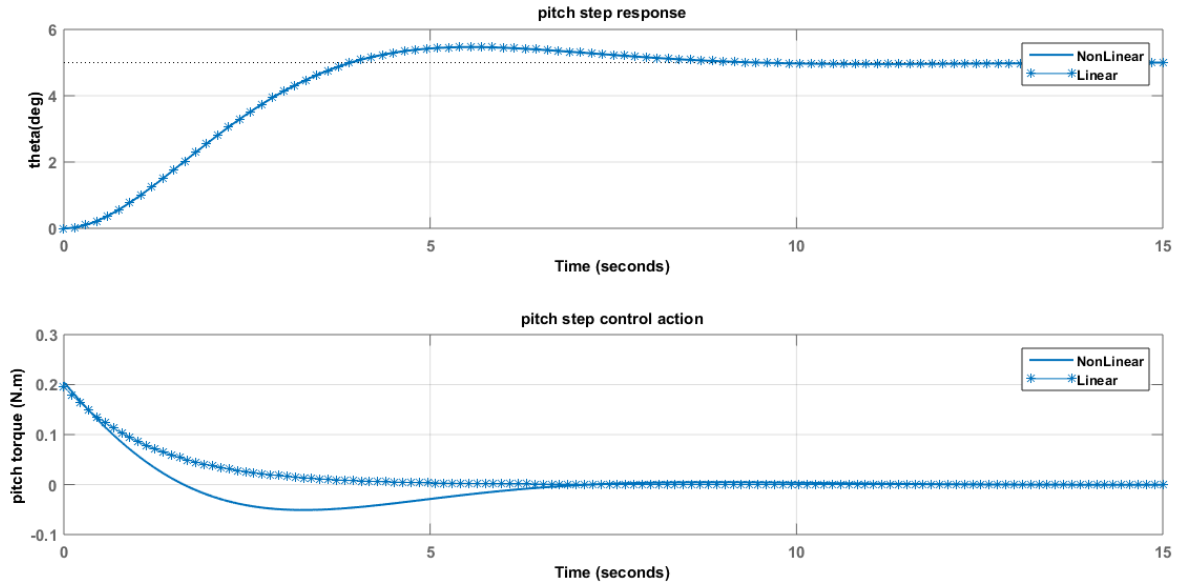


Figure 3.1.7: 5 deg desired pitch input

#### Yaw Controller:

From Linearization section using equations (2.2.6), (2.2.9):  $\therefore \ddot{\psi} = \frac{Mz}{I_z} \Rightarrow G_\psi = \frac{\psi}{Mz} = \frac{1}{I_z s^2}$   
ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

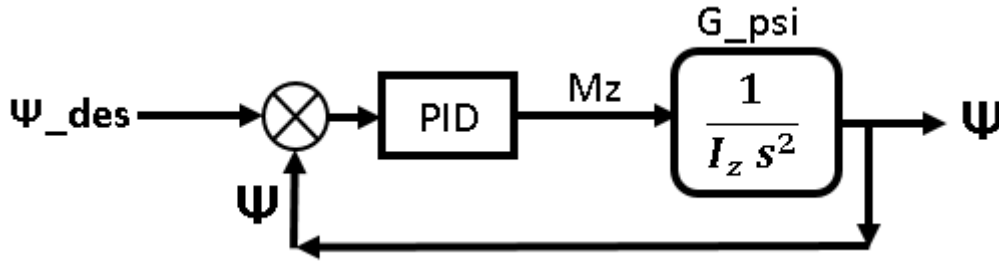


Figure 3.1.8: Yaw Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.058641$  ,  $D = 0.1$  ,  $I = 0$

Note: Higher gains for yaw than roll and pitch due to higher  $I_z$  than  $I_y$  and  $I_x$  so the controller needs to exert more effort to move the vehicle in yaw direction

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

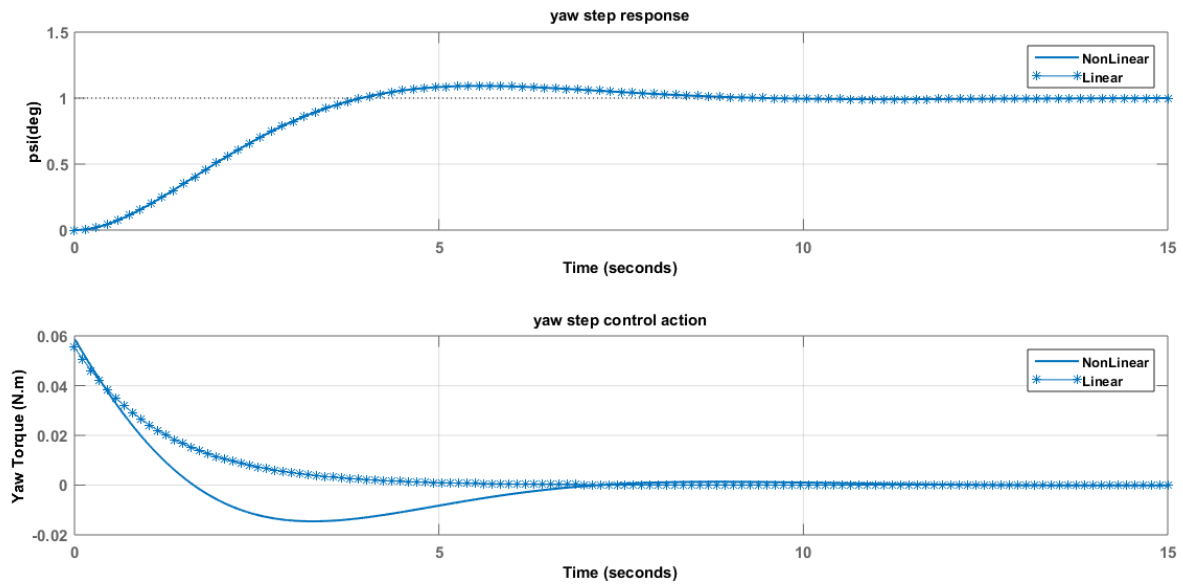


Figure 3.1.9: Yaw Step response and control action

The response for 5 deg desired Yaw input and control action (to show the effect of larger input on the nonlinear model) :

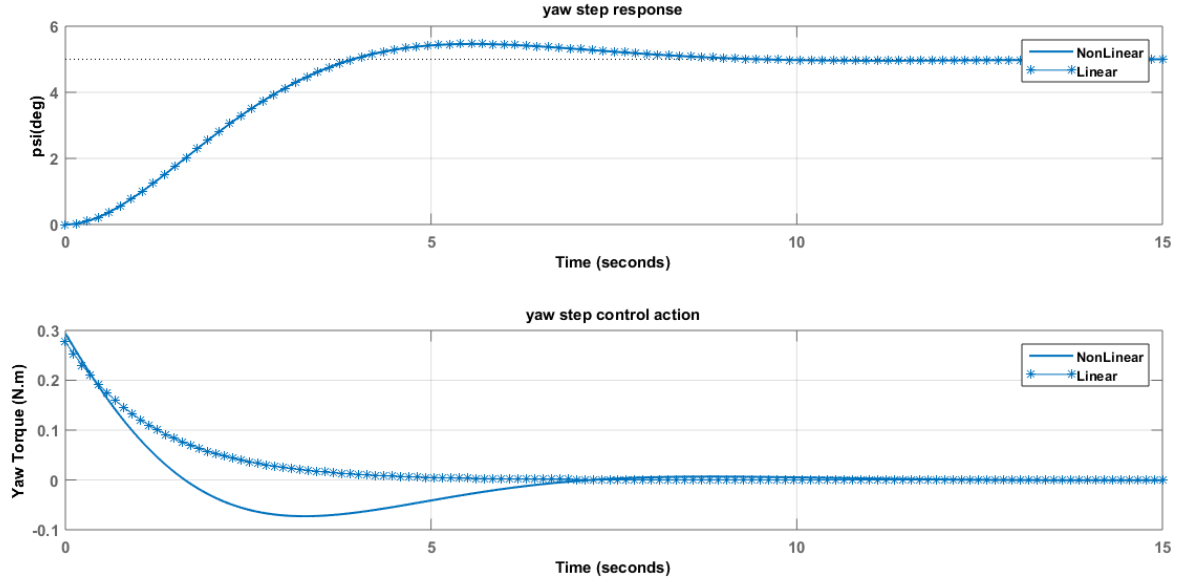


Figure 3.1.10: 5 deg desired yaw input

#### 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.

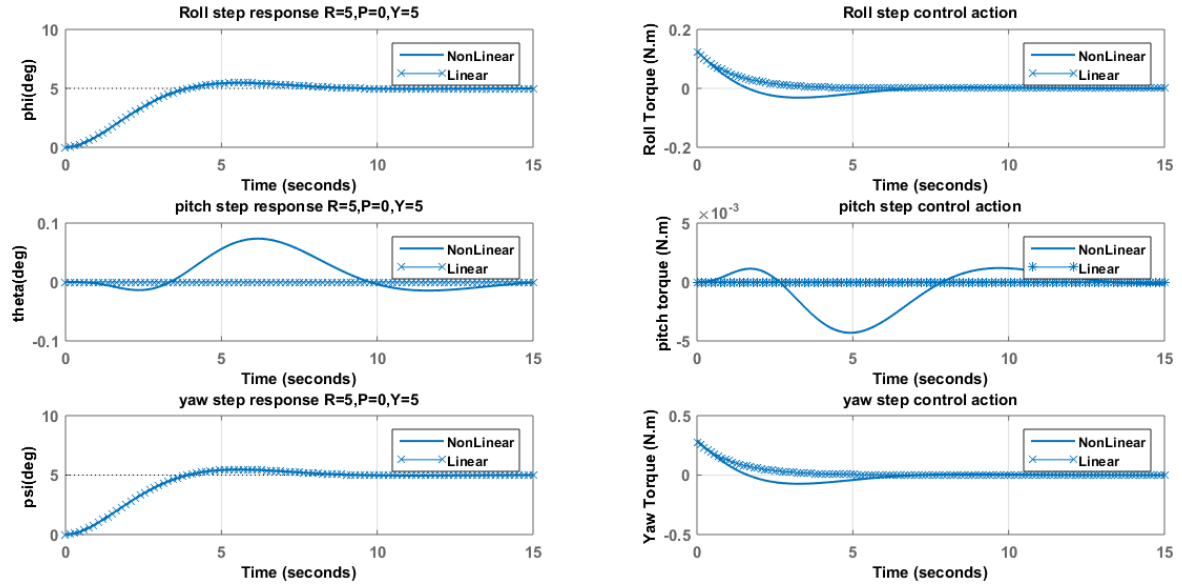


Figure 3.1.11: desired roll = 5 deg , desired pitch = 0 , desired yaw = 5

### 3.1.2 Altitude Controller

From Linearization section using equations( 2.2.3),(2.2.12):  $\therefore \ddot{Z} = -\frac{\Delta T_{allmotors}}{m} \Rightarrow G_Z = \frac{Z}{\Delta T_{allmotors}} = \frac{1}{m s^2}$  ignoring motor dynamics as it has fast response w.r.t. vehicle dynamics

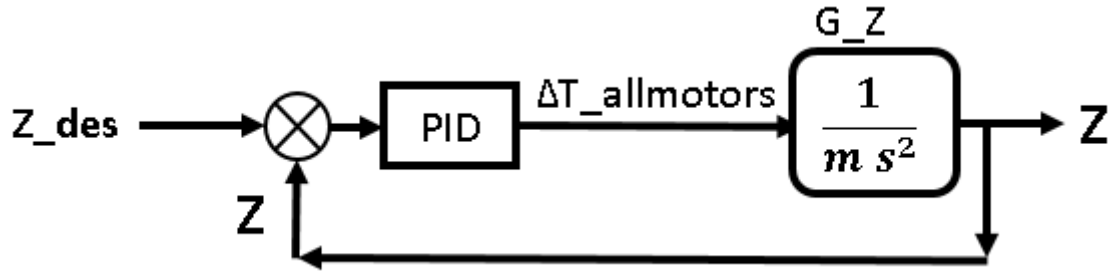


Figure 3.1.12: Altitude Controller

Using Matlab SISO tool to obtain robust controller :  $P = 0.7$  ,  $D = 1.19$  ,  $I = 0$

Note: Higher gains for altitude than attitude due to higher mass than inertia as values.

