

## **UAV Automation**

A Report Submitted in Partial Fulfilment of the Requirements

of the Internship of

**Technical Intern**

by

**Khush Vasudeo Patil**

Intern ID: INTN 10

Under the guidance of

**Dr. Abhishek Ghosh Roy**

**(Chief Technical Officer)**

**Paras Aerospace Pvt Ltd.**

**Duration:** 6 Months, From 23<sup>th</sup> January to 23<sup>th</sup> July, 2023



Paras Aerospace Pvt Ltd.

D-112, TTC Industrial Area, MIDC,

Nerul, Navi Mumbai, Maharashtra 400706

## **CERTIFICATE**

This is to certify that, **Mr. Khush Vasuedo Patil** student of **Electronics & Telecommunication Department (B.E. Electronics & Telecommunication Engineering)** of **University of Mumbai, Mumbai** has satisfactorily completed the Industrial Internship at **Paras Aerospace Pvt Ltd.** for the duration of 6 Months, from 23<sup>th</sup> January to 23<sup>th</sup> July to and found sincere and hardworking in tenure period of training.

Countersigned by,

**Date:**

**Dr. Abhishek Ghosh Roy**  
**(CTO)**

**Stamp**

## **ACKNOWLEDGEMENT**

Project training is truly a wonderful opportunity to learn the practical aspects of the theoretical concepts taught in our respective colleges. It's truly an honour to have been granted this opportunity to work as a trainee at a reputed organization such **Paras Aerospace Pvt Ltd**. I would like to express my gratitude towards all who have helped me on this journey and helped me undergo the training program practically with their encouragement and support.

I am highly thankful to mentor **Dr. Abhishek Ghosh Roy (CTO)**, my Mentor, for her continuous and valuable support and guidance throughout the period of this internship and suggestions, and helping me complete this training successfully at **Paras Aerospace Pvt Ltd**.

I would like to thank **Ms. Sujata Ghorai (HR)** and all the personnel of Paras Aerospace Pvt Ltd. and all the team members of the company for their support.

I would like to thank **Mr. Abhishek Haridass (Back End Developer)** and **Mr. Swapnil Mathur (Intern)** for their help and support during this Internship period.

I extend my gratitude to **Mr. Pankaj Akula (CEO)**, **Mr. Praveena Kokrady** and **Mr. Saurabh Srivastava** for providing me an amazing opportunity and learning experience.

**Khush Vasudeo Patil**

## Table of Content

<b>S No.</b>	<b>Topic</b>	<b>Page No.</b>
Chapter 1	About the Organisation	5
Chapter 2	Introduction to UAV	6
Chapter 3	Glenn Research Centre, NASA	13
Chapter 4	Drone Simulation Using MATLAB	15
Chapter 5	Mission Planner and SITL	23
Chapter 6	Version Control Software	27
Chapter 7	Robot Operating System (ROS)	30
Chapter 8	Linear Algebra using various Programming Languages	36
Chapter 9	ArduPilot SITL using with RealFlight	50
Chapter 10	Flight Dynamic Model of an Aircraft	62

## **Chapter 1: About the Organisation**

### **1.1 Paras Aerospace Pvt Ltd.**

Paras Aerospace Pvt. Ltd. is a subsidiary of Paras Defence and Space Technologies Ltd. The goal is to be the foremost Indigenous Technology Development firm providing both hardware and software solutions for UAVs in India and worldwide. The services encompass Development, Integration, Manufacturing, and Certification of UAV systems, with a focus on Military UAV, Industrial UAV, Indigenous Payload Development, Regulatory Compliance Consultancy, and an Agriculture focused UAV. The company offers end-to-end solutions across several industries, including NPNT Compliance Software, Perimeter Inspection Solutions, Agri-tech Solutions, Construction Monitoring, Energy (Power Grid, Solar & Wind), Forestry, Plant Inspection (Oil and Gas), and Engineering Consultancy for businesses interested in Drone Automation. The company has collaborated with top UAV Technologists from Israel and Italy and offer a Cloud-based NPNT As a Service Software and Indigenous Multispectral Camera as the flagship products.

### **1.2 Paras Defence and Space Technologies**

Paras Defence and Space Technologies Limited, headquartered in Mumbai, is the parent company comprising a group of firms focused on designing, manufacturing, testing, and commissioning products, systems, and solutions for Defence and Space Applications. With 30 years of sustained growth, the group is dedicated to Defence Engineering and offers the broadest range of products and solutions under one roof. The organization, with its exclusive Manufacturing setup, is associated with the most prestigious Electro-optics and Space programs in the country. The optics facilities boast an unparalleled manufacturing setup that enables large-scale production of ultra-precision optics. The Electronics Division, one of the oldest Defence Electronics groups in the country, has made significant contributions to major defence programs for Land, Naval, and Aerospace applications. The Heavy Engineering Group, the company's oldest arm, has exceptional engineering capabilities that have enabled import substitution and "MAKE IN INDIA" achievement for over three decades.

## Chapter 2: Introduction to UAV

### 2.1 Unmanned Aerial Vehicles (UAV)

An unmanned aerial vehicle (UAV), is a type of aircraft that is operated without a human pilot on board. They are controlled remotely or autonomously, and can be used for a variety of purposes such as military, farming, mapping, industrial, surveillance, reconnaissance, and delivery. UAVs come in a wide range of sizes and designs. UAVs can be remarkably efficient, offering substantially greater range and endurance than equivalent manned systems. UAVs are descended from target drones and remotely piloted vehicles (RPVs).

There are many different types of UAVs, each designed for a specific purpose or set of tasks. Some common types include:

- ❖ Fixed-wing UAVs: These have a traditional aircraft-like design and are typically used for long-range missions, such as surveillance and reconnaissance.
- ❖ Rotary-wing UAVs: These have a helicopter-like design and are able to hover in one spot, making them useful for tasks such as search and rescue and aerial photography.
- ❖ Hybrid UAVs: These combine the characteristics of both fixed-wing and rotary-wing UAVs and have the ability to take off and land vertically like a helicopter and fly horizontally like a fixed-wing aircraft.
- ❖ Micro UAVs: These are small UAVs, typically weighing less than 2 kg, and are used for tasks such as reconnaissance and surveillance in urban areas or indoor environments.
- ❖ Nano UAVs: These are even smaller than micro-UAVs and are capable of flying through tight spaces and performing tasks such as reconnaissance and surveillance in confined areas.
- ❖ Large UAVs: These are larger UAVs and are used for tasks such as cargo delivery and heavy payloads transportation.
- ❖ Autonomous UAVs: These UAVs can fly and navigate on their own, without human intervention. They use sensors, GPS, and software to navigate and perform tasks.
- ❖ Manned UAVs: These are UAVs that can be flown with a human pilot on board.

Figure (a) and (b) show fixed wing hybrid VTOL type UAV and Quadrotor type UAV respectively.



**Figure 1: Types of UAV**

## **2.2 Unmanned Aerial System**

UAS (Unmanned Aerial System) is the entire package needed to operate the system, which includes the UAV itself, the ground control system, camera, GPS, all the software, skills needed to operate the system and tools required for maintenance. These systems can be used in a variety of applications such as surveying, film making and by the military for gathering information.

## **2.3 Ground Control Station (GCS)**

A Ground Control Station (GCS) is a control centre for UAVs, which allows an operator to control the UAV, monitor its flight, and receive sensor data from the UAV. The GCS is typically used to plan, launch, and recover the UAV, as well as to monitor its flight and payload performance.

A typical GCS includes the following components:

- ◆ Control console: This is the main interface used by the operator to control the UAV and monitor its flight. It can include a joystick, keyboard, and display screens.
- ◆ Flight planning software: This allows the operator to plan the UAV's flight path, waypoints, and other mission parameters.
- ◆ Communication system: This allows the UAV to communicate with the GCS and other UAVs, and may include a radio or satellite link.

- ◆ Data processing software: This allows the operator to process and analyse the sensor data received from the UAV.
- ◆ Power supply: This provides power to the GCS.
- ◆ Safety systems: This includes emergency procedures, telemetry data backup and other safety features

A GCS can be used to control one or multiple UAVs at a time. The GCS can be located on the ground, in a vehicle, or on a aircraft, depending on the mission requirements.

In summary, a GCS is a control centre for UAVs, which allows an operator to plan, launch, monitor, and recover the UAV, as well as to receive and process sensor data from the UAV.

Intelligence.

## 2.4 Main Components of a Quadcopter

### 1. Propellers/ Wings

The Drones/ UAVs use either propellers/ wings or both (depending on the availability) to direct them. Propeller driven Drones have two types of propellers onboard for direction and thrust. These are

- Standard Propellers
- Pusher Propellers



**Figure 2: Quadcopter**

**Standard Propellers:** These are located in the front of Quadcopter. These propellers provide direction to the Drone.

**Pusher Propellers:** These are located in the back of Quadcopter. These propellers provide forward and backward thrust to the Drone.

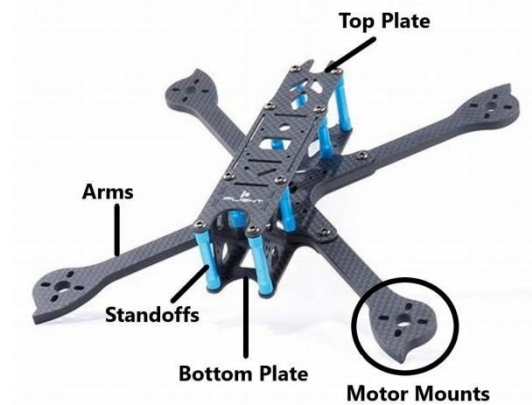


## 2. Chassis

This is the main body of Quadcopter which provides the housing facility to all other components.

## 3. Flight Controller

The flight controller is usually referred to as the brain of the drone. The flight controller controls the power supply to the electronic speed controller. It is also used to detect orientation changes in the drone. It controls the motors and ensures that the drone is in air.



**Figure 3: Quadcopter Chassis**

## 4. Electronic Speed Controller

Electronic speed controllers (ESC) are the electronic circuits that regulate the speed of the DC motors. It also provides dynamic braking and reversing options. It receives signals from the flight controller and adjusts the power going to the motors to control the speed and rotation of the propellers.

The flight controller sends control signals to the ESCs which in turn adjust the speed of the electric motors. This is done by controlling the amount of power that is delivered to each motor, which in turn affects the speed of the propellers. The ESCs also provide protection for the motors and battery by implementing safety features such as over-current protection, temperature protection, and low-voltage cut-off.

## 5. DC Motors

To ensure that the drone is airborne for a good amount of time, we need high torque motors. The high torque also helps to change the speed of the propellers. Brushless DC motors are preferred as they are lighter than the brushed ones.

## 6. Landing Gear

Landing gears are not required for small drones. However, bigger drones need a landing gear to avoid any damage while landing. The requirement of landing gear varies with functionality of the drone. For example – Delivery drones which carry parcels require a spacious landing gear as they need space to hold the items.

## 7. Transmitter

The transmitter sends the signals from controller to the drone to generate command of direction and thrust.

## 8. Receiver

The receiver receives the signals sent by the transmitter and passes it to Flight Controller PCB.

## 9. GPS Module

The GPS module provides the navigational data (longitude, latitude and elevation) to the operator. This module assists the controller in recognizing the taken path and safely return to the initial point in case of lost connection.

## 10. Battery

It provides power to the drone. Generally, rechargeable battery is used in drone.

A battery failsafe system is a critical component in ensuring the safe operation of unmanned aerial vehicles (UAVs). The main goal of a battery failsafe system is to prevent the UAV from crashing or otherwise malfunctioning due to a battery failure. Some common features of battery failsafe systems include Low battery voltage warning, Automatic return-to-home, Battery capacity monitoring, Battery over-temperature protection. Additionally, some UAVs have redundant batteries to ensure safe landing in case of primary battery failure.

## 11. Camera

There is normally an attached inbuilt camera with drones. A camera is a common payload for UAVs, and it is used for a variety of applications such as aerial photography, surveillance, and reconnaissance.



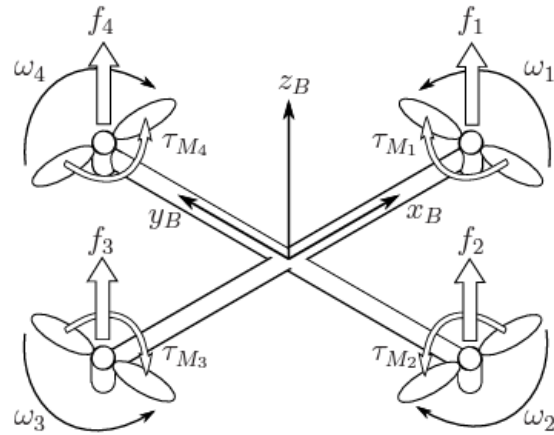
**Figure 4: Gimbal Cam with Quadcopter**

## 2.5 Quadcopter Dynamics

When a quadcopter's propellers spin, they push air downward. Using Newton's third law, this represents the action. For Newton's law to be true, there must be an equal and opposite reaction.

This reaction is an upward force pushing on the quadcopter. Once this force exceeds the force of gravity pulling the quadcopter downward, the quadcopter begins to move up.

The dynamics of a quadcopter are controlled by the flight controller, which receives input from various sensors such as accelerometers, gyroscopes, and GPS to determine the quadcopter's attitude and position. The flight controller then sends signals to the ESCs (Electronic Speed Controllers) to adjust the speed of the motors and hence control the thrust generated by each rotor.



**Figure 5: Body frames of a quadcopter**

The basic working principle of a quadcopter is that it uses the thrust generated by its four rotors to achieve lift and propulsion. By varying the speed of each rotor, the quadcopter can control its pitch, roll, yaw, and altitude. This allows the quadcopter to move in any direction, hover in one spot, or even fly upside down.

The drone rotates in the opposite direction to the net rotational direction of its propellers. Thus, by decreasing the speed of the clockwise-rotating propellers, the whole drone rotates clockwise (opposite to the faster counter-clockwise propellers). The yaw of the drone has its own control stick.

Quadcopter use a combination of sensors and control algorithms to achieve stability and control. The control algorithm receives sensor data and compute the necessary control signal to send to the ESCs. This control algorithm is often implemented using a combination of Proportional-Integral-Derivative (PID) controllers and Linear Quadratic Regulator (LQR) controllers.

In summary, a quadcopter's dynamics and working principle is based on the control of thrust generated by each rotor and the control of attitude and position by the flight controller using sensor data and control algorithms.

## 2.6 Importance of Dynamics

As shown in the figure, to move from point A to point B, the speed of the propellers is to change in such sequences and combination. This will allow the quadcopter to move in space. The

change in propeller speed will directly change the acceleration of the quadrotor. Mathematically, this depends on Mass, Size and Shape. We can call it as the **Moment of Inertia Matrix**. Some other important mathematical parameters are Newton-Euler equation, Forces and Moments and Controller Inputs.



**Figure 6: Movement of a Quadrotor**

Flight dynamics play a critical role in the design and operation of quadcopters, which are a type of unmanned aerial vehicle (UAV) that uses four rotors for propulsion and control. Some of the key importance of flight dynamics in quadcopters include:

1. **Stability:** Flight dynamics are used to ensure that the quadcopter remains stable and flies in a consistent manner, even in the presence of external disturbances such as wind or turbulence.
2. **Control:** Flight dynamics are used to design the control systems that allow the operator to control the quadcopter's movement, including its altitude, speed, and direction.
3. **Agility:** Flight dynamics are also used to design the quadcopter's control systems to enable it to perform complex manoeuvres such as hovering, fast forward flight, and tight turns.
4. **Safety:** Flight dynamics play a critical role in ensuring the safety of the quadcopter and its surroundings. By understanding the quadcopter's behaviour in different flight conditions, engineers can design control systems that can respond appropriately in case of failures or unexpected events.
5. **Efficiency:** Flight dynamics can also be used to optimize the quadcopter's design for energy efficiency, which can increase the flight time of the quadcopter.

## Chapter 3: Glenn Research Centre, NASA

The Glenn Research Centre (GRC) of NASA has a long history of research and development in the field of unmanned aerial vehicles (UAVs) also known as drones. The centre conducts research on advanced propulsion systems, aerodynamics, and control systems for UAVs, with a focus on improving their performance, efficiency, and safety.

The Glenn Research Centre (GRC) of NASA has a long-standing research program in the field of Electric Aircraft Propulsion (EAP). The goal of this program is to develop advanced electric propulsion systems for aircraft, including UAVs, with a focus on increasing their efficiency, reducing their noise, and reducing their environmental impact.

Some of the specific areas of UAV research at GRC include:

- ❖ **Electric propulsion systems:** GRC researchers are working on developing advanced electric propulsion systems for UAVs, with a focus on increasing their efficiency and reducing their weight.
- ❖ **Autonomous systems:** GRC is researching on the development of advanced autonomous systems for UAVs, including guidance, navigation, and control systems, to enable them to fly safely and efficiently.
- ❖ **Aerodynamics:** GRC researchers are studying the aerodynamics of UAVs to improve their flight performance and stability, and to reduce their noise.
- ❖ **Flight testing:** GRC has facilities for testing UAVs, including the Lewis Field, which is used to test aircraft and aircraft systems, including unmanned aerial systems and other advanced aircraft technologies.
- ❖ **Power and energy storage:** GRC is also researching on advanced power and energy storage systems for UAVs, including batteries and fuel cells, to improve their endurance and range.

Overall, GRC is using its expertise in aeronautics, propulsion, and autonomous systems to advance the state of the art in UAV technology and to support the development of new UAV applications. The figure shows the communication links between ground stations, airports, satellites and unmanned aerial vehicles.

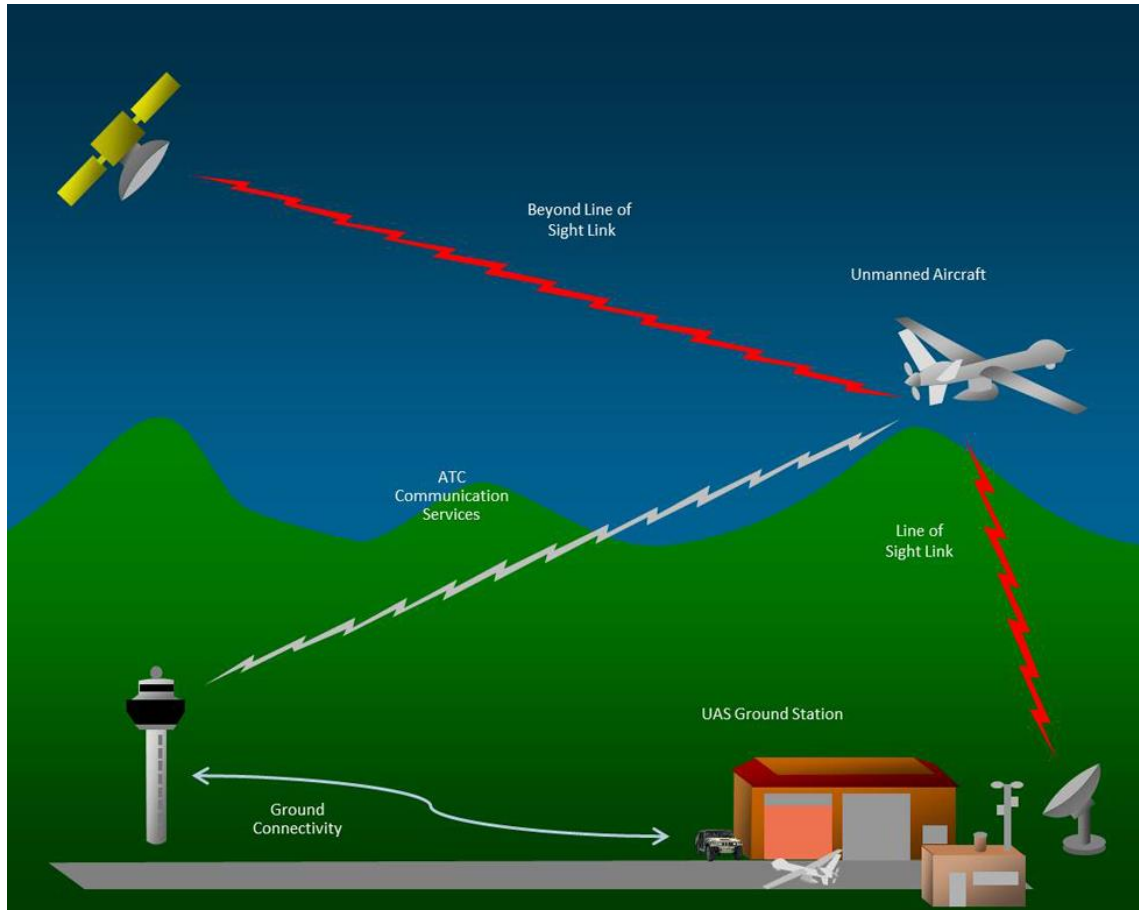


Figure 7: Glenn Research Centre, NASA

## Chapter 4: Drone Simulation Using MATLAB

### 4.1 Introduction

Quadcopters are becoming increasingly popular for a wide range of applications, from aerial photography and surveillance to package delivery and search and rescue missions. The ability to simulate a quadcopter in a virtual environment can provide valuable insights into the dynamics and performance of these aircraft, as well as help to optimize the design and control of quadcopters. In this simulation, we will use MATLAB and Simulink to model a quadcopter and simulate its behaviour in response to different control inputs. The simulation will take into account the dynamics of the rotor blades, the control systems for the quadcopter, and the interactions between different subsystems. We will also consider the yaw, pitch, and roll of the quadcopter, and how these are affected by the control inputs. The simulation results will be analysed and visualized using the MATLAB plotting tools to gain a better understanding of the quadcopter's behaviour and performance.