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

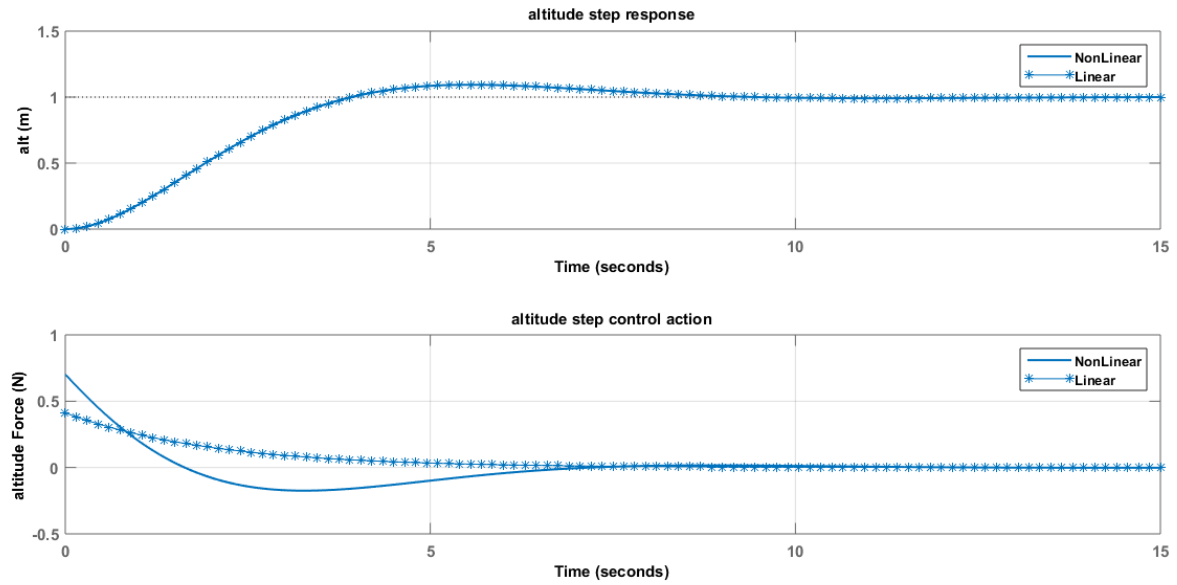


Figure 3.1.13: Altitude Step response and control action

The response for  $-4$  m ( $Z = 4$  m) desired altitude input and control action (to show the effect of larger input on the nonlinear model) :

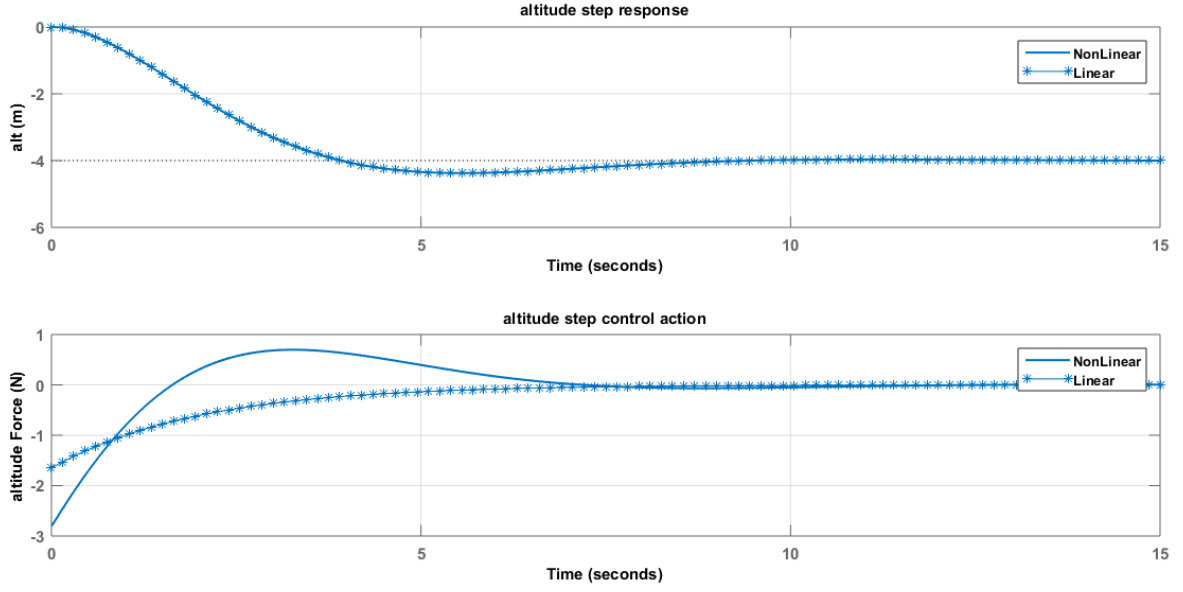


Figure 3.1.14: 5 deg desired yaw input

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

### 3.1.3 Attitude & Altitude controllers together in action

We will study the effect of changing both attitude and altitude 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

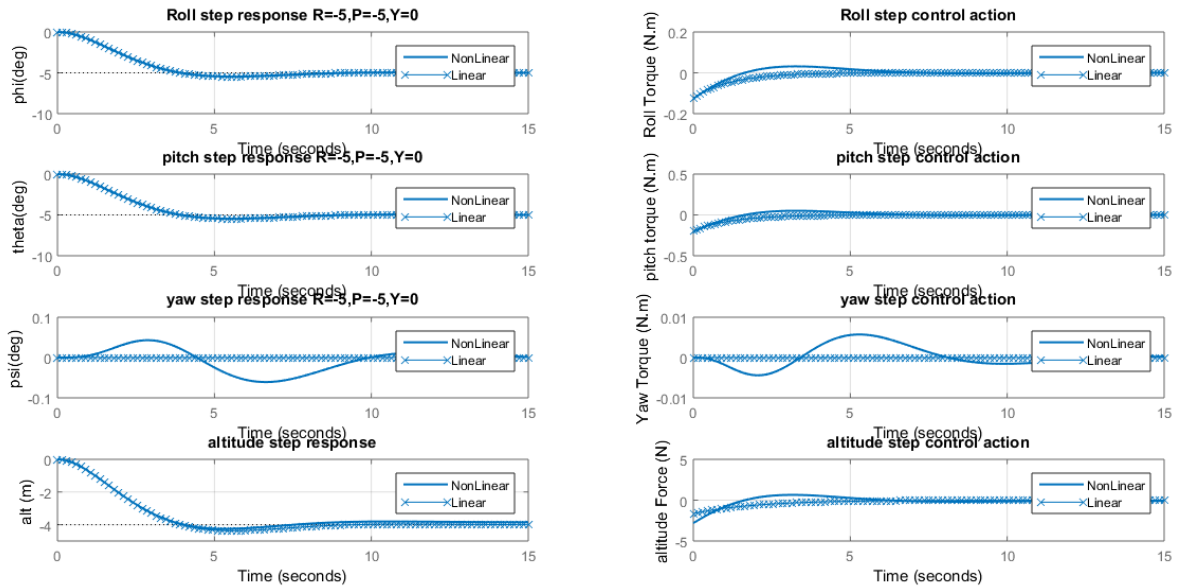


Figure 3.1.15: desired roll = -5 deg , desired pitch = -5 deg , desired altitude = -4 m ( Z = 4 m)



### 3.1.4 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.

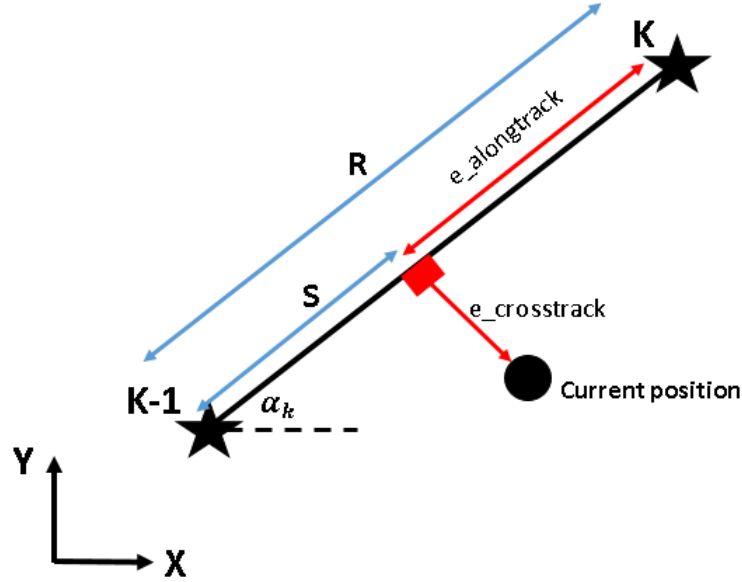


Figure 3.1.16: LOS

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

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

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

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

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

$$e_{along-track} = R - s$$

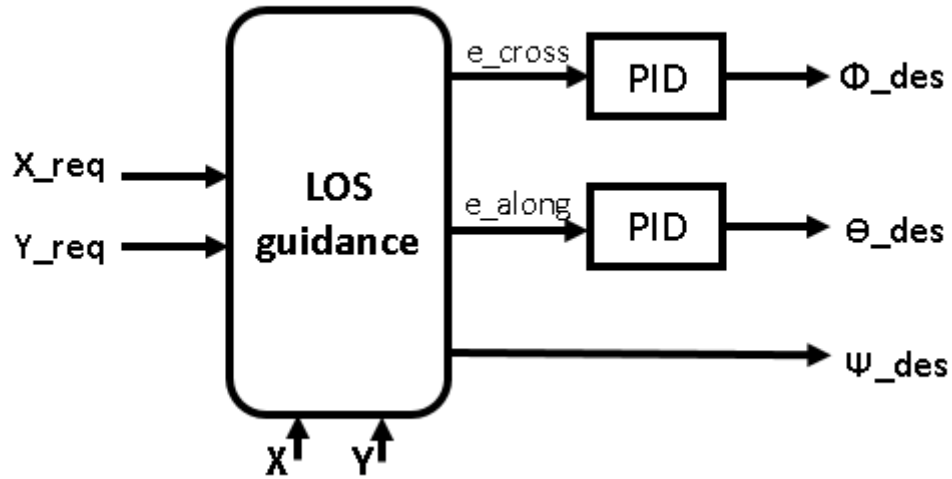


Figure 3.1.17: LOS block diagram

Our strategy to get PID values is using the 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.