### 4.2 Simulation

#### 1. Starting the simulation: Entering the code in command window

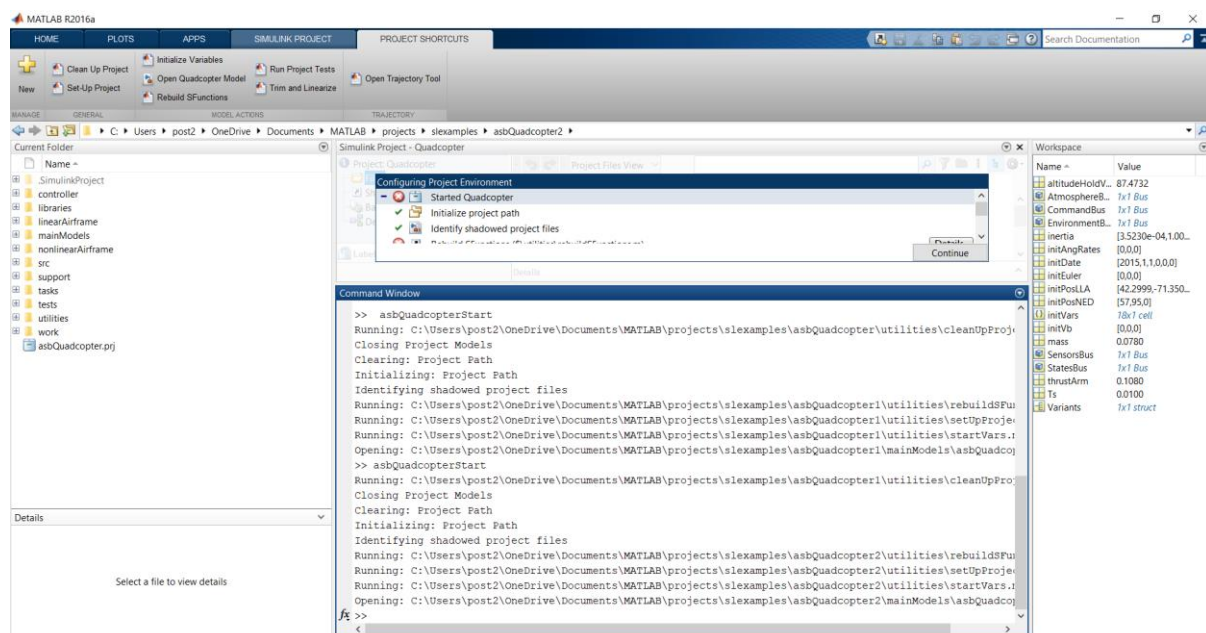


Figure 8: MATLAB Simulation

#### 2. Command Window opens Simulink

As shown in the figure, the following blocks are added and connected

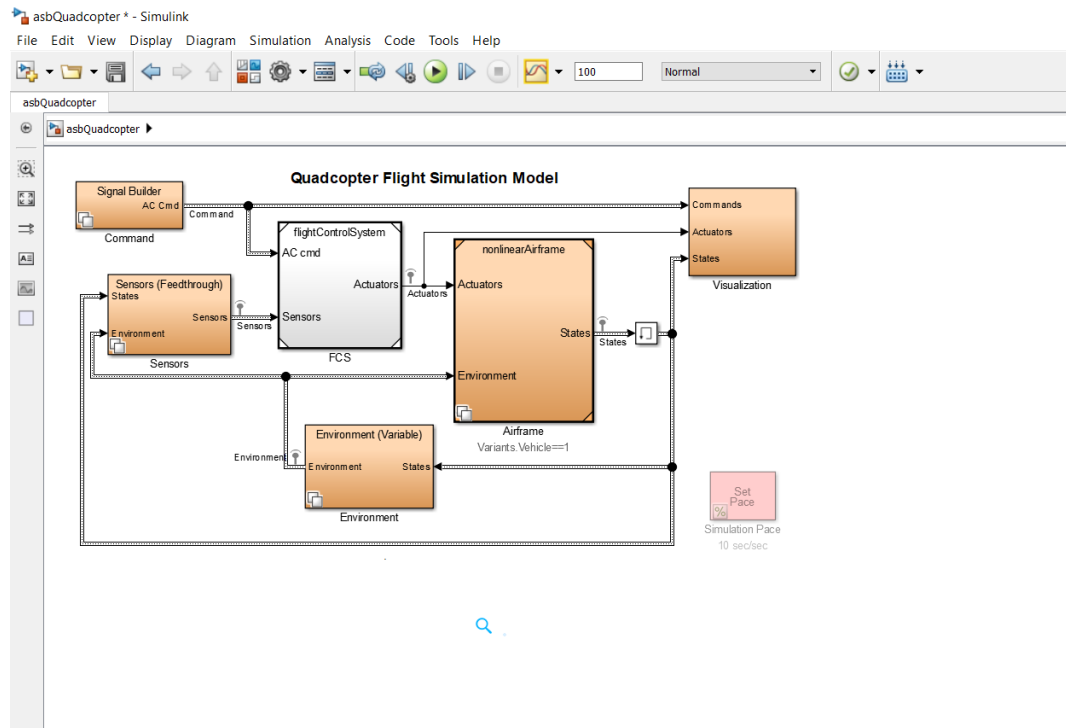


Figure 9: MATLAB Simulation

### 3. Setting Block Parameters

#### a. Signal Builder

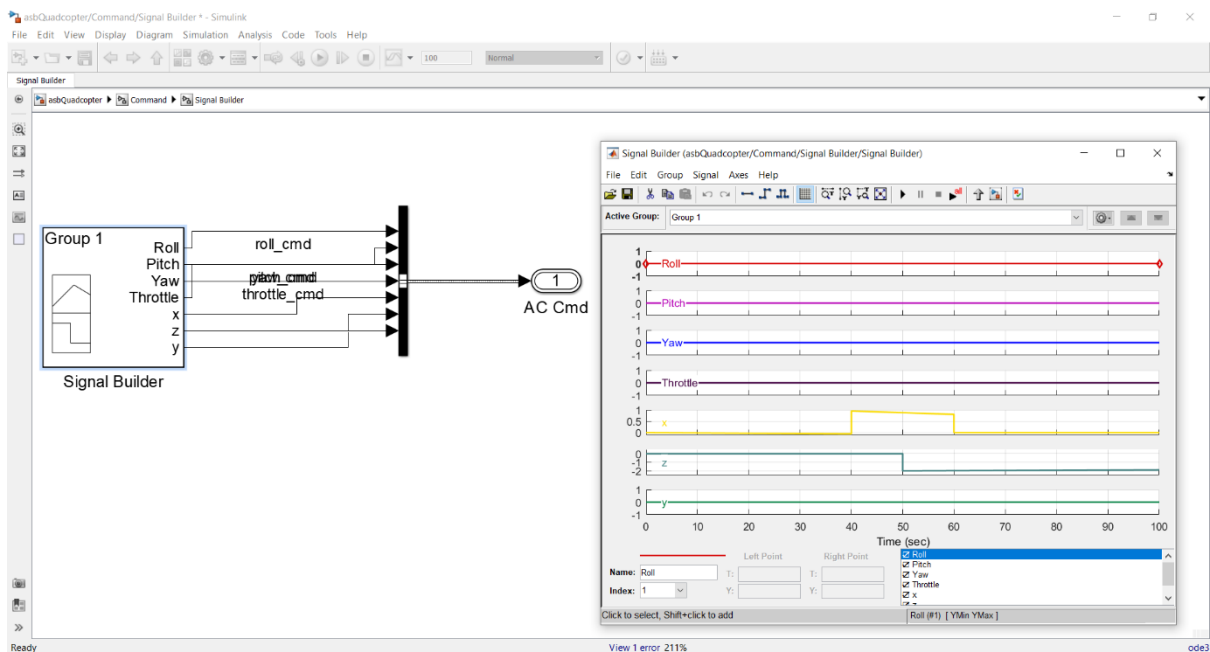
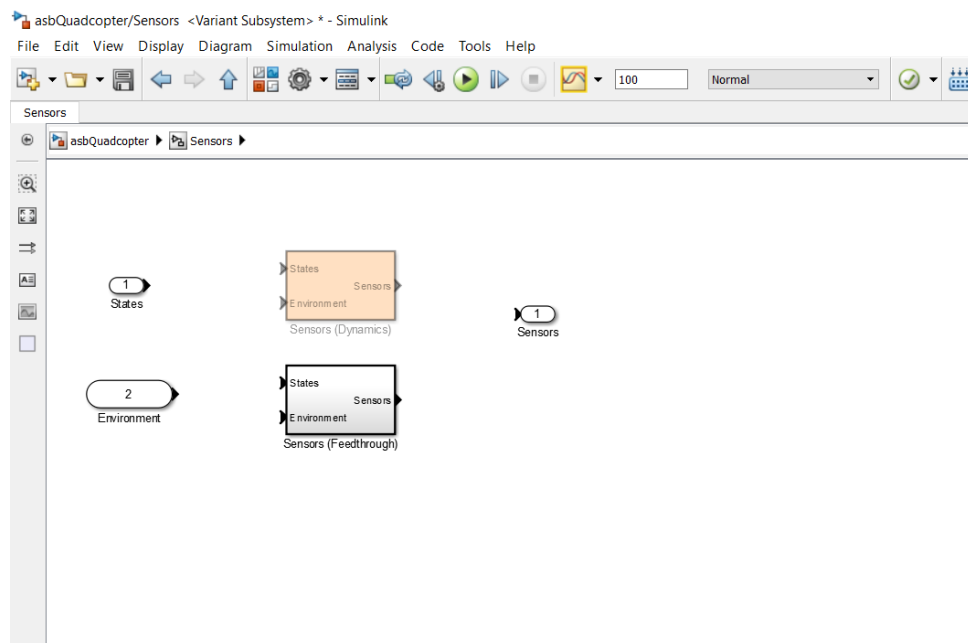


Figure 10: MATLAB Simulation

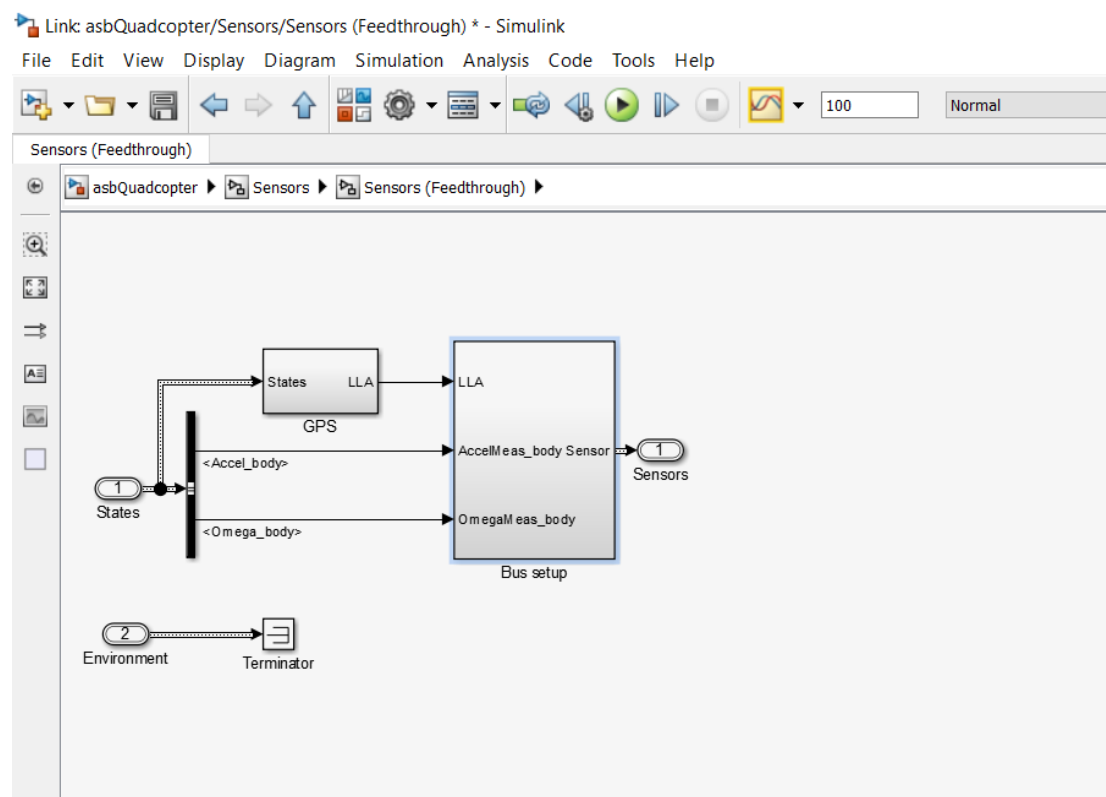


## b. Sensors

Blocks are automatically connected during simulation.



**Figure 11: MATLAB Simulation**



**Figure 12: MATLAB Simulation**

### c. Flight Control System

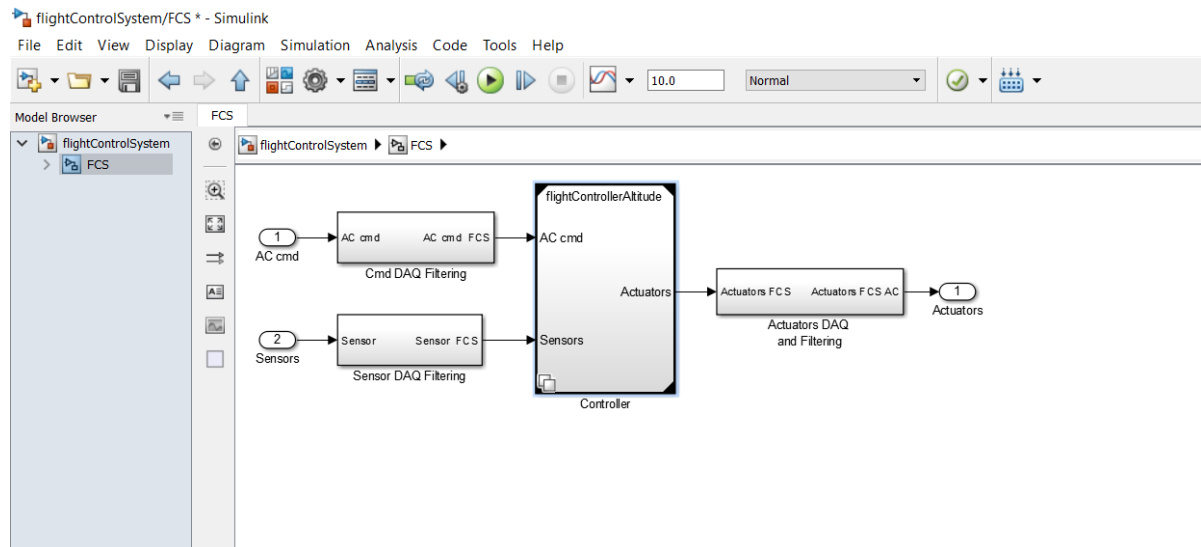


Figure 13: MATLAB Simulation

### d. Environment

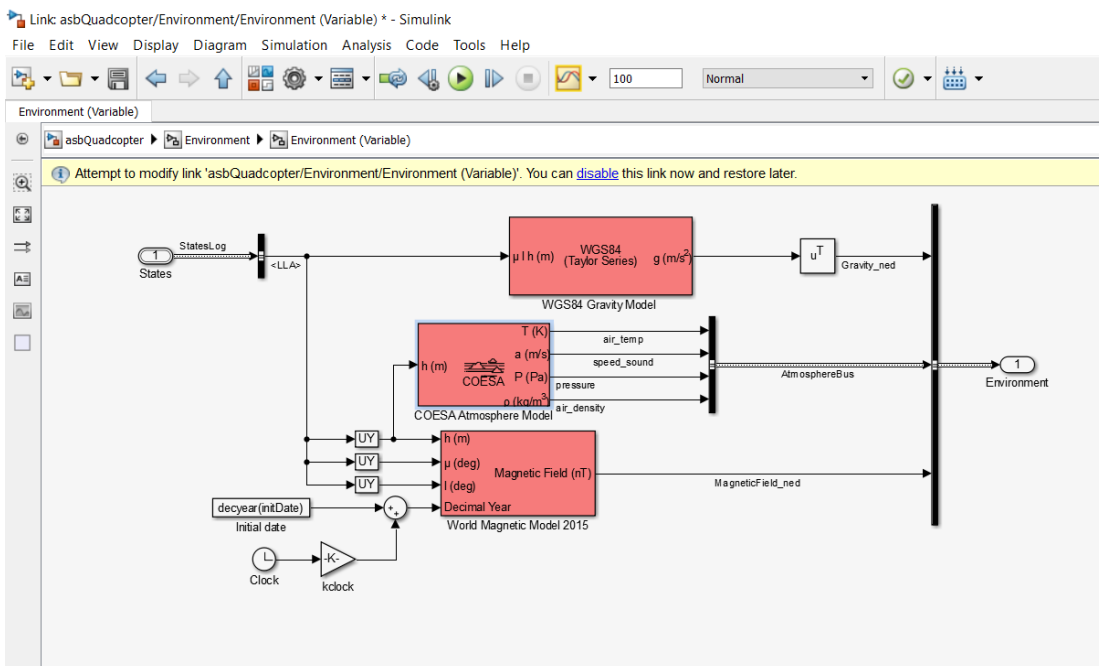


Figure 14: MATLAB Simulation

## e. Airframe

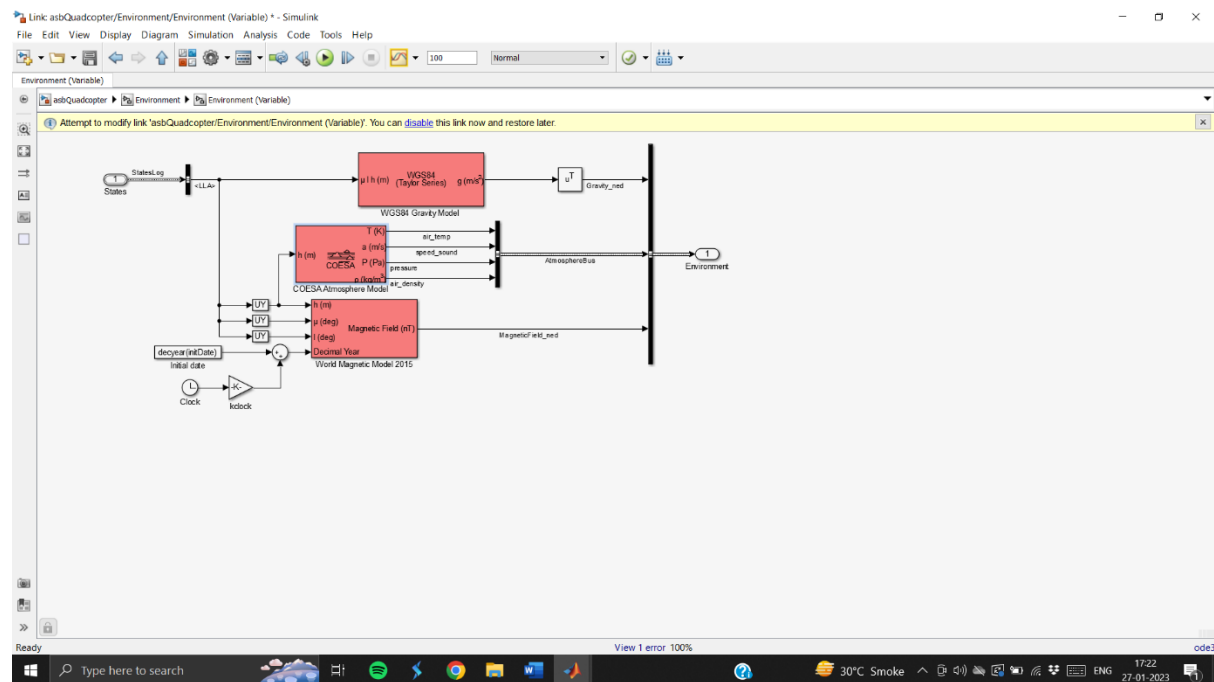


Figure 15: MATLAB Simulation

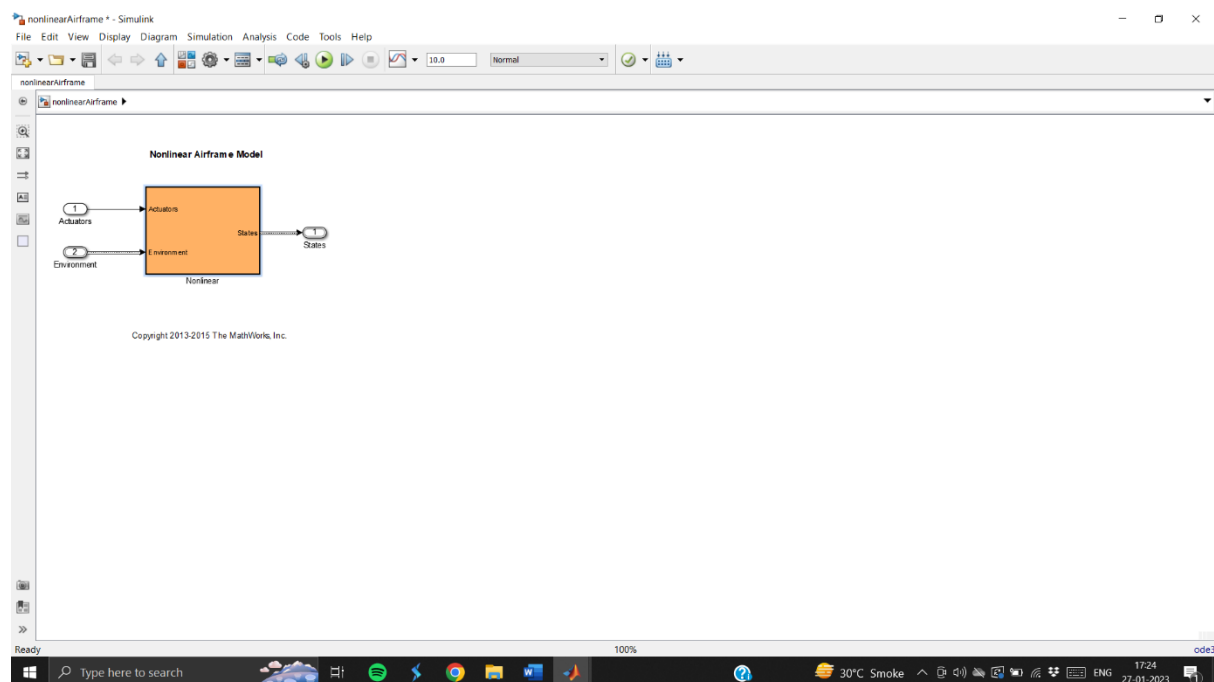


Figure 16: MATLAB Simulation

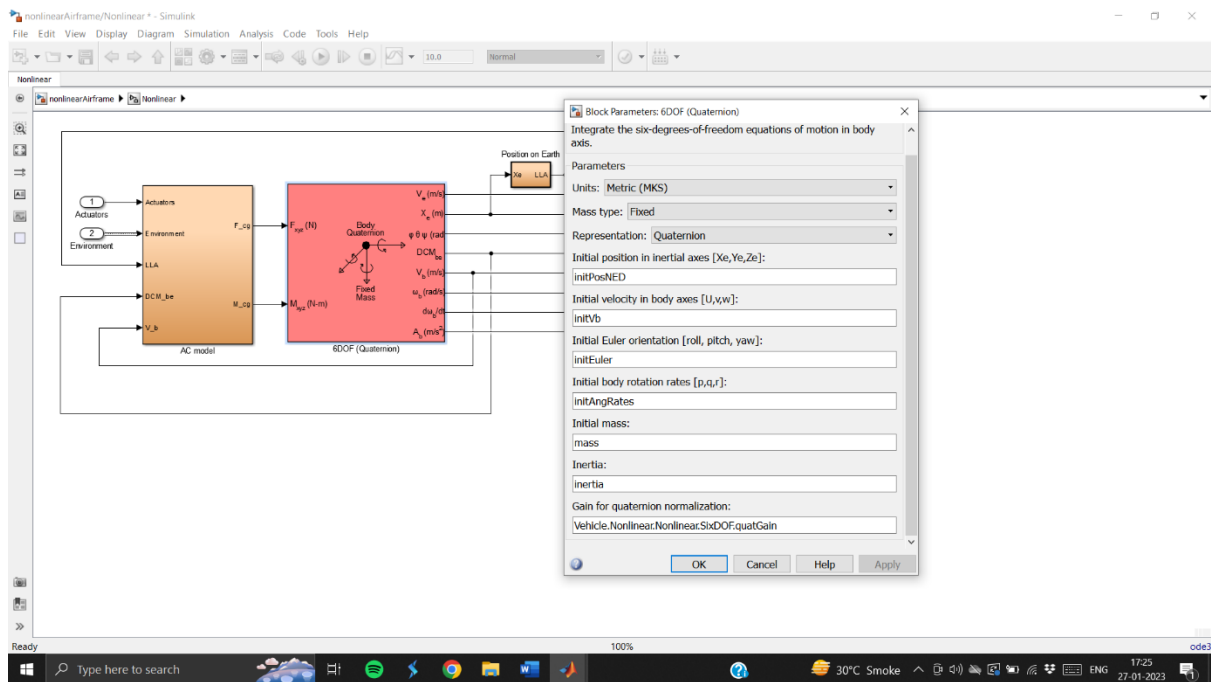


Figure 17: MATLAB Simulation

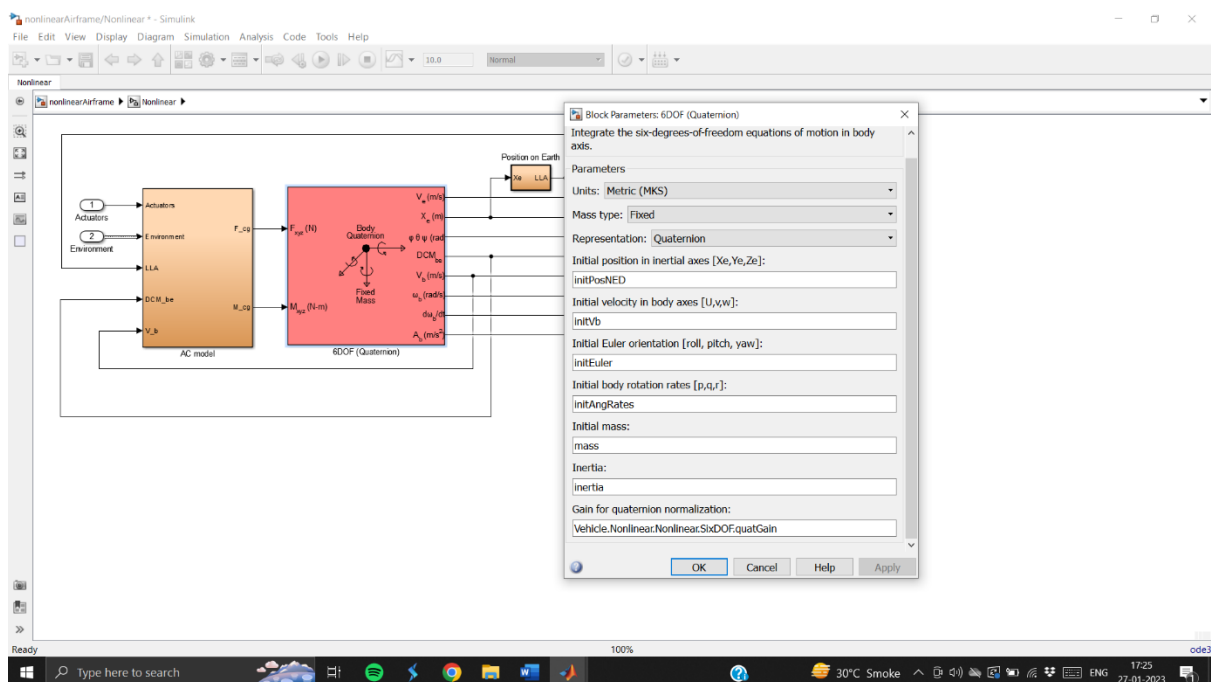


Figure 18: MATLAB Simulation

## f. Visualisation

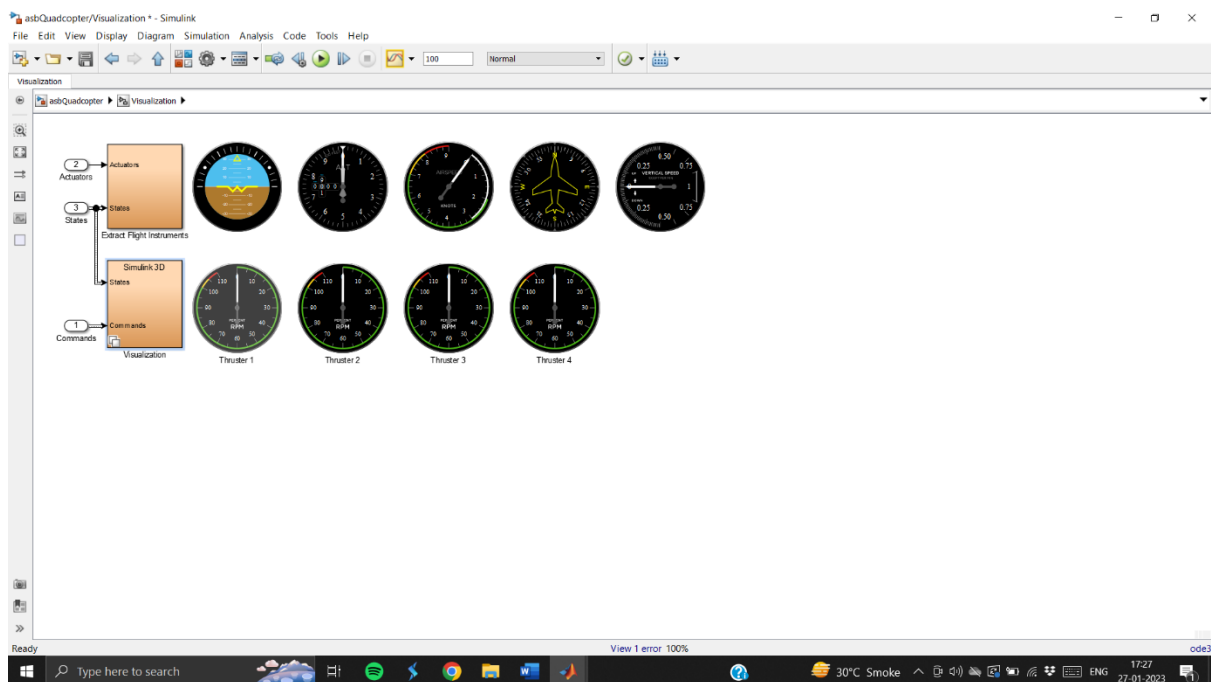


Figure 19: MATLAB Simulation

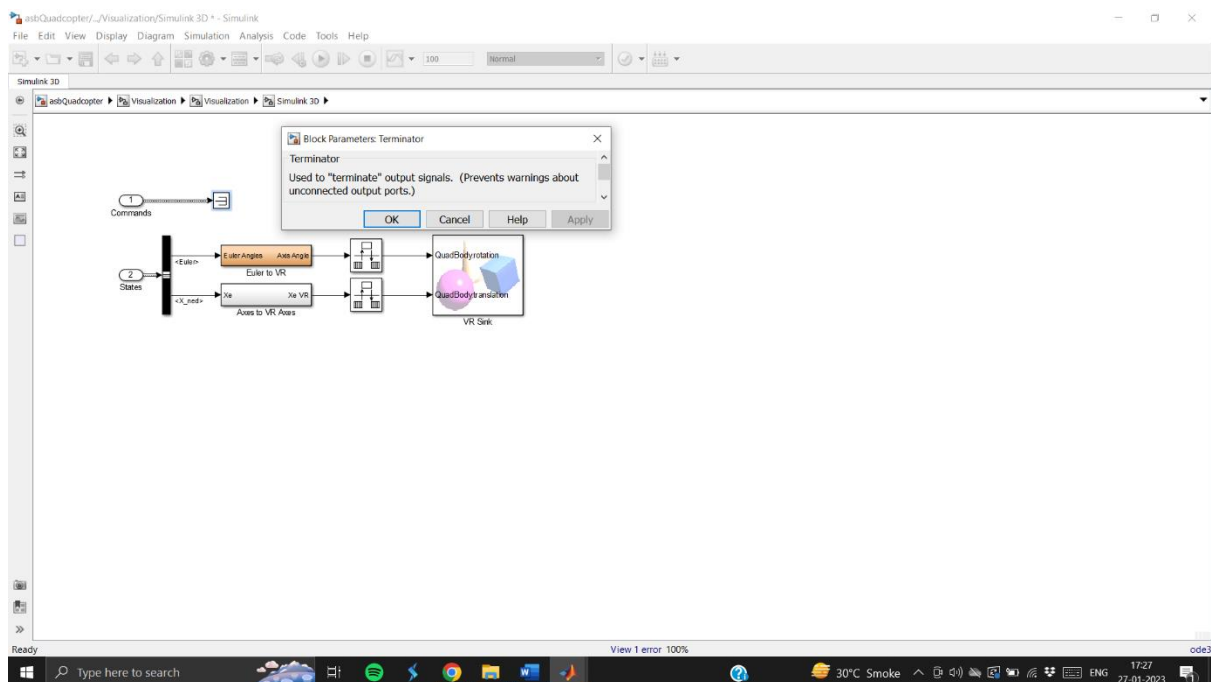


Figure 20: MATLAB Simulation

## Quadcopter Simulation

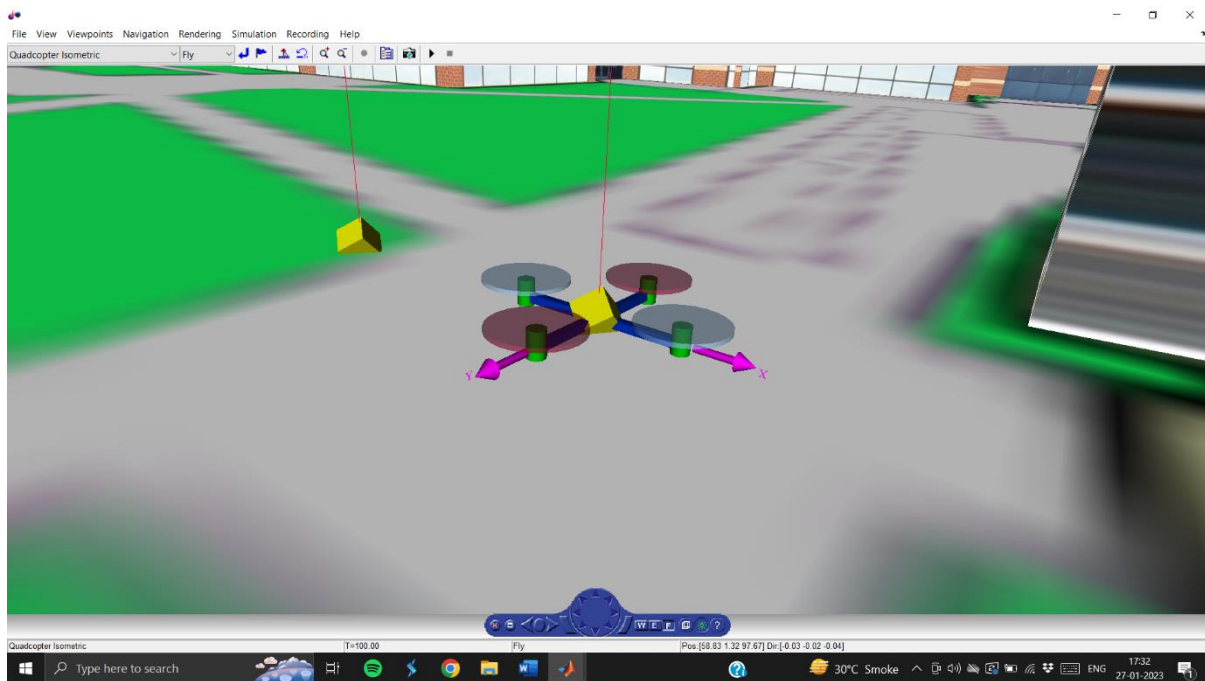


Figure 21: MATLAB Simulation

## **Chapter 5: Mission Planner and SITL**

### **5.1 Software In the Loop (SITL)**

Software-in-the-loop (SITL) is a simulation testing method that involves executing the software component of a system in a controlled environment, typically as a standalone simulation or a virtual device. This allows developers to test the software and verify its functionality before integrating it with the rest of the system. SITL testing is commonly used in the development of embedded systems, control systems, and real-time systems.