From equ.s (2.2.1),(2.2.10) :  $\ddot{X} = -g\theta \Rightarrow G_X = \frac{X}{\theta} = \frac{-g}{s^2}$

It's clear that the linearized model only accounts for g component effect but the  $\Delta T_{all motors}$  component doesn't appear

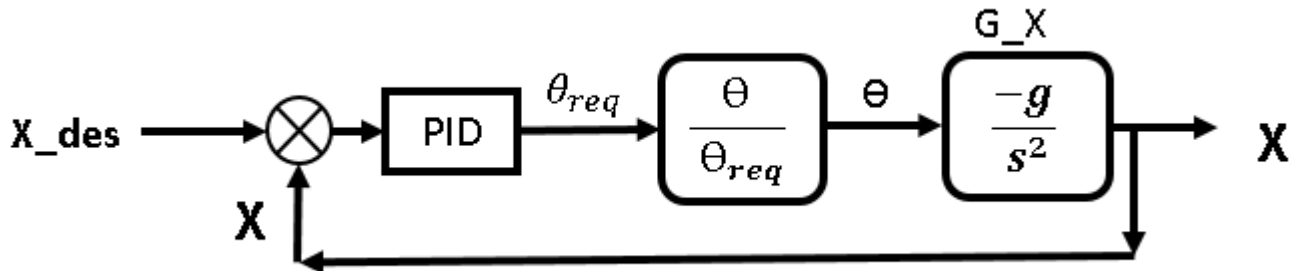


Figure 3.1.18: X guidance

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

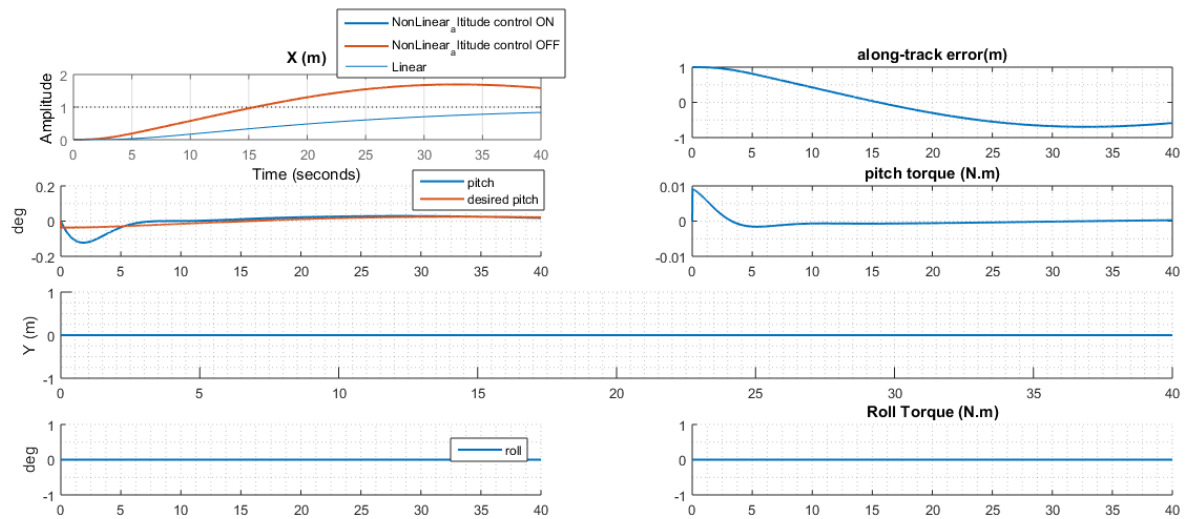


Figure 3.1.19: along-track control (trial : 1)

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

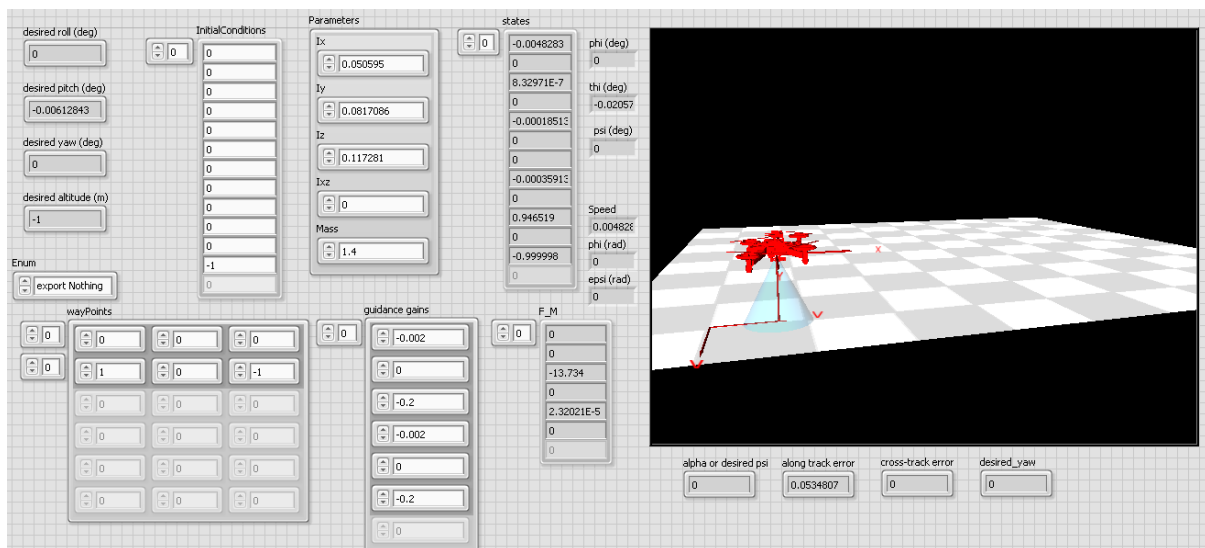


Figure 3.1.20: LabVIEW front panel

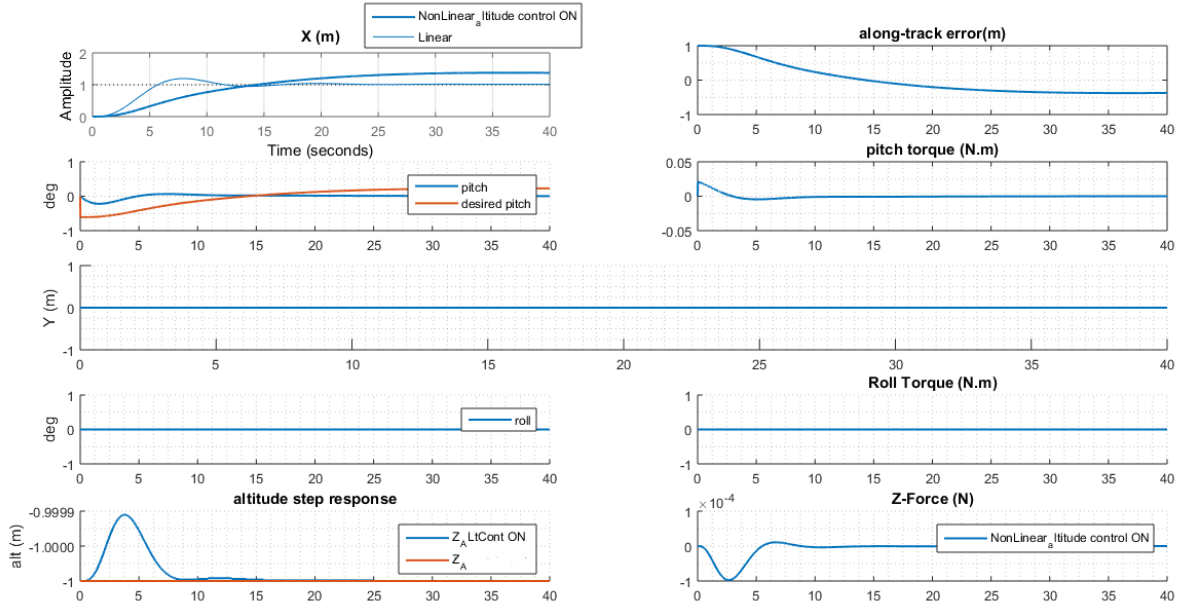


Figure 3.1.21: along-track control (trial : 2 increasing P) : high steady-state error

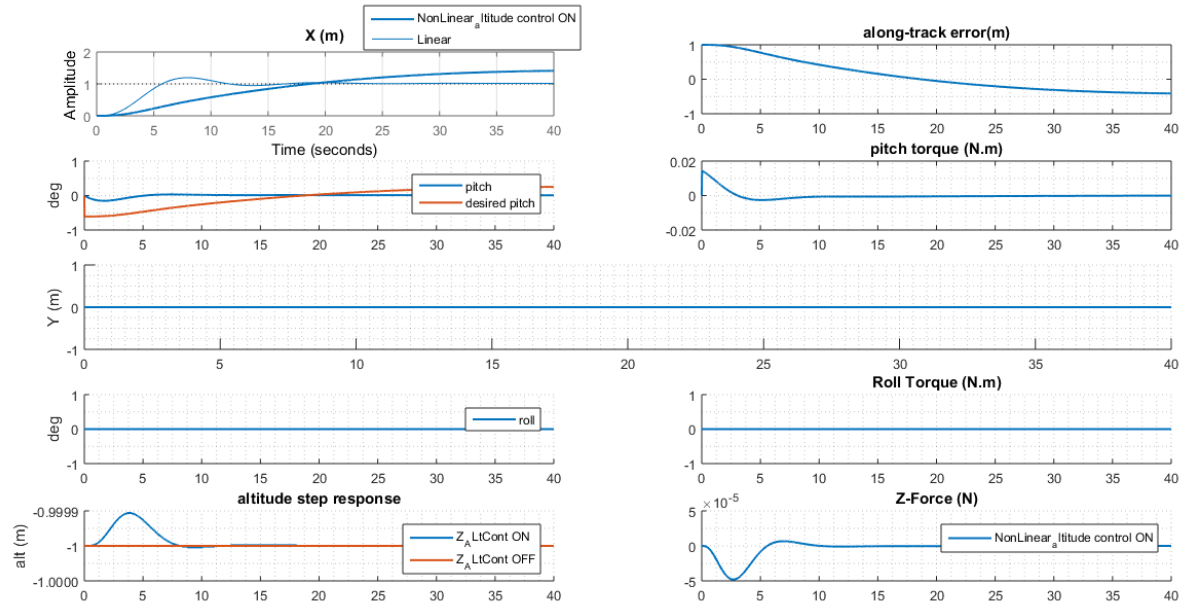


Figure 3.1.22: along-track control (trial : 2 decreasing D) : high steady-state error and the settling time is increased (as expected but we only wanted see the effect)