SITL (Software-in-the-loop) in ArduPilot refers to a simulation environment that allows developers to test and verify the functionality of the ArduPilot software without the need for physical hardware. SITL provides a virtual representation of the flight hardware and allows developers to run simulations of the autopilot software in a controlled environment, thereby reducing the cost and time required for physical testing. SITL is widely used by the ArduPilot development community to test and debug new features and improvements before they are integrated into the main codebase.

### **5.2 Hardware In the Loop (HITL)**

Hardware-in-the-loop (HITL) simulation is a testing method in which real hardware components are integrated with a simulated environment to test the hardware's interaction with the rest of the system. In HITL simulation, the hardware component is connected to a simulation model that represents the other components of the system, allowing the hardware to be tested in a realistic, real-time environment. HIL simulation is commonly used in the development of real-time systems to validate the performance of hardware components and to identify and correct any issues before the hardware is integrated into the final system.

### **5.3 Mission Planner**

Mission Planner is a ground control software (GCS) developed by the ArduPilot project. It is used to plan, monitor, and control unmanned aerial vehicles (UAVs) running the ArduPilot autopilot system. Mission Planner provides a user-friendly graphical interface for setting up and configuring the autopilot, as well as for monitoring the UAV's performance during flight. The software supports a wide range of features, including waypoint navigation, real-time telemetry, advanced tuning and analysis tools, and more. Mission Planner is widely used in the UAV community and has a large and active user community.

### **5.4 Simulation of an ArduCopter in Mission Planner**

The aim is to simulate a quadcopter on a spiral path of outer diameter of 500 meters. The vehicle has to start from the centre, then move to the extreme diameter of the circle and then reach back to the centre. Simulating an ArduCopter in Mission Planner involves using the Software-in-the-loop (SITL) simulation feature to test and verify the functionality of the ArduPilot autopilot software. To simulate an ArduCopter in Mission Planner, follow these steps:

1. Install Mission Planner.
2. Download and install the latest version of the ArduPilot SITL simulation software.
3. Start Mission Planner and connect to the SITL simulation by selecting the "SITL" option in the "Connect" menu.
4. Plan a flight mission using the mission planner's Flight Planner menu, which allows you to set the desired flight path, altitude, speed, and other parameters.
5. Start the simulation by clicking the "Do Action" button in the "Flight data" menu.
6. Monitor the simulated flight in real-time, including the vehicle's position, altitude, attitude, and other performance metrics.

Output:

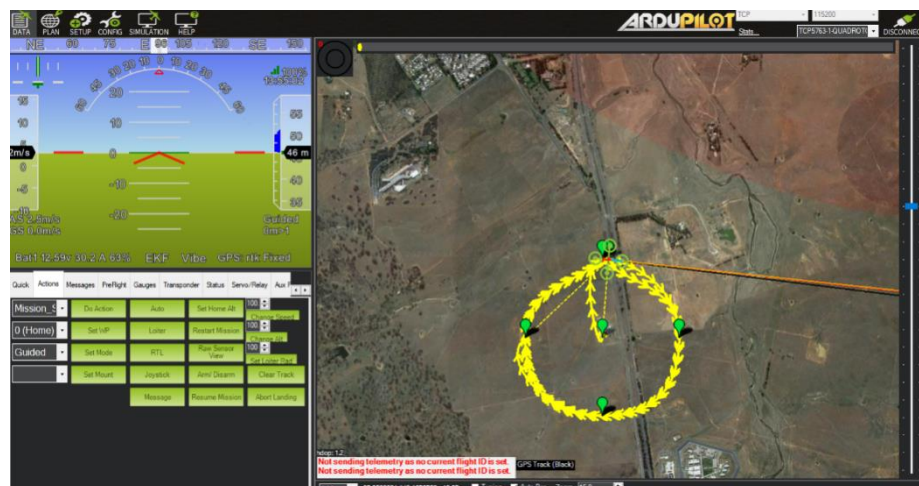


Fig 22: Mission Planner Simulation

### 5.5 Setting up a simulation environment using sim\_vehicle.py

SITL can be run under Linux using a tool named sim\_vehicle.py from a Linux or WSL command line. With sim\_vehicle.py, MAVproxy is automatically started. To run



sim\_vehicle.py, first, Ubuntu or WSL needs to be setup on our system. The following steps are involved in setting up a simulation environment:

1. Write the commands to get git on WSL.
2. Cloning Ardupilot stack in WSL.
3. Install python compiler.
4. Install the prerequisites from ardupilot directory.
5. Setup for other Distributions Using the STM Toolchain.
6. A start up script, 'sim\_vehicle.py' is provided to automatically build the SITL firmware version for the current code branch, load the simulation models, start the simulator, setup environment and vehicle parameters, and start the MAVProxy GCS. Many script start-up parameters can be specified.
7. Select a vehicle/frame type.

Output:



**Fig 23: Simulation Environment**

```

mathurswapnil@LAPTOP-P2N8K6G1 - 
benchmarks                : disabled
unit tests                 : enabled
Scripting                  : enabled
Scripting runtime checks   : enabled
debug build                : disabled
Coverage build             : disabled
SITL 32-bit build         : disabled
checking for program `rsync` : /usr/bin/rsync
Load commands finished successfully (5.087s)
[mode!]: "+", "waf_target": "bin/arducopter", "default_params_filename": "default_params/copter.param", "sitl-port": True)
SIM_VEHICLE: Building
SIM_VEHICLE: /home/mathurswapnil01/ardupilot/modules/waf-waf-light "build" "--target" "bin/arducopter"
waf: Entering directory /home/mathurswapnil01/ardupilot/build/sitl
Embedding file locations.txt:Tools/autotest/locations.txt
Embedding file models/Callisto.json:Tools/autotest/models/Callisto.json
Embedding file models/plane_3d.parm:Tools/autotest/models/plane_3d.parm
Embedding file models/plane.parm:Tools/autotest/models/plane.parm
Embedding file models/xplane_heli.json:Tools/autotest/models/xplane_heli.json
Embedding file models/xplane_plane.json:Tools/autotest/models/xplane_plane.json
waf: Leaving directory /home/mathurswapnil01/ardupilot/build/sitl

BUILD SUMMARY
Build directory: /home/mathurswapnil01/ardupilot/build/sitl
Target          Text (B) Data (B) BSS (B) Total Flash Used (B) Free Flash (B)
-----
bin/arducopter 3880742    169317    209704           4050059 Not Applicable

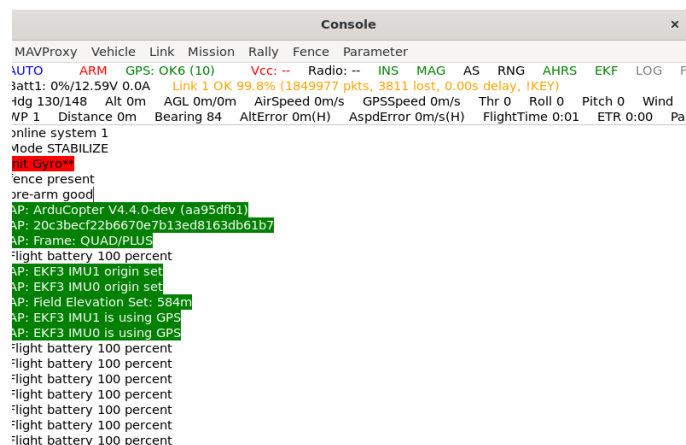
Build commands will be stored in build/sitl/compile_commands.json
[build] finished successfully (11.611s)
SIM_VEHICLE: Using defaults from (ardupilot/Tools/autotest/default_params/copter.param)
SIM_VEHICLE: Run ArduCopter
SIM_VEHICLE: /home/mathurswapnil01/ardupilot/Tools/autotest/run_in_terminal_window.sh "ArduCopter" "/home/mathurswapnil01/ardupilot/build/sitl/bin/arducopter" "-S" "--model" "+" "--speedup" "1" "--slave" "0"
--defaults" "ardupilot/Tools/autotest/default_params/copter.param" "--sim-address-127.0.0.1" "-10"
RTW: Starting ArduCopter : /home/mathurswapnil01/ardupilot/build/sitl/bin/arducopter -S -model + --speedup 1 --slave 0 --defaults ardupilot/Tools/autotest/default_params/copter.param --sim-address-127.0.0.1 10
/bin/sh: 1: route: not found
SIM_VEHICLE: Run mavproxy
SIM_VEHICLE: "mavproxy.py" "--out" "127.0.0.1:14550" "--out" "127.0.0.1:14551" "--master" "tcp://127.0.0.1:5760" "--sitl" "127.0.0.1:5501" "--map" "--console"
stern: cannot load font "10x20"
stern: cannot load font "misc-fixed-medium-r-normal--20-200-75-75-c-100-iso10646-1"
connect tcp://127.0.0.1:5760 source_system=235
loaded module console
loaded module map
Log Directory:
telemetry log: mav.tlog
Waiting for heartbeat from tcp://127.0.0.1:5760
MAV0 Detected vehicle 1:1 on link 0
STABILIZE Received 1333 parameters (ftp)
Saved 1333 parameters to mav.parm
GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: RTLL: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED: AUTO: GUIDED:

```

**Fig 24: WSL command window**



**Fig 25: ArduCopter SITL pop-up**



**Fig 26: MAVProxy**

## 6.1 GIT

## 6.2 Branching in GIT

You can create a new branch, make changes, merge the branch back into the main branch, and even delete the branch once it's no longer needed. By using branches, multiple developers can work on the same project simultaneously, and Git helps manage and merge the changes.



## 6.3 Submodules in GIT

Submodules can be used to manage dependencies, incorporate external code into your project, or organize your project into smaller, reusable components. However, managing submodules can be a bit more complex than simply using a normal Git repository, and requires a deeper

understanding of Git's internals. If you need to maintain a strict version management over your external dependencies, it can make sense to use git submodules.

#### **6.4 “Push” command in GIT**

“Push” in Git is a command used to upload your local changes to a remote repository. When you make changes to your code and commit them locally, you can use the git push command to send those changes to a remote repository, such as GitHub or GitLab, where they can be shared with others and backed up. It's important to note that you can only push changes to a remote repository if you have “write” access to it. Also, before pushing, it's always a good idea to pull any changes from the remote repository to ensure your local repository is up-to-date.

#### **6.5 “Pull” request in GIT**

“Pull” in Git is a command used to download changes from a remote repository and integrate them into your local repository. When multiple people are working on the same project, you may need to regularly download their changes so that your local repository is up-to-date.

Pull requests are a mechanism for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their Bitbucket account. This lets everybody involved know that they need to review the code and merge it into the master branch.

When you file a pull request, you're requesting that another developer pulls a branch from your repository into their repository. This means that you need to provide 4 pieces of information to file a pull request:

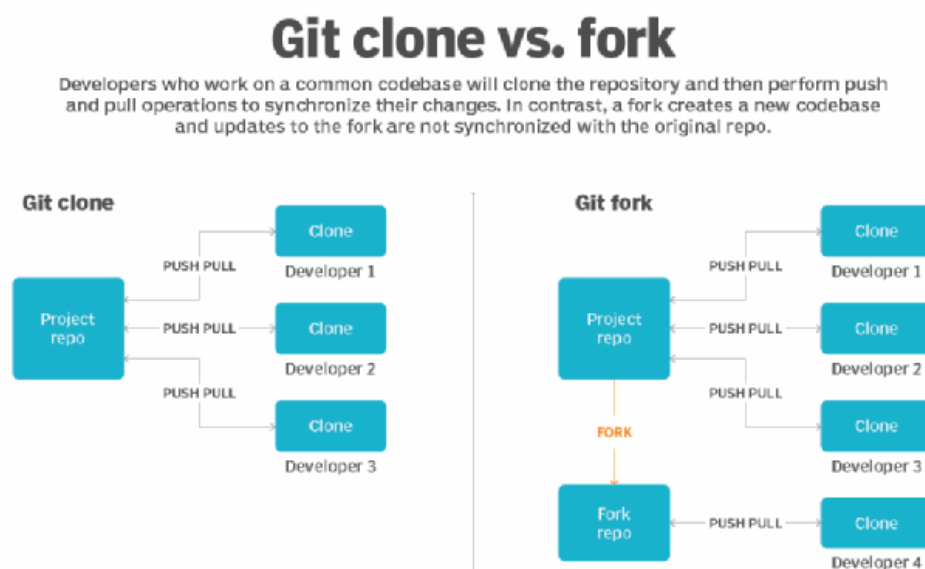
- ❖ the source repository
- ❖ The source branch
- ❖ The destination repository
- ❖ The destination branch

#### **6.6 Forking in GIT**

Forking in Git refers to the process of creating a separate copy of a Git repository. This allows you to make changes to the code, create new features, or experiment with different ideas without affecting the original repository.

Forking in Git is different from forking in platforms like GitHub, where forking refers to creating a copy of a repository on a different user's account. In Git, forking simply means creating a local copy of the repository. When a git fork occurs, previous contributors will not be able to commit code to the new repository without the owner giving them access to the forked repo, either by providing developers the publicly accessible Git URL, or by providing explicit access through user permission in tools like GitHub or GitLab.

If you have a Git repository on your personal computer, you can create a fork simply by copying the Git repo to a new folder and then removing any remote references in the Git config file. That will create an isolated and independent Git fork that will no longer synchronize with the original codebase.



**Figure 28: Git Cloning and Forking**

## Chapter 7: Robot Operating System (ROS)

Robot Operating System (ROS) is an open-source, meta-operating system for robots. It provides a set of tools and libraries for developing software for robots, as well as a runtime environment for executing that software on a real robot. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS was developed to enable the development of robotic applications, and it has been widely adopted by the robotics community, particularly in the field of unmanned aerial vehicles (UAVs) and drones.