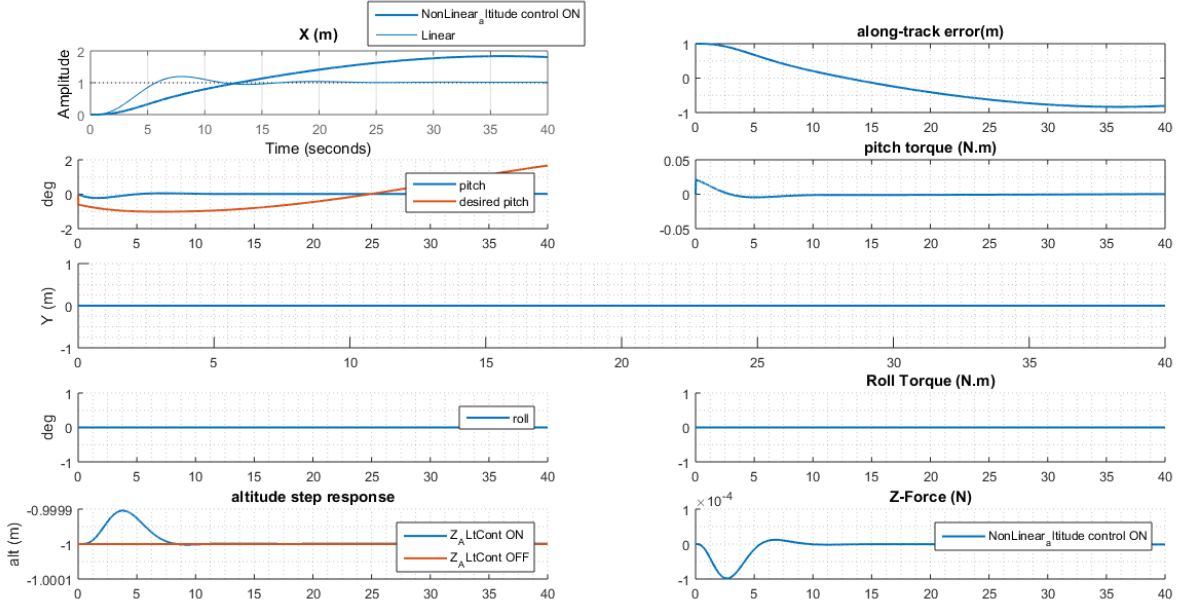


Figure 3.1.23: along-track control (trial : 3 adding small I term to eliminate steady-state error) : settling time is highly increased

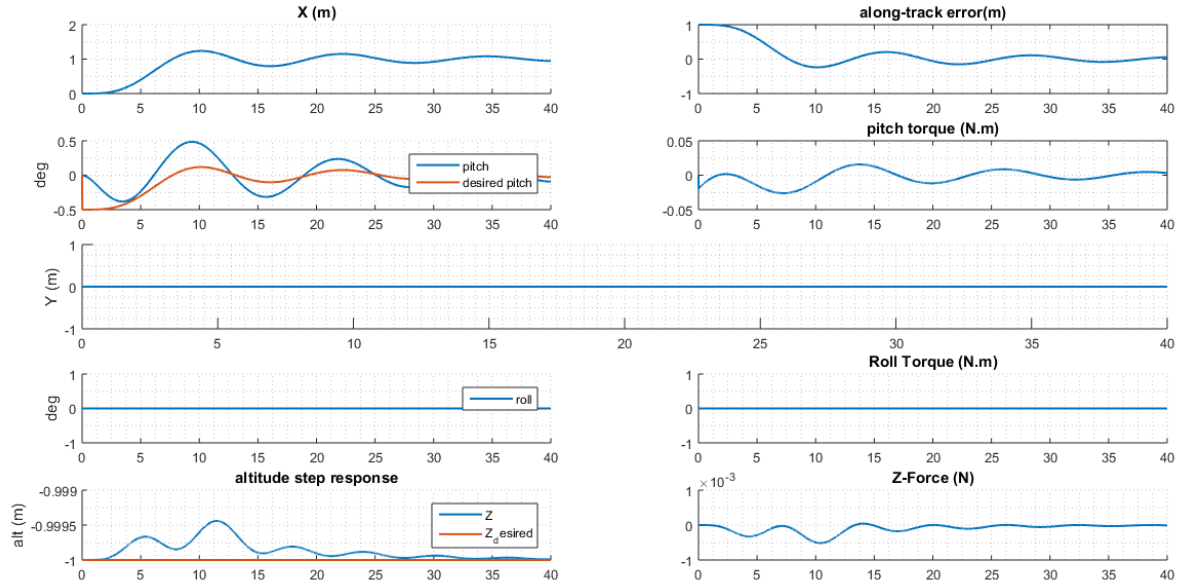


Figure 3.1.24: along-track error control (final trial) : increasing P and D

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

new gains after tuning using nonlinear simulation :  $P = -0.002$  ,  $D = -0.2$  ,  $I = 0$

in all previous results the altitude control is working as could be seen from Z-force response.

Similarly, For cross-track error we used the same procedure:  $\ddot{Y} = -g\phi \Rightarrow G_Y = \frac{Y}{\phi} = \frac{-g}{s^2}$

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

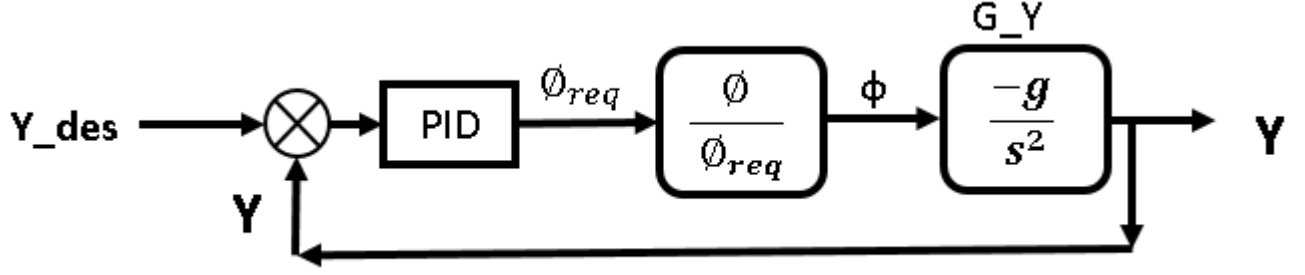


Figure 3.1.25: Y guidance

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

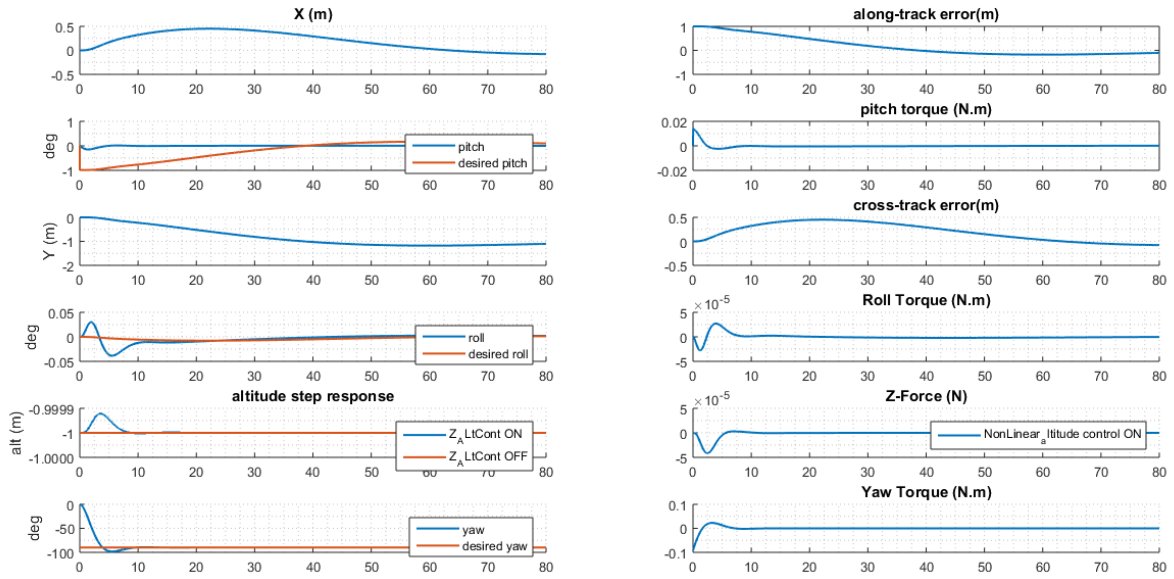
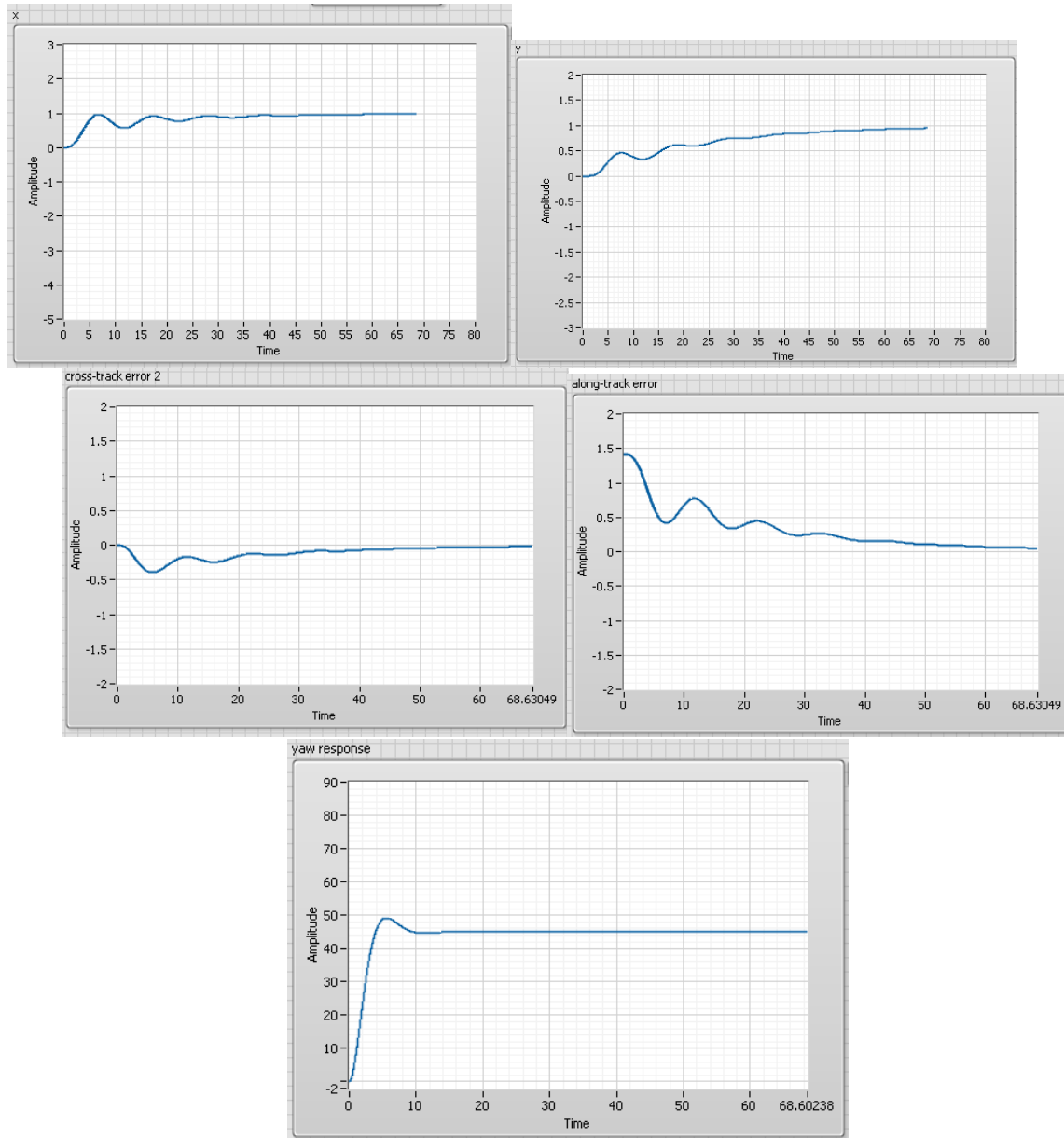


Figure 3.1.26: X,Y guidance waypoint(X=0 , Y=1 , Z=-1)

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

Figure 3.1.27: X,Y Guidance waypoint ( $X=1$  ,  $Y=1$  ,  $Z=-1$ )

## 3.1.5 Simulation using LabVIEW

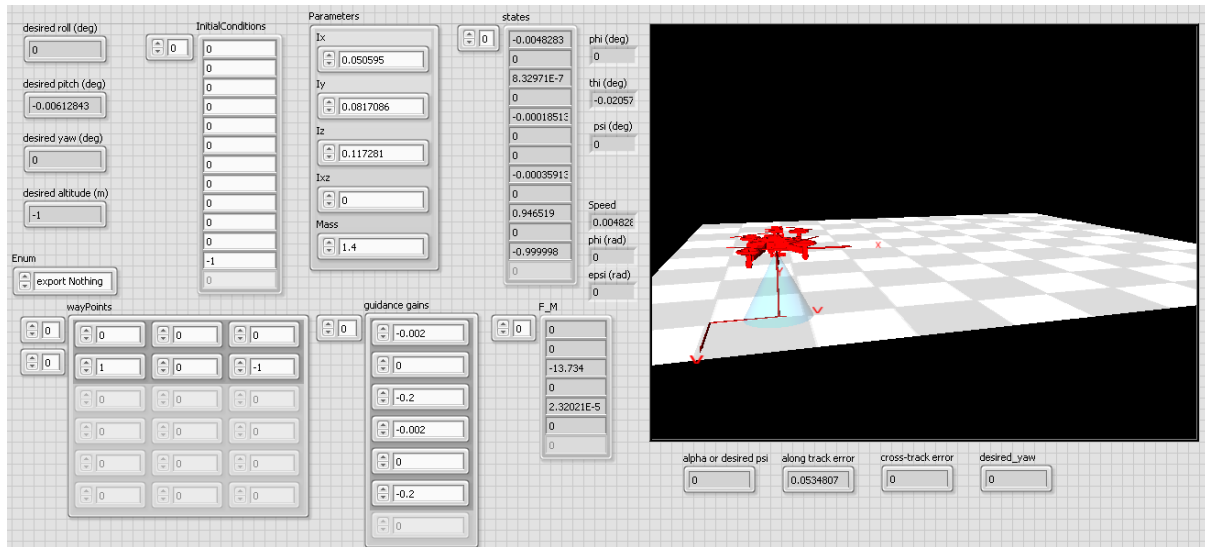


Figure 3.1.28: LabVIEW front panel

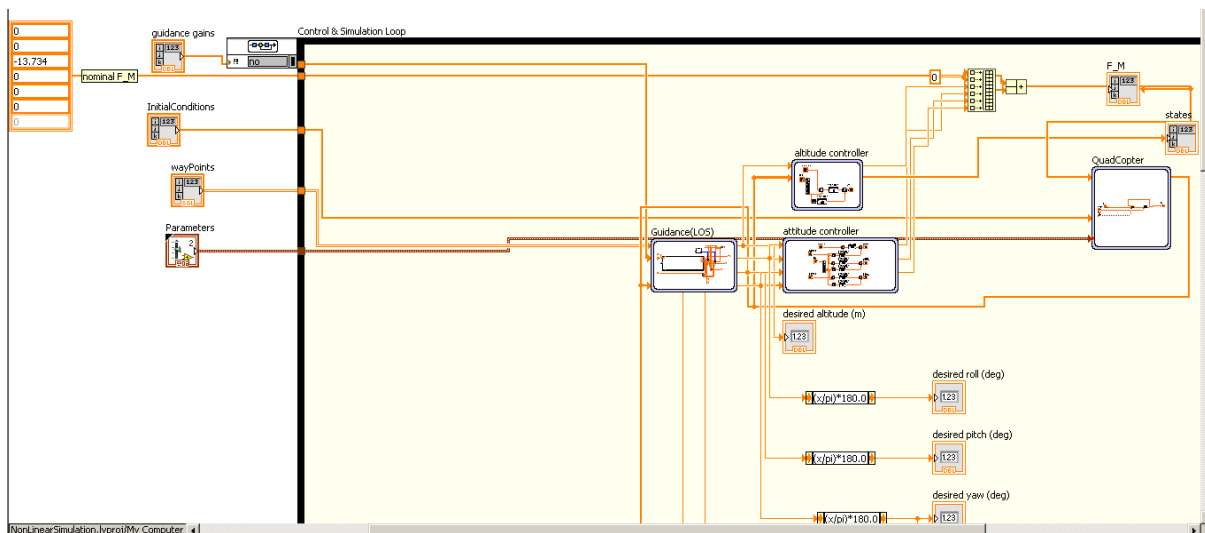


Figure 3.1.29: LabVIEW blockdiagram



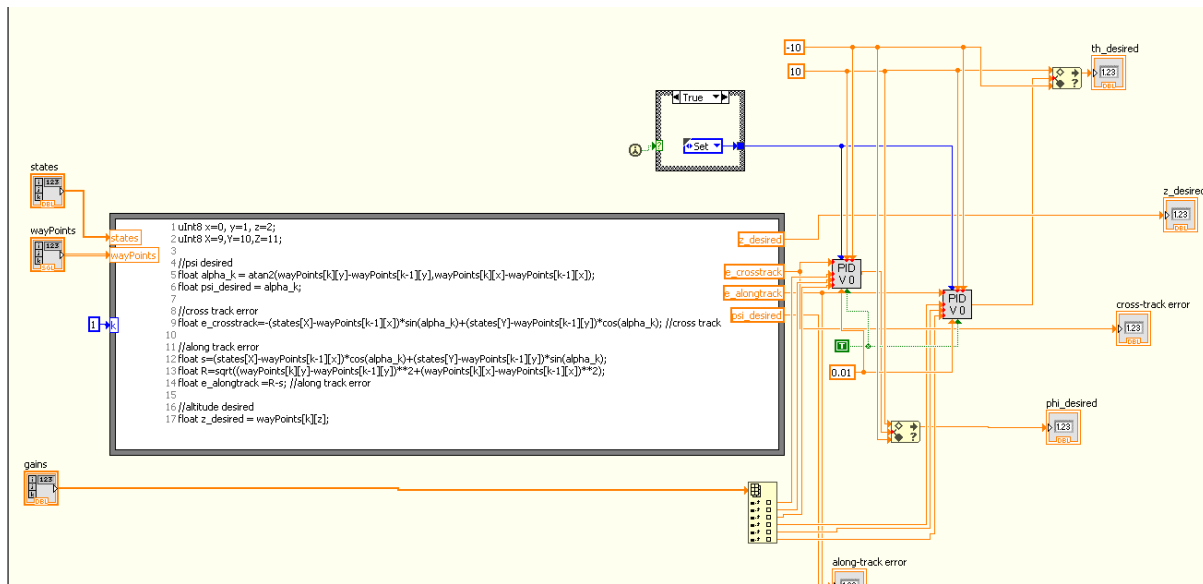


Figure 3.1.30: LabVIEW Guidance blockdiagram

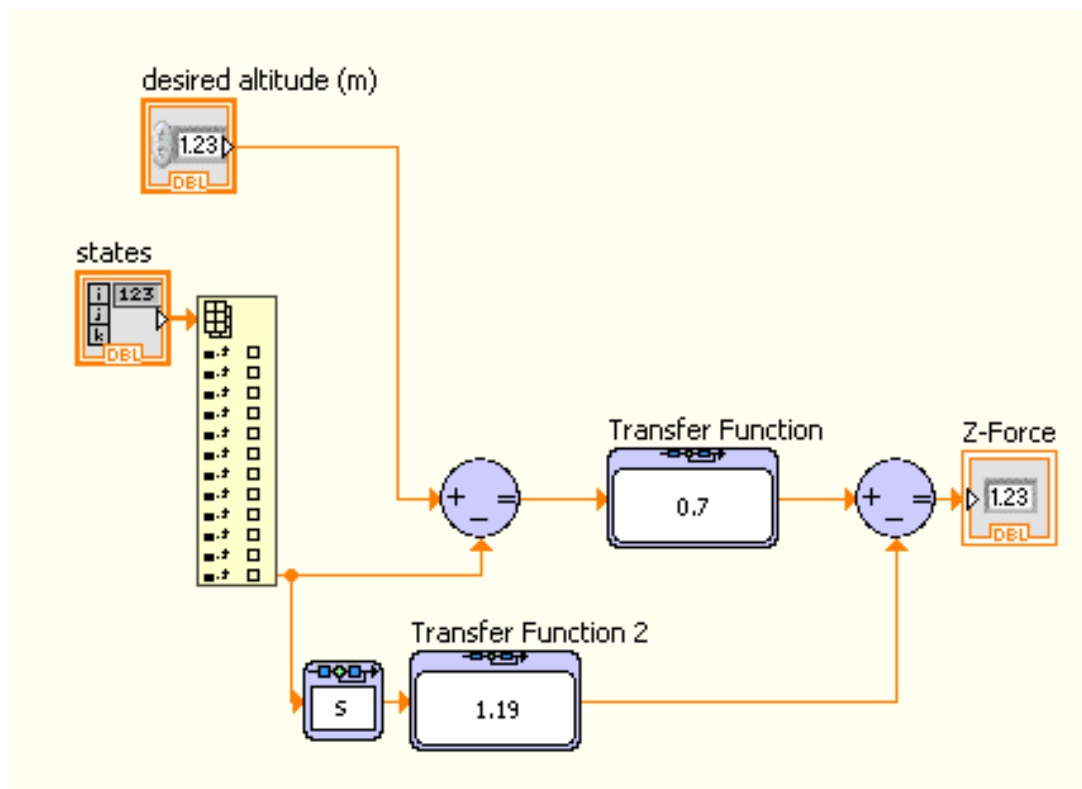


Figure 3.1.31: LabVIEW altitude BD

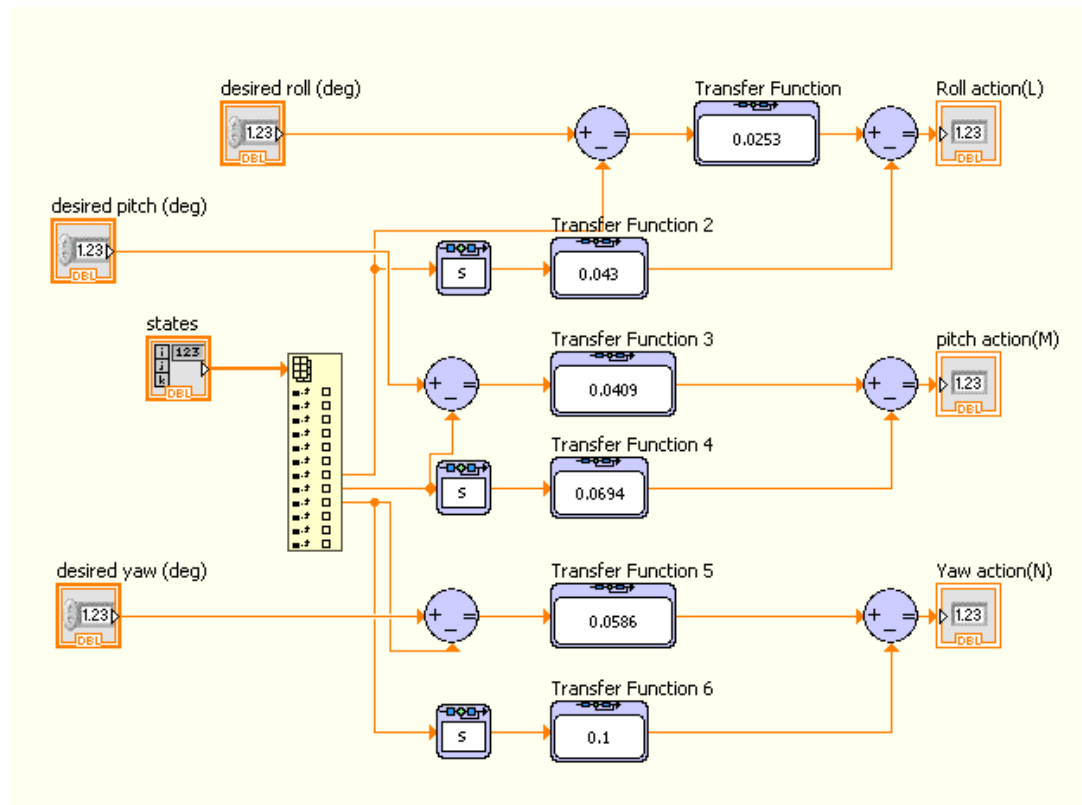


Figure 3.1.32: LabVIEW attitude controller BD

## 3.2 Commercial Autopilot

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

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

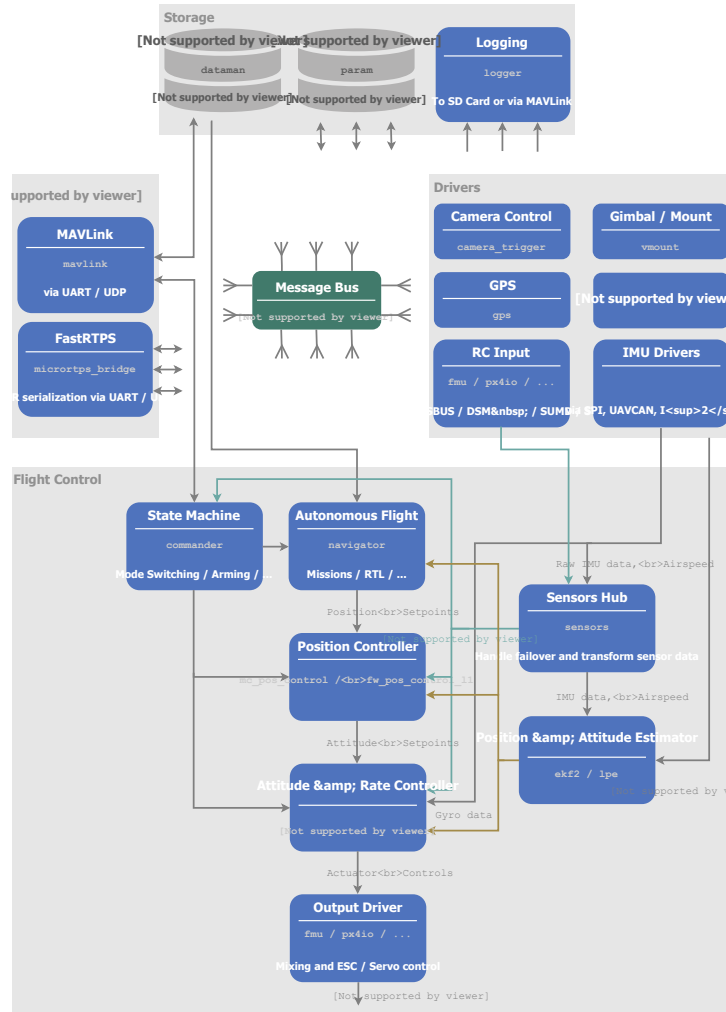


Figure 3.2.1: PX4 architecture from PX4 website

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

### 3.2.1 Commander

The commander is a state-machine architecture which can switch the flight modes based on specific conditions from the remote control or even from the mavlink (using companion computer). A simple example shows how the commander works: The commander has a main