### 7.1 Components of ROS:

1. **ROS Master:** A central node that manages the overall ROS system and keeps track of the topics, services, and parameters that are available in the system.
2. **Nodes:** Software components that perform specific tasks within the system. They communicate with each other through topics, services, or parameters, and can publish or subscribe to messages.
3. **Topics:** A publish-subscribe mechanism that enables nodes to exchange messages with each other.
4. **Services:** A request-response mechanism that allows nodes to request specific actions from other nodes.
5. **Parameters:** Variables that can be used to store and share data between nodes.
6. **Messages:** Data structures that are used to communicate between nodes.
7. **Bags:** Bags are a format for saving and playing back ROS message data.

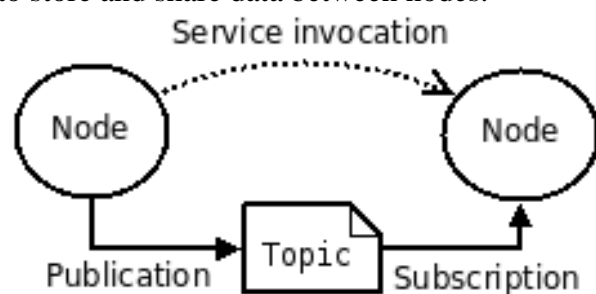


Figure 29: Publisher and Subscriber

### 7.2 Features of ROS:

- ❖ **Thin:** ROS is designed to be as thin as possible so that code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks. ROS has already been integrated with OpenRAVE, Orocos, and Player.

- ❖ ROS-agnostic libraries: the preferred development model is to write ROS-agnostic libraries with clean functional interfaces.
- ❖ Language independence: the ROS framework is easy to implement in any modern programming language. It has been implemented in Python, C++, and Lisp, and we have experimental libraries in Java and Lua.
- ❖ Easy testing: ROS has a built-in unit/integration test framework called *rostopic* that makes it easy to bring up and tear down test fixtures.
- ❖ Scaling: ROS is appropriate for large runtime systems and for large development processes.

### 7.3 Client Libraries:

- i. `roscpp` : `roscpp` is a C++ client library for ROS. It is the most widely used ROS client library and is designed to be the high-performance library for ROS.
- ii. `rospy`: `rospy` is the pure Python client library for ROS and is designed to provide the advantages of an object-oriented scripting language to ROS.

### 7.4 Usage of ROS in UAVs and Drones:

ROS has been widely adopted in the development of UAVs and drones due to its flexibility and ease of use. It enables developers to build and test their applications on a simulated robot in a virtual environment before deploying them on a real robot. ROS also provides a large number of pre-built software components that can be reused, saving time and effort in the development process.

In UAVs and drones, ROS is used to control the flight and navigation of the vehicle, as well as to process sensor data and perform data analysis. It is also used to develop applications for mission planning, obstacle avoidance, and autonomous navigation.

In conclusion, ROS is a powerful and flexible platform for developing robotic applications, and its widespread adoption in the field of UAVs and drones highlights its potential in this area.



## 7.5 Installation & Enviromental Setup :

I have used Ubuntu 20.04.6 as Base Operating System. We can use Ubuntu 18.04.6, Debian Buster for Installation.

There are Two Types of ROS distributions, Some are older releases with long term support, making them more stable, while others are newer with shorter support life times.

I am going to install ROS 1 distribution which is ROS with long term support. There are Two Types of ROS 1 Distribution i.e.



ROS Melodic Morenia



ROS Noetic Ninjemys

**Fig 30: ROS Distributions**

Here I will install ROS Noetic Ninjemys on Ubuntu 20.04.6. I have to follow the steps given in the ROS wiki

1. I have configured Ubuntu repositories to allow "restricted," "universe," and "multiverse."

2. I have setup sources.list in Ubuntu

```
➤ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

I can use Mirrors to setup the sources.list and have better access to nearer Mirror for latency,etc

3. I have setup keys from Github

```
sudo apt install curl # if you haven't already installed curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

4. Installation

I have made sure Debian package index is up-to-date:

```
sudo apt update
```

I have to pick which parts of ROS I would like to install.

It is Recommended to install full desktop version of ROS

```
➤ sudo apt install ros-noetic-desktop-full # Everything in Desktop plus 2D/3D simulators and 2D/3D perception packages
```

I can just do Desktop Install:



- `sudo apt install ros-noetic-desktop` # Everything in ROS-Base plus tools like rqt and rviz

I can install ROS-Base

- `sudo apt install ros-noetic-ros-base` # ROS packaging, build, and communication libraries. No GUI tools.

I can always install a specific package directly.

- `sudo apt install ros-noetic-PACKAGE`  
e.g. `sudo apt install ros-noetic-slam-gmapping`

To find available packages, see ROS Index or use:

- `apt search ros-noetic`

## 5. Environment setup

I have source this script in every bash terminal you use ROS in.

```
source /opt/ros/noetic/setup.bash
```

It can be convenient to automatically source this script every time a new shell is launched.

Bash

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

zsh

```
echo "source /opt/ros/noetic/setup.zsh" >> ~/.zshrc
```

```
source ~/.zshrc
```

## 6. Dependencies for building packages

I have to install dependencies to run the core ROS packages. To create and manage my own ROS workspace, there are various tools and requirements that are distributed separately. For example, `rosinstall` is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

- `sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential`

Initialize `rosdep`

I will need to initialize `rosdep`. It enables to install system dependencies for source you want to compile and is required to run some core components in ROS. If `rosdep` is not installed. I have to give the command

- `sudo apt install python3-rosdep`

I have to initialize `rosdep` with following commands

- `sudo rosdep init`

➤ `roscd update`

## 7.6 Simple Publisher and Subscriber

I'm going to explain Simple Publisher and Subscriber and Example of similar to it

I have to change directories to `beginner_tutorials` package directory

➤ `roscd beginner_tutorials`

I have to create a `src` directory in the `beginner_tutorials` package directory

➤ `mkdir -p src`

I have to create `talker.cpp` file within the `beginner_tutorials` package and paste the code from [https://raw.githubusercontent.com/ros/ros\\_tutorials/kinetic-devel/roscpp\\_tutorials/talker/talker.cpp](https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/talker/talker.cpp)

This code is an example of a publisher node in ROS that sends messages to a topic named "chatter". The node is initialized, and the "chatter" topic is registered by calling the `advertise()` function. Then, a loop is started to send messages to the topic. Each message contains a string with a count variable. The loop also includes a call to `ros::spinOnce()` to process incoming messages and a sleep function to control the rate at which messages are sent. Finally, the program returns 0 to indicate successful execution.

I have to create `listener.cpp` file within the `beginner_tutorials` package and paste the code from

[https://raw.githubusercontent.com/ros/ros\\_tutorials/kinetic-devel/roscpp\\_tutorials/listener/listener.cpp](https://raw.githubusercontent.com/ros/ros_tutorials/kinetic-devel/roscpp_tutorials/listener/listener.cpp)

This code subscribes to a ROS topic called "chatter" and receives messages published on that topic. When a message is received, it calls the function "chatterCallback" which prints the message to the console. The program uses the ROS framework and requires the ROS system to be initialized before running. It runs until it is interrupted by a Ctrl-C command or until the node is shut down by the master.

After Writing the code, We have to build the nodes

You have to add these Lines in `CMakeLists.txt`

```
add_executable(talker src/talker.cpp)
```

```
target_link_libraries(talker ${catkin_LIBRARIES})
```

```
add_dependencies(talker beginner_tutorials_generate_messages_cpp)
```

```
add_executable(listener src/listener.cpp)
```

```
target_link_libraries(listener ${catkin_LIBRARIES})
```

```
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

Now run `catkin make`:

# In your catkin workspace

```
$ cd ~/catkin_ws && $ catkin_make
```

We have to Run the Publisher and Subscriber Node

```
roscore # Main Command
```

```
roslaunch beginner_tutorials talker # run Publisher
```

```
roslaunch beginner_tutorials listener # run Subscriber
```

I was asked to write ROS code which publish simple string/number typed by user & subscriber display the message

I have to change directories to beginner\_tutorials package directory

➤ `roscd beginner_tutorials`

I have to create a src directory in the beginner\_tutorials package directory

➤ `mkdir -p src`

I have created custom\_talker.cpp file and written code

The code creates a ROS publisher node that sends messages of type "std\_msgs::String" to a specified topic. It takes user input in the form of a number, assigns it to the "data" field of the message, and publishes it. The while loop keeps the node running until the user terminates it.

I have created custom\_listener.cpp file and written code

The code sets up a ROS node as a subscriber to a topic called `~/topic_name`. Whenever a message is received on the topic, the `callback` function is called, which simply prints the message's contents to the console. The program uses the `ros::spin()` function to keep the node running and waiting for messages to arrive.

We have to build the Nodes and add these lines in CMakeLists.txt

```
add_executable(custom_talker src/custom_talker.cpp)
```

```
target_link_libraries(talker ${catkin_LIBRARIES})
```

```
add_dependencies(custom_talker beginner_tutorials_generate_messages_cpp)
```

```
add_executable(custom_listener src/custom_listener.cpp)
```

```
target_link_libraries(listener ${catkin_LIBRARIES})
```

```
add_dependencies(custom_listener beginner_tutorials_generate_messages_cpp)
```

Now run catkin\_make:

# In your catkin workspace

```
$ cd ~/catkin_ws && $ catkin_make
```

We have to Run the Publisher and Subscriber Node

roscore # Main Command

```
roslaunch beginner_tutorials custom_talker # run Publisher
```

```
roslaunch beginner_tutorials custom_listener # run Subscriber
```

## Chapter 8: Linear Algebra using various Programming Languages

### 8.1 Linear Algebra Using MATLAB

- **Problem Statement:** There are two consecutive rotations at angles  $a$  and  $b$ . Show that the two consecutive rotations are equivalent to one single rotation at angle  $(a+b)$ .

MATLAB Program: Let's take 3 vectors, "u", "v" and "w"

```
u = [1, 1, 3]'
```

```
v = [1, 1, 6]'
```

```
w = [1, 1, 9]'
```

the angles between these three vectors are  $a$ ,  $b$ ,  $c$  respectively. Where  $c$  is the total angle between  $u$  and  $w$  which should be equal to  $a+b$ .

```
a = atan2d(norm(cross(u,v)), dot(u,v))
```

```
b = atan2d(norm(cross(v,w)), dot(v,w))
```

```
c = atan2d(norm(cross(u,w)), dot(u,w))
```

Implementing "areEssentiallyEqual" command.

```
areEssentiallyEqual = ismembertol(a+b,c)
```

If the condition is satisfied, it prints "areEssentiallyEqual=1"

```
ThetaInDegrees1 =
    10.0250
CosTheta2 =
    0.7714
ThetaInDegrees2 =
    39.5212
areEssentiallyEqual =
    logical
     1
Two consecutive rotation is equivalent to one single rotation by angle
```

```
>> tutorial1
u =
     1
     3
     1
v =
     2
     2
     2
w =
     3
     2
     3
CosTheta =
    0.8704
ThetaInDegrees =
    29.4962
CosTheta1 =
    0.9847
```

- **Problem Statement:** For what value of  $k$ , the vectors  $[2, 4, 1, k]^T$  and  $[k, 2, 4, 5]^T$  are orthogonal?

MATLAB Program: First, define “k”.

```
syms k
```

Let the 2 vectors be “u” and “v”.

```
u = [2, 4, 1, k];
```

```
v = [k, 2, 4, 5];
```

And their transpose is represented by “a” and “b” respectively.

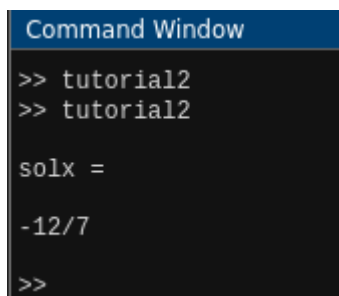
```
a = u.'
```

```
b = v.'
```

For vectors “a” and “b” to be orthogonal, their dot product should be equal to 0, Forming the equation:

```
eqn = dot(a,b) == 0;
```

```
solx = solve(eqn, k)
```

A screenshot of the MATLAB Command Window. The title bar is blue and says "Command Window". The background is black with white text. The text shows the user entering "tutorial2" twice, followed by the output "solx =" and the value "-12/7". The prompt ">>" is visible at the bottom.

```
Command Window
>> tutorial2
>> tutorial2

solx =
-12/7
>>
```

Hence,  $k = -12/7$ .

➤ **Problem Statement:** For what value of k, the vectors  $[8, 4, 1, k]^T$  and  $[2, -4, k, k]^T$  are orthogonal?

**MATLAB Program:** First, define “k”.

```
syms k
```

Let the 2 vectors be “u” and “v”.

```
u = [8, 4, 1, k];
```

```
v = [2, -4, k, k];
```

And their transpose is represented by “a” and “b” respectively.

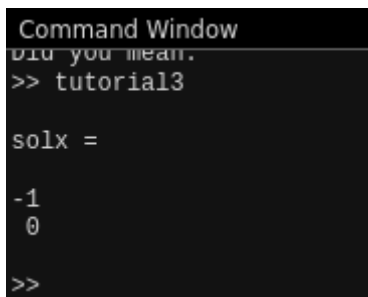
```
a = u.'
```

```
b = v.'
```

For vectors “a” and “b” to be orthogonal, their dot product should be equal to 0, Forming the equation:

```
eqn = dot(a,b) == 0;
solx = solve(eqn, k)
```

Hence, k has 2 solutions. k= -1 and k=0



```
Command Window
did you mean:
>> tutorial3

solx =

-1
0
>>
```

➤ **Problem Statement:** Write the vector  $v = [1, -2, 5]$  as a linear combination of the vectors

$a = [1, 1, 1]$ ,  $b = [1, 2, 3]$ ,  $c = [2, -1, 1]$ .

**MATLAB Program:** Defining variables:

```
syms x y z
```

Write all the vectors.