loop inside which check if there is an “Arming” signal coming from the remote control and if there is , it will call a function called “arming\_state\_transition()” and after checking if it’s valid to “Arm” the 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.

### 3.2.2 Navigator

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

### 3.2.3 Position controller

inputs: Position setpoints & heading

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

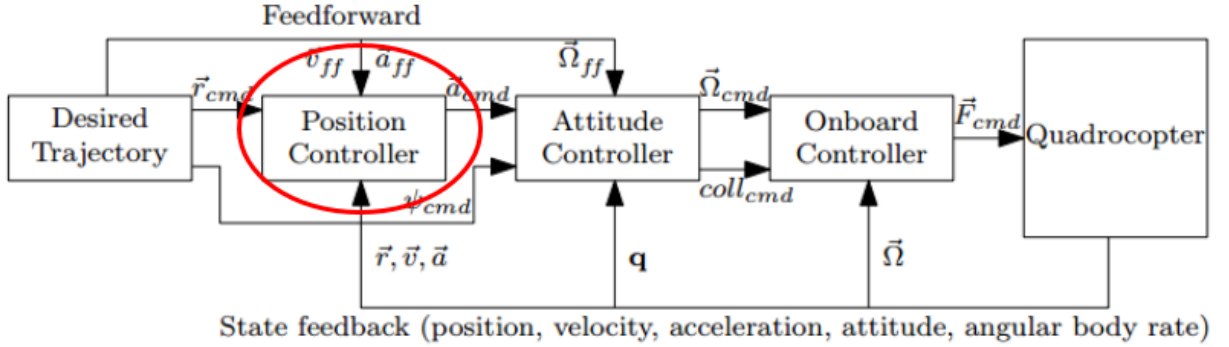


Figure 3.2.2: cascaded control architecture from “Nonlinear Quadcopter Attitude Control Technical Report, Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello, 2013”

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.

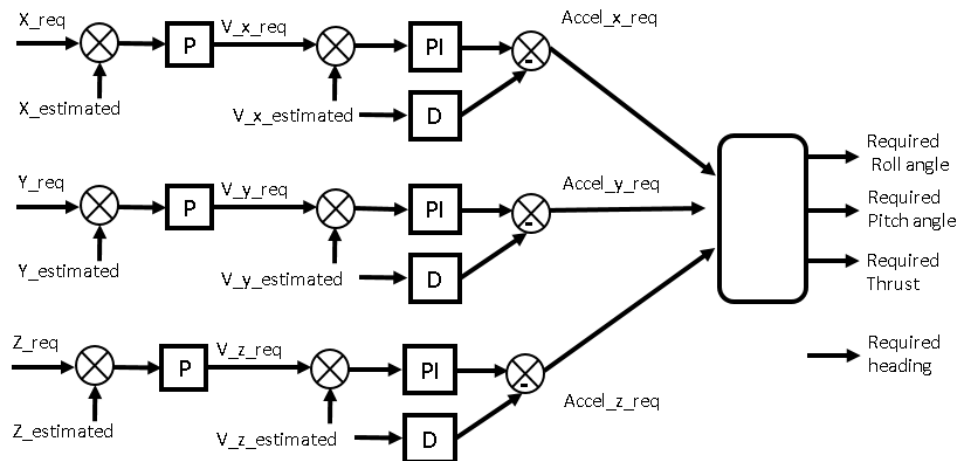


Figure 3.2.3: Position Controller

### 3.2.4 Attitude & Rate Controller

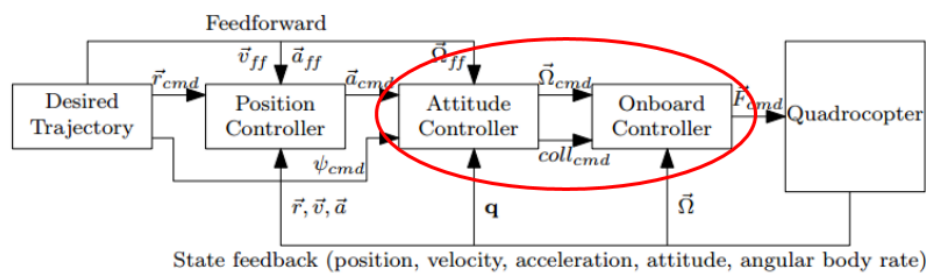


Figure 3.2.4: cascaded control architecture from “Nonlinear Quadrocopter Attitude Control Technical Report, Brescianini, Dario; Hehn, Markus; D’Andrea, Raffaello, 2013”

inputs: attitude setpoints(Roll-Pitch-Yaw)

outputs: **Required Forces and Moments**

The Attitude controller consists also of 2 loops :

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

```
// mix Full and reduced desired attitude
Quatf q_mlx = qd_red.inversed() * qd;
q_mlx *= math::sgnZero(q_mlx(0));

// catch numerical problems with the domain of acosf and asinf
q_mlx(0) = math::constrain(q_mlx(0), -1.f, 1.f);
q_mlx(3) = math::constrain(q_mlx(3), -1.f, 1.f);
qd = qd_red * Quatf(cosf(_yaw_u * law, 0, 0, sinf(_yaw_u * asinf(q_mlx(3)))));

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

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

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

### Control Law of Outer Loop of attitude controller

$$\vec{\Omega}_{cmd}(\mathbf{q}) = \frac{2}{\tau} \text{sgn}(q_{e,0}) \mathbf{q}_{e,1:3}, \quad \text{sgn}(q_{e,0}) = \begin{cases} 1, & q_{e,0} \geq 0 \\ -1, & q_{e,0} < 0 \end{cases}$$

$$\mathbf{q}_e := \mathbf{q}^{-1} \cdot \mathbf{q}_{\text{cmd}}$$
 error measure, representing the rotation from  $\mathbf{q}$  to  $\mathbf{q}_{\text{cmd}}$ 

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));

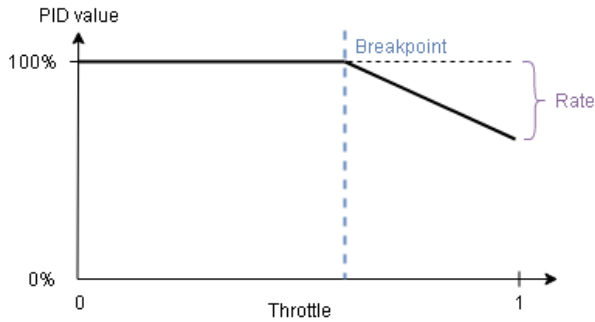
_att_control = rates_p_scaled.emult(rates_err) +
    _rates_int -
    rates_d_scaled.emult((rates_filtered - _rates_prev_filtered) / dt +
        _rate_ff.emult(_rates_sp));

_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.



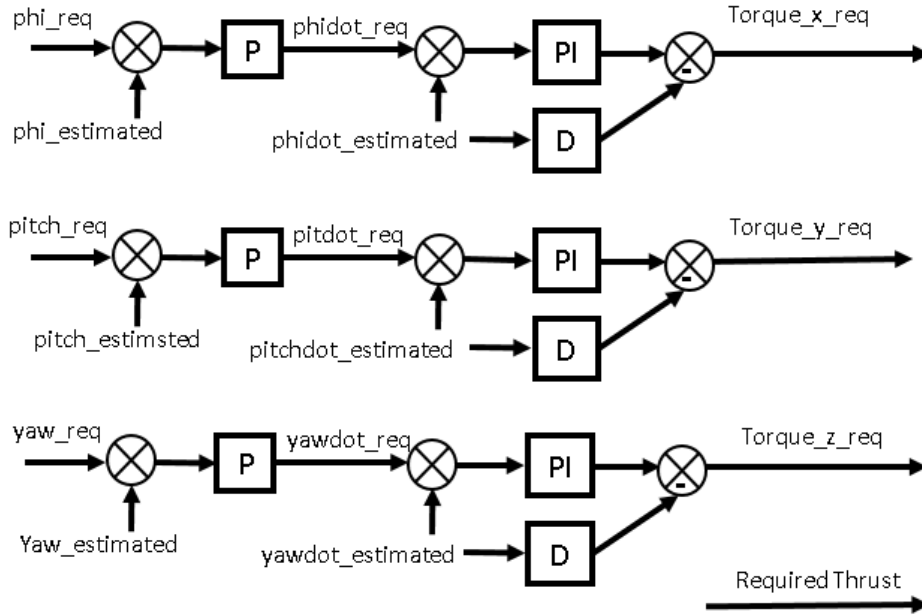


Figure 3.2.5: Attitude&amp;rate controller

### 3.2.5 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 <b>complementary filter</b> for <b>attitude</b> .	It's a <b>complementary filter</b> for estimating <b>3D position and velocities</b> .	It's an <b>EKF(Extended Kalman filter)</b> for estimating <b>3D position and velocities</b> .	It's as an <b>EKF(Extended Kalman filter)</b> for estimating <b>3D position , velocities , wind states and attitude</b> .

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** .

### 3.2.6 Output drivers

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

### 3.3 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 inertial data can aid the visual odometry. So GPS, IMU and Visual odometry can be used together for better state estimation. In our case, the ZED Stereo camera is used to get Point Cloud data<sup>1</sup> from which we can get visual odometry as well as information about obstacles. Obstacles information is used by the obstacle avoidance algorithm which evaluates the path planning and generates the reference trajectory. The visual odometry is fused with the PX4 position and/or attitude estimations to get a better state estimates.

### 4.1 Stereo Vision

---

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

Stereo Camera	Depth Cameras	Laser Scanner
<ul style="list-style-type: none"> <li>• 2 cameras are rigidly mounted to a common mechanical structure.</li> <li>• Its performance depends on : mechanical design , resolution , lens type &amp; quality ,....</li> <li>• It can only estimate the distances to features like: sharp , high contrast corners (because they have high gradient in x,y directions). so It can't get distance to featureless wall but practically outdoor scenes have sufficient textures for stereo vision to work well.</li> <li>• ROS message: sensor_msgs/Image and sensor_msgs/CameraInfo</li> <li>• Image processing packages can be used to handle the outputs of the cameras to achieve the stereo vision</li> </ul> <div>../pasted16.png</div> <div>../pasted17.png</div> <p>ZED stereo Camera : from 0.5 to 20m at 100FPS, indoors and outdoors.</p>	<ul style="list-style-type: none"> <li>• Unlike the passive stereo camera, they are active cameras that shine a texture pattern on the surfaces and from the deformation of the pattern they can get the informations of the scene.</li> <li>• They greatly improves the system performance.</li> <li>• typically operate near-infrared wavelength to reduce system sensitivity to the colors of the objects.</li> <li>• There are 3 types of depth cameras: Kinect(structured light) - unstructured light depth cameras which employ a random texture - time-of-flight depth cameras which use an IR or laser pulses and special pixel structures in image sensors to estimate the depth from the time of laser bounding back and the speed of light.</li> <li>• the output of those cameras is point clouds which are estimated 3D points of the scene</li> <li>• ROS message: sensor_msgs/PointCloud2</li> </ul> <div>../pasted18.png</div>	<ul style="list-style-type: none"> <li>• superior accuracy (about 14 mm error)</li> <li>• longer sensing range than cameras (0.2 - 6 m)</li> </ul> <div>../pasted19.png</div> <div>../pasted20.png</div> <ul style="list-style-type: none"> <li>• ROS message: sensor_msgs/LaserScan</li> </ul>

Table 4.1.1: different sensors for vision

- **discussion**

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

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

- ZED stereo camera vs Kinect:

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

Table 4.1.2: ZED stereo camera vs Kinect

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

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

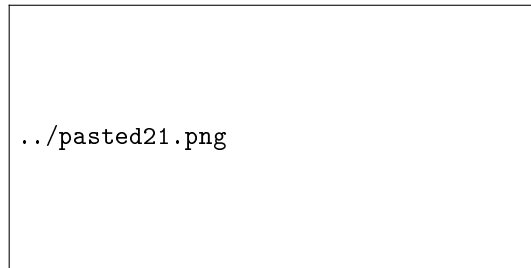


Figure 4.1.1: Ref[?] MAV uses 4 cameras to get pose estimate real-time on-board

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

- 2 forward cameras used with reduced(modified) SLAM based on PTAM<sup>5</sup> and it estimates the pose of the MAV
- 2 downward cameras used with ground-plane detection algorithm(to estimate the height , pitch and roll) and frame-to-frame tracking algorithm(to estimate the yaw and horizontal displacement)
- finally, they fused the data from the 2 stereo configurations and outputted an estimate for the MAV pose used to control the position and attitude of the MAV instead of IMU only unlike PIXHAWK
- They used stereo matching algorithm twice for both the forward stereo and downward stereo on-board
- They showed that adding 2 downward cameras significantly improves the self-localization
- 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 synchronised to capture data at the same time.
  - On-board computer with Intel Core 2 Duo CPU with 1.86GHz to execute image processing and motion estimation
  - Microcontroller with IMU to send IMU data to the on-board computer and receive from it the high-level control instructions using I2c bus.
- The 2 forward facing cameras
  - apply feature detection (800 feature upper bound) then do the sparse stereo matching
  - SLAM employed to get pose from the successfully matched features

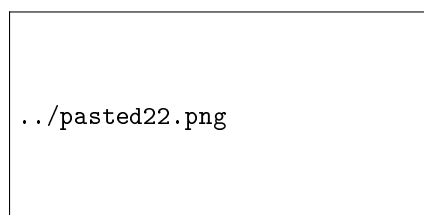


Figure 4.1.2: 2 forward-facing stereo configuration

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

- 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

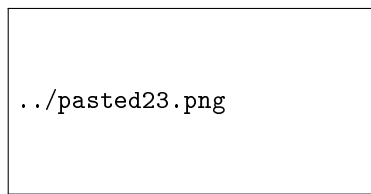


Figure 4.1.3: final configuration

- Final Results



## 4.2 Autonomous robots architecture

Important terminologies in autonomous robots:

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

Autonomous robot architecture can be abstracted into 3 main layers:

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

## 4.3 Obstacle Avoidance

obstacle avoidance methods can be concluded as follows:

Global Obstacle avoidance			Local Obstacle avoidance
Grassfire algorithm	Dijkstra's algorithm	A* search	Bug0 - Bug1 - Bug2
			Bug3 - Bug4 - Bug5

### 4.3.1 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.

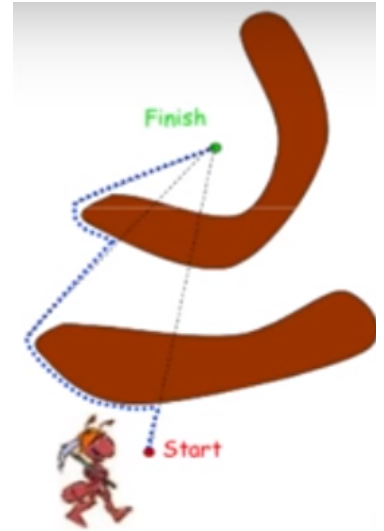
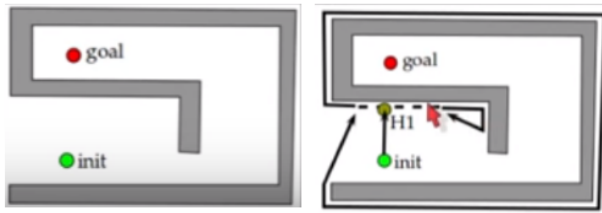


- **Bug Algorithms**

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**



# Chapter 5

## Implementation

### 5.1 ROS

#### 5.1.1 What is ROS

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

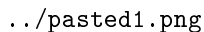


Figure 5.1.1: ROS architecture

##### 5.1.1.1 History of ROS

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

##### 5.1.1.2 ROS Philosophy

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

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

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

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

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

---

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

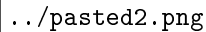
<sup>2</sup>Application programming interface : it is a set of methods of communication among various components

**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.

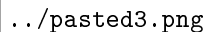


..../pasted2.png

Figure 5.1.2: Why ROS?

#### **NOTE**

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



..../pasted3.png

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

---

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

## 5.1.2 Pros and Cons of ROS

### 5.1.2.1 Advantages:

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

### 5.1.2.2 Disadvantages:

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

## 5.1.3 Example

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

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

---

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

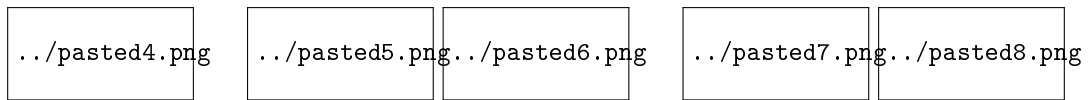


Figure 5.1.4: ROS example

### 5.1.4 Summary of some important features in ROS

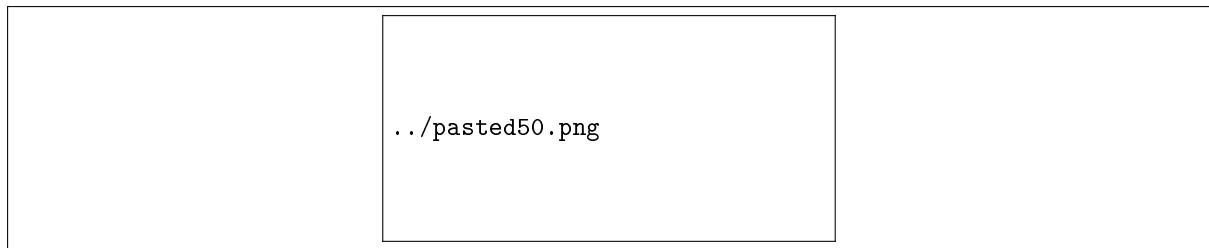
- Catkin

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 rosbuilt 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:



- Source Space (Folder)

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

- Build Space (Folder)

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

- Development (Devel) Space (Folder)

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

- Install Space (Folder)

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

- default workspace is loaded with:

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

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

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

overlay your catkin workspace with :

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

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

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

check your catkin workspace with by:

```
>echo $ROS_PACKAGE_PATH
```

Or by writing

```
>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-purpose executable program
- individually compiled, executed and manged
- organized in packages
- Run a node:

```
>roslaunch package_name node_name
```

See active nodes:

```
>roslaunch list
```

Retrieve info about a node:

```
>roslaunch 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 avoid race conditions

- ```
>rostopic list
>rostopic echo /topic
>rostopic info /topic
```

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

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

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

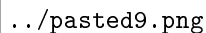
- remapping is a very useful feature ROS can do on topics. For example if I have a node called “image\_view” reads from a topic called “image” which published in by the “camera” node but we have two cameras (left and right) so we can remap the topic to be exist in a two separate namespaces each hold 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.
- composed of a nested structure of integers, floats, booleans, strings, arrays of objects, ...
- We can define one in \*.msg files but It is preferred to use ready-made ROS messages to keep compatible with the community besides there is a variety of messages that we may never need to define new one.
- ```
>rostopic type /topic_name
>rostopic pub /topic_name msg_type data
```



../pasted9.png

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 practice to use rosrund on every node.
  - We can define launch file as \*.launch in launch folder in our package.
  - If the roscore isn't running, launch automatically starts a roscore
  - ```
> roslaunch package_name launchfile.launch
```
- 

- **ROS Packages**

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

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

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

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

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

- for creating a package go to ~/catkin\_ws/src and use this instruction in the terminal:

```
>catkin_create_pkg package_name <dependencies> {for example: package_name_msgs}
```

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

- **ROS services**

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

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

“{ }” is an Empty msg we can use if we want to only execute 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
```

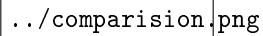


Figure 5.1.7: comparison between ROS parameters, Dynamic Reconfigure , Topics , Services , and Actions

## 5.2 Proof of Concept

### 5.2.1 State Estimation (Rover)

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

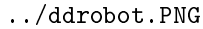


Figure 5.2.1: Differential Drive Robot (DD robot)

- **Objective**

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

- **Modeling**



Figure 5.2.2: differential drive model


- **Block Diagram**



Figure 5.2.3: differential\_drive package

- **Hardware**

MPU 6050 & Encoder

## 5.2.1.1 Software

Control the DD robot linear velocity and angular velocity

- nodes

| package            | node name       | description                                                                                                                                                             | parameters                                                |
|--------------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| differential_drive | twist_to_motors | transforms the required linear velocity and angular velocity (Twist) of the DD robot to required velocity of motors.                                                    | <code>~rate</code> (float, default:10)                    |
|                    |                 |                                                                                                                                                                         | <code>ticks_meter</code> (int, default:50)                |
|                    |                 |                                                                                                                                                                         | <code>~base_width</code> (float, default:0.245)           |
|                    |                 |                                                                                                                                                                         | <code>~base_frame_id</code> (string, default:"base_link") |
|                    |                 |                                                                                                                                                                         | <code>~odom_frame_id</code> (string, default:"odom")      |
|                    |                 |                                                                                                                                                                         | <code>encoder_min</code> (int, default:-32768)            |
|                    |                 |                                                                                                                                                                         | <code>encoder_max</code> (int, default:32768)             |
|                    |                 |                                                                                                                                                                         | <code>wheel_low_wrap</code>                               |
|                    |                 |                                                                                                                                                                         | <code>wheel_high_wrap</code>                              |
|                    | pid_velocity    | The controller using feedback from encoder and set point from twist_to_motors and outputs the PWM to the motor                                                          | <code>~Kp</code> (float, default:10)                      |
|                    |                 |                                                                                                                                                                         | <code>~Ki</code> (float, default:10)                      |
|                    |                 |                                                                                                                                                                         | <code>~Kd</code> (float, default:0.001)                   |
|                    |                 |                                                                                                                                                                         | <code>~out_min</code> (float, default:-255)               |
|                    |                 |                                                                                                                                                                         | <code>~out_max</code> (float, default:255)                |
|                    |                 |                                                                                                                                                                         | <code>~rate</code> (float, default: 20)                   |
|                    |                 |                                                                                                                                                                         | <code>~rolling_pts</code> (float, default: 2)             |
|                    |                 |                                                                                                                                                                         | <code>~timeout_ticks</code> (int, default: 2)             |
|                    |                 |                                                                                                                                                                         | <code>ticks_meter</code> (float, default: 20)             |
|                    |                 |                                                                                                                                                                         | <code>encoder_min</code> (int, default:-32768)            |
|                    |                 |                                                                                                                                                                         | <code>encoder_max</code> (int, default:32768)             |
|                    |                 |                                                                                                                                                                         | <code>wheel_low_wrap</code>                               |
|                    |                 |                                                                                                                                                                         | <code>wheel_high_wrap</code>                              |
|                    | diff_tf         | produces the axes transformations of the DD robot due to the movement which detected by encoders, and publishes the odometry (the current pose and twist) of the robot. | <code>~rate</code> (float, default:10)                    |
|                    |                 |                                                                                                                                                                         | <code>ticks_meter</code> (int, default:50)                |
|                    |                 |                                                                                                                                                                         | <code>~base_width</code> (float, default:0.245)           |
|                    |                 |                                                                                                                                                                         | <code>~base_frame_id</code> (string, default:"base_link") |
|                    |                 |                                                                                                                                                                         | <code>~odom_frame_id</code> (string, default:"odom")      |
|                    |                 |                                                                                                                                                                         | <code>encoder_min</code> (int, default:-32768)            |
|                    |                 |                                                                                                                                                                         | <code>encoder_max</code>                                  |
|                    |                 |                                                                                                                                                                         | <code>wheel_low_wrap</code>                               |
|                    |                 |                                                                                                                                                                         | <code>wheel_high_wrap</code>                              |

- 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 | <b>_speed</b> is the required linear velocity |
|                          |                       |                                                                                   | <b>_turn</b> is the required angular velocity |

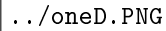
- topics

| 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**

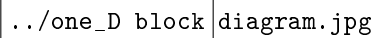
Purpose of this experiment is to validate ROS performance in aerospace applications and compare its performance with Ardupilot.



.. /oneD.PNG

- **Configuration 1 (ROS pid)**

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



.. /one\_D block diagram.jpg

Figure 5.2.4: 1D Bi-rotor conf1 Block Diagram

\AM@currentdo

.pdf



- **Configuration 2 (myPIDA)**

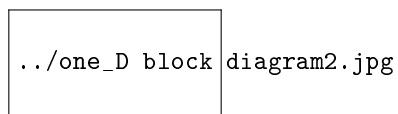


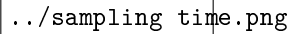
Figure 5.2.5: 1D Bi-rotor conf2 Block Diagram

| 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



../sampling time.png

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

- IMU Bias

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

- Gains tuning

## 5.3 px4 and Mavros

### 5.3.1 Introduction

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

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

- **What's Pixhawk**

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

- **Other devices like Pixhawk**

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

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

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

- **Comparison between various autopilot devices**

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

- **Why Pixhawk**

We choose the Pixhawk over Arducopter for the following reasons

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

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

### 5.3.2 User guides to Pixhawk

- **Step 1: Wiring and Connections**

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

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

../documentation3/pixhawk/powermodule-analog-pixhawk.png

- **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

../documentation3/pixhawk/maxresdefault.jpg

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

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

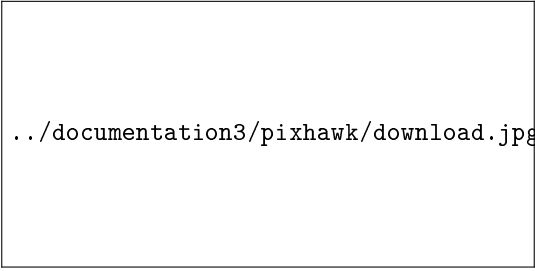
../documentation3/pixhawk/jDrones\_PPM\_Encoder\_V21\_Above\_with\_numbe

- **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.



../documentation3/pixhawk/download.jpg

- **Android USB cable:**

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

- **Step 2: Connect to PC**

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

- **Step 3: Ground Station**

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


- **Step 4: Firmware selection**

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

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

- **Step 5: Frame Selection**

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



../documentation3/pixhawk/AQ-All-Frames.jpg

- 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.

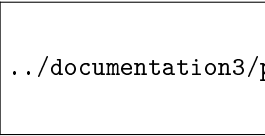
../documentation3/pixhawk/cae0384839ae0630258b3882725779b08ab5a024

- 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 stabilize 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.



../documentation3/pixhawk/px4\_copter\_basic.jpg

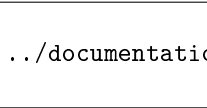
In our case the Hover Throttle is set to 80% because the power supply is not strong enough to provide the ESCs with the power needed.

The Manual minimum throttle is set to 8%.

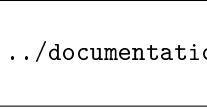
- The next step is going into the advanced PID tuning of the vehicle by clicking on “advanced”.

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.

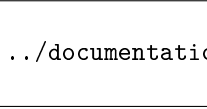
- A low stabilize P will cause the copter to rotate very slowly and may cause the copter to feel unresponsive and could cause a crash if the wind disturbs it. Try lowering the RC\_Feel parameter before lowering Stability P if smoother flight is desired.
- Rate Roll/Pitch P, I and D terms control the 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.



../documentation3/pixhawk/pitch.PNG



../documentation3/pixhawk/rolling.PNG



../documentation3/pixhawk/yaw2.PNG

# References



## Appendix A

# ROS Examples

A.1 publisher.py

A.2 subscriber.py

A.3 publisher.cpp

A.4 subscriber.cpp

A.5 ROS services using python

A.5.1 Server

A.5.2 node uses server

A.6 ROS actions using python

A.6.1 Action server

A.6.2 node uses action