```
v = [1, -2, 5];
a = [1, 1, 1];
b = [1, 2, 3];
c = [2, -1, 1];
```

Run the following command:

```
eqn = a*x + b*y + c*z == v
[solx, soly, solz] = solve(eqn, x, y, z)
```

**Command window output:**

```
eqn = [ x + y + 2*z == 1, x + 2*y - z == -2, x + 3*y + z == 5]
solx = -6
soly = 3
solz = 2
```

**Output Equation:**

$$[1, -2, 5] = -6[1, 1, 1] + 3[1, 2, 3] + 2[2, -1, 1]$$

```

Command Window
>> tutorial4

solx =

-6

soly =

3

solz =

2

>>

```

- **Problem Statement:** Super comp Ltd produces two computer models PC1086 and PC1186. The matrix A shows the cost per computer (in thousands of dollars) and B the production figures for the year 2010 (in multiples of 10,000 units.) Find a matrix C that shows the shareholders the cost per quarter (in millions of dollars) for raw material, labour, and miscellaneous.

PC1086    PC1186		Quarter	
		1    2    3    4	
<b>A</b> =	$\begin{bmatrix} 1.2 & 1.6 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$	Raw Components	<b>B</b> = $\begin{bmatrix} 3 & 8 & 6 & 9 \\ 6 & 2 & 4 & 3 \end{bmatrix}$
	Labor	PC1086	
	Miscellaneous	PC1186	

**MATLAB Program:**

```

A = [1.2 1.6; 0.3 0.4; 0.5 0.6];
B = [3 8 6 9; 6 2 4 3];
C = A*B;
rowNames = {'Raw Components', 'Labour', 'Miscellaneous'};
colNames = {'Q1', 'Q2', 'Q3', 'Q4'};
sTable2 =
array2table(C, 'RowNames', rowNames, 'VariableNames', colName
s)

```

**Command Window Output:**



Command Window

```
>> tutorial5
```

sTable2 =

3×4 [table](#)

	Q1	Q2	Q3	Q4
Raw Components	13.2	12.8	13.6	15.6
Labor	3.3	3.2	3.4	3.9
Miscellaneous	5.1	5.2	5.4	6.3

- **Problem Statement:** Suppose that in a weight-watching program, a person of 185 lb burns 350cal/hr in walking (3mph), 500 in bicycling (13mph), and 950 in jogging (5.5mph). Bill, weighing 185/b, plans to exercise according to the matrix shown. Verify the calculations. (W = Walking, B = Bicycling, J = Jogging).

Given Matrices:

	W	B	J
Mon	1	0	0.5
Wed	1	1	0.5
Fri	1.5	0	0.5
Sat	2	1.5	1

350
500
950

=

825	Mon
1325	Wed
1000	Fri
2400	Sat

**MATLAB Program:** A and B are defined as the exercise per hour and total calories burned per day respectively. C is their product which is to be verified. D is for verification.

```
A = [1 0 0.5; 1 1 0.5; 1.5 0 0.5; 2 1.5 1]
B = [350; 500; 950]
C = [825; 1325; 1000; 2400]
D = A*B
if (D == C) % the statement is true
    fprintf ('answer verified')
end
```

```

Command Window
>> tutorial6

C =

    825
   1325
   1000
   2400

Ans =

    825
   1325
   1000
   2400

Calculations are verified

```

➤ **Problem Statement:** To find the inverse of a 3x3 matrix.

#### MATLAB Program 1:

```

X = [9 6 12; 14 7 9; 24 8 16]
Y = inv(X)
%I is the inverse matrix to verify the answer%
I = Y*X

```

#### Output:

```

>> tutorial7

Y =

   -0.0030    0.0010    0.0018
   -0.0076    0.0479   -0.0012
    0.2131   -0.0678    0.0003

Z =

    1.0000         0    0.0000
         0    1.0000   -0.0000
         0   -0.0000    1.0000

I =

     1     0     0
     0     1     0
     0     0     1

```

## 8.2 Linear Algebra Using Python

- **Problem Statement:** There are two consecutive rotations at angles  $a$  and  $b$ . Show that the two consecutive rotations are equivalent to one single rotation at angle  $(a + b)$ .

### Output:

Angle between x and y: 0.20903329903451998 radians

Angle between y and z: 0.07561698626360407 radians

Angle between x and z: 0.28465028529812614 radians

Angle between x and y + angle between y and z: 0.28465028529812403 radians

```
kp@kp:~/embedded_paras$ python3 problem1.py
Angle between x and y: 29.49620849656643 degrees
Angle between y and z: 10.024987862075765 degrees
Angle between x and z: 39.521196358642186 degrees
Angle between x and y + angle between y and z: 39.5211963586422 degrees
```

- **Problem Statement:** For what value of  $k$ , the vectors  $[2, 4, 1, k]^T$  and  $[k, 2, 4, 5]^T$  are orthogonal?

### Output:

```
kp@kp:~/embedded_paras$ python3 problem2.py
aT = Transposed Array:
[[2]
 [4]
 [1]
 [x]]
bT = Transposed Array:
[[x]
 [2]
 [4]
 [5]]
the value of k is:
[-12/7]
```

the value of  $k$  is:

$[-12/7]$

- **Problem Statement:** For what value of  $k$ , the vectors  $[8, 4, 1, k]^T$  and  $[2, -4, k, k]^T$  are orthogonal?

```

kp@kp:~/embedded_paras$ python3 problem3.py
aT = Transposed Array:
[[8]
 [4]
 [1]
 [x]]
bT = Transposed Array:
[[2]
 [-4]
 [x]
 [x]]
the value of k is:
[-1, 0]

```

**Output:**

the value of k is:

[-1, 0]

- **Problem Statement:** Write the vector  $v = [1, -2, 5]$  as a linear combination of the vectors

$a = [1, 1, 1]$ ,  $b = [1, 2, 3]$ ,  $c = [2, -1, 1]$ .

```

kp@kp:~/embedded_paras$ python3 problem4.py
x = -6.0, y = 3.0, z = 2.0

```

**Output:**

$x = -6.0$ ,  $y = 3.0$ ,  $z = 2.0$

- **Problem Statement:** Super comp Ltd produces two computer models PC1086 and PC1186. The matrix A shows the cost per computer (in thousands of dollars) and B the production figures for the year 2010 (in multiples of 10,000 units.) Find a matrix C that shows the shareholders the cost per quarter (in millions of dollars) for raw material, labour, and miscellaneous.

<table border="0" style="width: 100%;"> <tr> <td style="width: 15%;"></td> <td style="width: 15%; text-align: center;">PC1086</td> <td style="width: 15%; text-align: center;">PC1186</td> <td style="width: 15%;"></td> </tr> <tr> <td style="vertical-align: middle;"><math>\mathbf{A} =</math></td> <td style="vertical-align: middle;"> <math display="block">\begin{bmatrix} 1.2 &amp; 1.6 \\ 0.3 &amp; 0.4 \\ 0.5 &amp; 0.6 \end{bmatrix}</math> </td> <td style="vertical-align: middle;"></td> <td style="vertical-align: middle;">           Raw Components            Labor            Miscellaneous         </td> </tr> </table>		PC1086	PC1186		$\mathbf{A} =$	$\begin{bmatrix} 1.2 & 1.6 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$		Raw Components Labor Miscellaneous	<table border="0" style="width: 100%;"> <tr> <td style="width: 15%;"></td> <td style="width: 15%; text-align: center;">Quarter</td> <td style="width: 15%;"></td> </tr> <tr> <td></td> <td style="text-align: center;">1   2   3   4</td> <td></td> </tr> <tr> <td style="vertical-align: middle;"><math>\mathbf{B} =</math></td> <td style="vertical-align: middle;"> <math display="block">\begin{bmatrix} 3 &amp; 8 &amp; 6 &amp; 9 \\ 6 &amp; 2 &amp; 4 &amp; 3 \end{bmatrix}</math> </td> <td style="vertical-align: middle;">           PC1086            PC1186         </td> </tr> </table>		Quarter			1   2   3   4		$\mathbf{B} =$	$\begin{bmatrix} 3 & 8 & 6 & 9 \\ 6 & 2 & 4 & 3 \end{bmatrix}$	PC1086 PC1186
	PC1086	PC1186																
$\mathbf{A} =$	$\begin{bmatrix} 1.2 & 1.6 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$		Raw Components Labor Miscellaneous															
	Quarter																	
	1   2   3   4																	
$\mathbf{B} =$	$\begin{bmatrix} 3 & 8 & 6 & 9 \\ 6 & 2 & 4 & 3 \end{bmatrix}$	PC1086 PC1186																

Output:

```
kp@kp:~/embedded_paras$ python3 problem5.py
[13.2 12.8 13.6 15.6]
[3.3 3.2 3.4 3.9]
[5.1 5.2 5.4 6.3]
      Q1      Q2      Q3      Q4
Raw Components 13.2 12.8 13.6 15.6
Labour          3.3  3.2  3.4  3.9
Misc            5.1  5.2  5.4  6.3
```

- **Problem Statement:** Suppose that in a weight-watching program, a person of 185 lb burns 350cal/hr in walking (3mph), 500 in bicycling (13mph), and 950 in jogging (5.5mph). Bill, weighing 185/b, plans to exercise according to the matrix shown. Verify the calculations. (W = Walking, B = Bicycling, J = Jogging).

Given Matrices:

	W	B	J
Mon	1	0	0.5
Wed	1	1	0.5
Fri	1.5	0	0.5
Sat	2	1.5	1

350
500
950

=

825	Mon
1325	Wed
1000	Fri
2400	Sat

Output

```
kp@kp:~/embedded_paras$ python3 problem6.py
[825.]
[1325.]
[745.]
[2400.]
The two matrices, result and C, are equal
```

- **Problem Statement:** To find the inverse of a 3x3 matrix.

Given Matrix:  $A = \begin{pmatrix} 4.0 & 7 & 5 \\ 15 & 22 & 1 \\ 567 & 0 & 8 \end{pmatrix}$

**Output:**

```
kp@kp:~/embedded_paras$ python3 problem7.py
Inverse of Matrix:
[[-0.00300665  0.00095666  0.00175957]
 [-0.0076362  0.04788424 -0.00121291]
 [ 0.21309599 -0.06780327  0.00029041]]
Identity Matrix:
[[ 1.00000000e+00 -5.55111512e-17 -4.33680869e-19]
 [ 0.00000000e+00  1.00000000e+00 -1.62630326e-18]
 [ 0.00000000e+00 -1.11022302e-16  1.00000000e+00]]
```

### 8.3 Linear Algebra Using C++

- **Problem Statement:** There are two consecutive rotations at angles  $\alpha$  and  $\beta$ . Show that the two consecutive rotations are equivalent to one single rotation at angle  $(\alpha + \beta)$ .

**Output:**

```
kp@kp:~/embedded_paras$ ./problem1
Cosine of an angle is 1st vector and 2nd vector:29.5112
Cosine of an angle is 2nd vector and 3rd vector:10.0301
Cosine of an angle is 1st vector and 3rd vector:39.5412
Two consecutive rotation is equivalent to one single rotation by angle ( $\alpha + \beta$ )
```

- **Problem Statement:** Write the vector  $v = [1, -2, 5]$  as a linear combination of the vectors

$a = [1, 1, 1]$ ,  $b = [1, 2, 3]$ ,  $c = [2, -1, 1]$ .

**Solution:**

$x=-6$ ,  $y=3$ ,  $z=2$

**Output:**

```
kp@kp:~/embedded_paras$ ./problem4
The vector v = [1, -2, 5] can be written as a linear combination of the vectors
u1 = [1, 1, 1], u2 = [1, 2, 3], and u3 = [2, -1, 1] as:
v = -6*u1 + 3*u2 + 2*u3
```

- **Problem Statement:** Super comp Ltd produces two computer models PC1086 and PC1186. The matrix A shows the cost per computer (in thousands of dollars) and B the production figures for the year 2010 (in multiples of 10,000 units.) Find a matrix C that shows the shareholders the cost per quarter (in millions of dollars) for raw material, labour, and miscellaneous.

<div style="display: flex; justify-content: space-around; margin-bottom: 10px;"> <span>PC1086</span> <span>PC1186</span> </div> $  \mathbf{A} = \begin{bmatrix} 1.2 & 1.6 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}  $ <div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">             Raw Components              Labor              Miscellaneous           </div> </div>	<div style="display: flex; justify-content: space-around; margin-bottom: 10px;"> <span>Quarter</span> </div> <div style="display: flex; align-items: center;"> <math display="block">  \mathbf{B} = \begin{bmatrix} 3 &amp; 8 &amp; 6 &amp; 9 \\ 6 &amp; 2 &amp; 4 &amp; 3 \end{bmatrix}  </math> <div style="margin-left: 10px;">             PC1086              PC1186           </div> </div>
---	---

**Output:**

```

kp@kp:~/embedded_paras$ ./problem5
Result Matrix:

```

	Q1	Q2	Q3	Q4
Raw Components	13.20	12.80	13.60	15.60
Labour	3.30	3.20	3.40	3.90
Misc.	4.50	5.00	5.00	6.00

- **Problem Statement:** Suppose that in a weight-watching program, a person of 185 lb burns 350cal/hr in walking (3mph), 500 in bicycling (13mph), and 950 in jogging (5.5mph). Bill, weighing 185/b, plans to exercise according to the matrix shown. Verify the calculations. (W = Walking, B = Bicycling, J = Jogging).

Given Matrices:

	W	B	J
Mon	1	0	0.5
Wed	1	1	0.5
Fri	1.5	0	0.5
Sat	2	1.5	1

350
500
950

=

825	Mon
1325	Wed
1000	Fri
2400	Sat

Output:

```

kp@kp:~/embedded_paras$ ./problem6
Result Matrix:
R1 C1: 825.00
R2 C1: 1325.00
R3 C1: 1000.00
R4 C1: 2400.00
Matrices match.

```

- **Problem Statement:** To find the inverse of a 3x3 matrix.

Given Matrix:  $A = \begin{pmatrix} 4.0 & 7 & 5 \\ 15 & 22 & 1 \\ 567 & 0 & 8 \end{pmatrix}$

**Output:**

Enter elements of matrix row wise:

9 6 12

14 7 9

24 8 16



```
kp@kp:~/embedded_paras$ ./problem7
```

```
Given matrix is:
```

```
4      7      5
15     22     1
567    0      8
```

```
Inverse of matrix is:
```

```
-0.0030 0.0010 0.0018
-0.0076 0.0479 -0.0012
0.2131 -0.0678 0.0003
```

## Chapter 9: ArduPilot SITL using with RealFlight

This Section explains how to use SITL (**sim\_vehicle.py**) to execute survey missions in RealFlight 9.5 simulator. Linux PC is used to run the SITL and Windows 10 PC is used to run RealFlight. **Ubuntu version 20.04 is mandatory** for this simulation. RealFlight 9.5 is a RC flight simulator software designed for both novice and experienced pilots. It offers an immersive experience by simulating real-world physics, allowing users to train and improve their flying skills without the risks and expenses associated with actual RC flying. In order to run RealFlight, a controller must be connected with the computer.

### 9.1 Building the Simulation Environment

First **sim\_vehicle.py** must be configured on the Linux PC according to the terminal commands:

1. Get git:

```
sudo apt-get update  
  
sudo apt-get install git  
  
sudo apt-get install gitk git-gui
```

2. Cloning the git repository

Developers should clone the main ArduPilot repository (if they simply want to download and compile the latest code) or their own fork (if they want to make changes to the source code and potentially submit changes back).

Clone the repository using the following command:

- `git clone https://github.com/ParasIntern-SwapnilMathur/ardupilot`
- `cd ardupilot`
- `git submodule update --init --recursive`

3. Install some required packages:

```
Tools/environment_install/install-prereqs-ubuntu.sh -y
```

Reload the path using: `. ~/.profile`

4. Now to run copter SITL, in the terminal, write:

```
cd ArduCopter
```

```
sim_vehicle.py -v ArduCopter --console --map
```

Now, your SITL should run on the linux PC with the arducopter console, MAVlink console and the map which shows the icon of a drone.

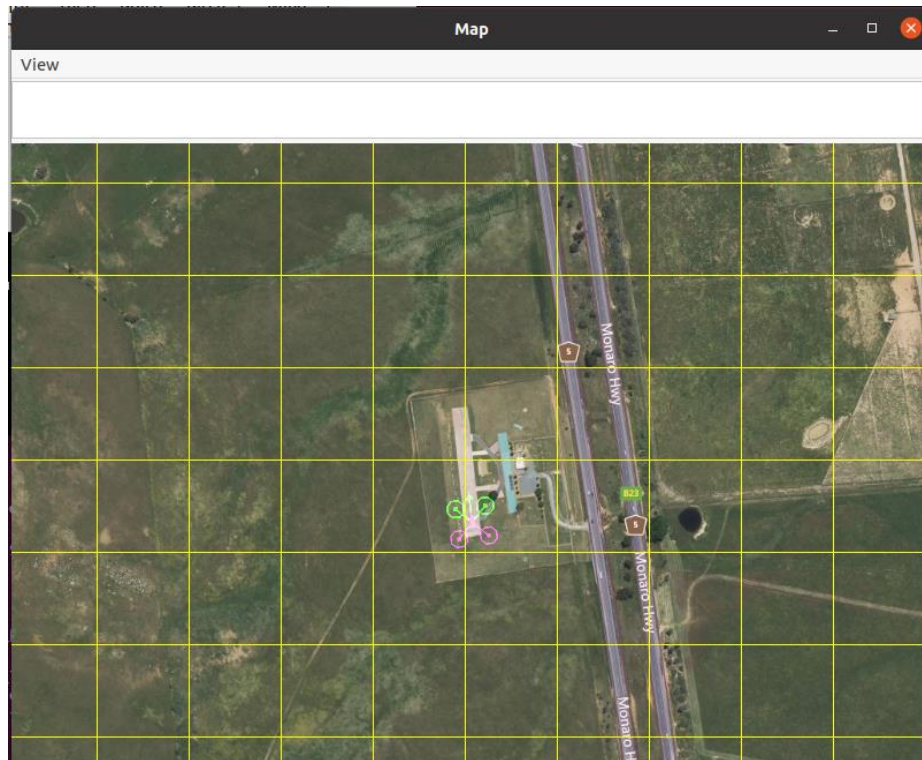
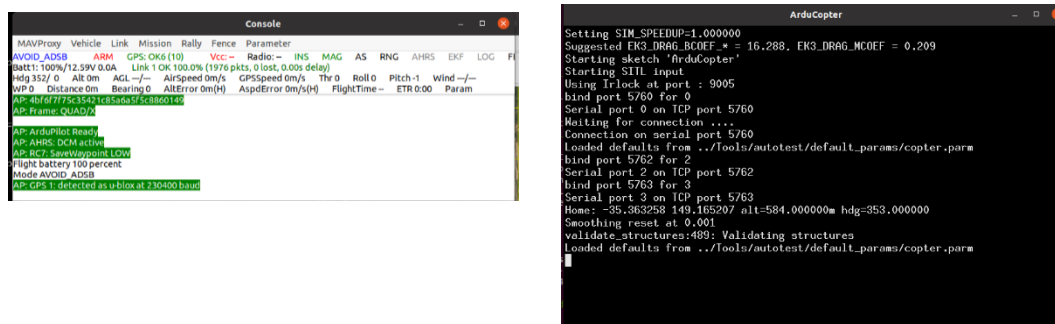


Fig 31. Map with Quadcopter icon



(a)

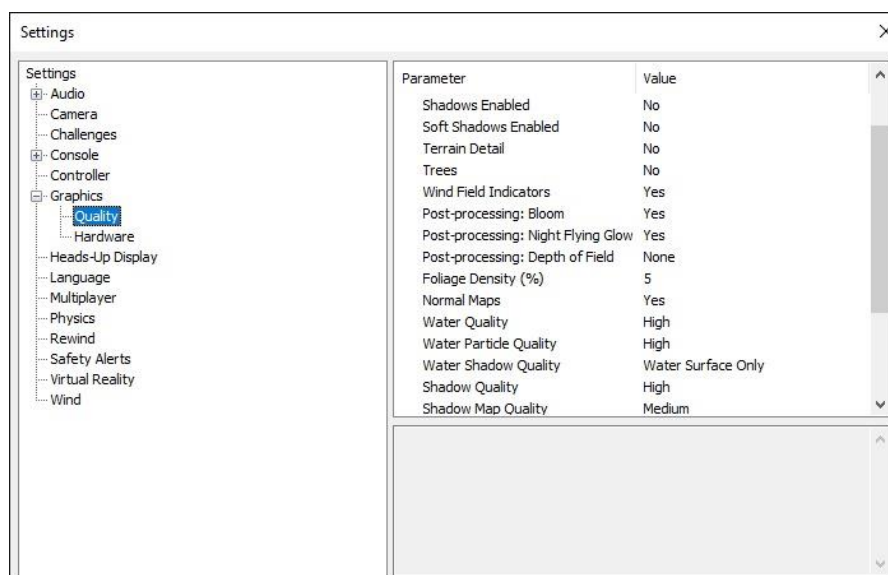
(b)

Fig 32: (a) MAVproxy Console and (b) ArduCopter console

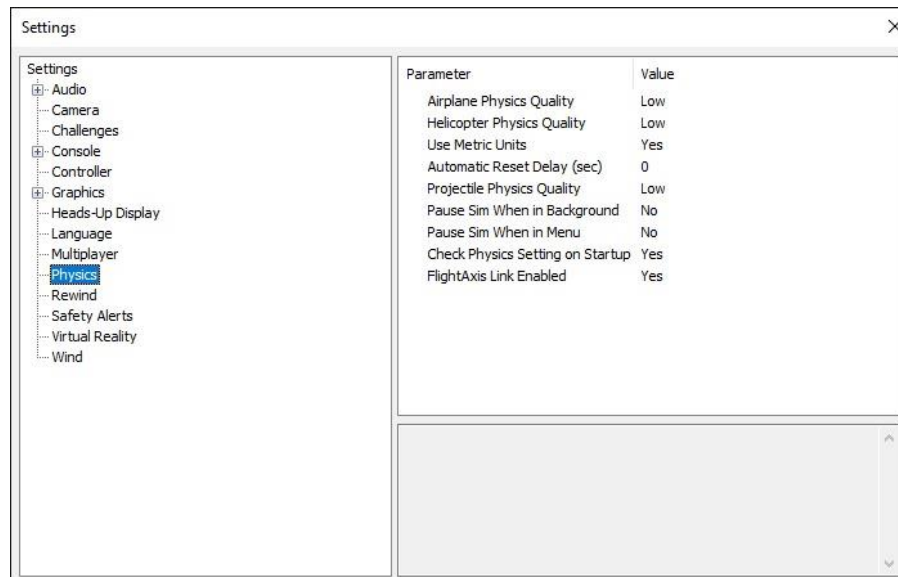
## 9.2 Configuring RealFlight

To run realflight with SITL, the simulator needs to be configured as per the following:

- Go to “Simulation.”
- Click on “Setting.” > “Physics” > Enable flightaxis.
- Under “Physics” settings, change the option for “Pause Sim When in Background” to No, and “Automatic Reset Delay(sec)” to 2.0, and be sure “Enable flightaxis” is “Yes.”
- Now, go to “Graphics” > “Quality” > set everything to “No” or “Low.”
- Now, go to “Graphics” > “Hardware” > set “Resolution” to “800 x 600.”
- Restart RealFlight.



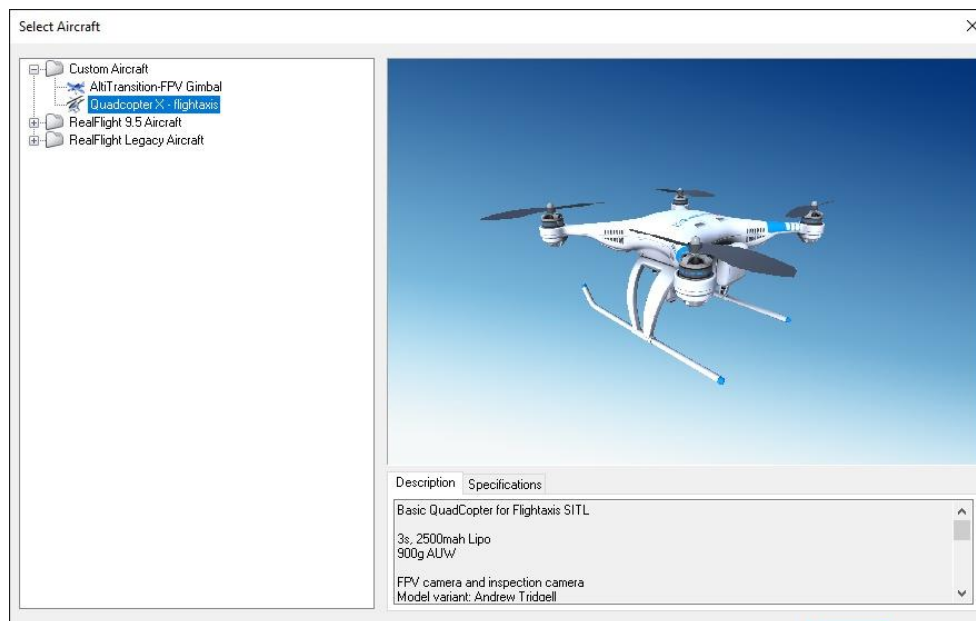
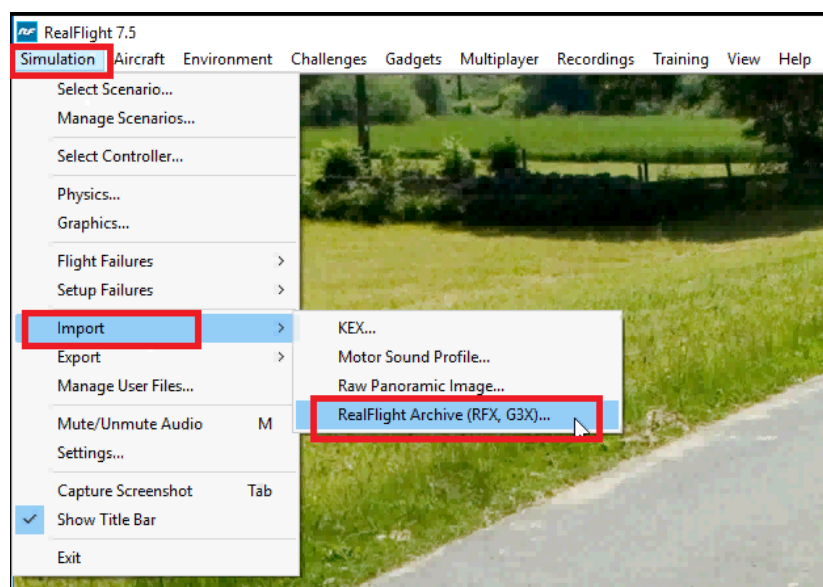
**Fig 33: Quality settings of RealFlight**



**Fig 34: Physics settings of RealFlight**

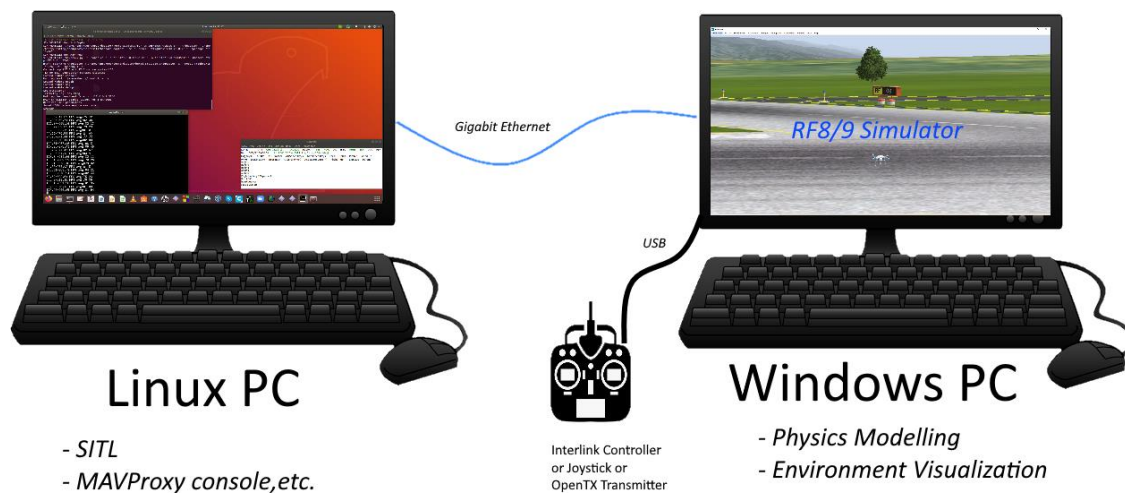
Installing Custom flight model on the Windows PC:

- I. Download the .RFX file for QuadcopterX model from the following GitHub link:
- II. [https://github.com/ParasIntern-SwapnilMathur/SITL\\_Models/blob/master/RealFlight/Released\\_Models/Multicopters/QuadcopterX/QuadcopterX-flightaxis\\_AV.RFX](https://github.com/ParasIntern-SwapnilMathur/SITL_Models/blob/master/RealFlight/Released_Models/Multicopters/QuadcopterX/QuadcopterX-flightaxis_AV.RFX)
- III. On the linux PC, download the parameter file in .txt format from: [https://github.com/ParasIntern-SwapnilMathur/SITL\\_Models/blob/master/RealFlight/Released\\_Models/Multicopters/QuadcopterX/QuadcopterX.param](https://github.com/ParasIntern-SwapnilMathur/SITL_Models/blob/master/RealFlight/Released_Models/Multicopters/QuadcopterX/QuadcopterX.param)
- IV. On RealFlight, go to “Simulation” > “Import” > “RealFlight Archive (RFX, G3X)” > select the QuadcopterX file downloaded above > “OK.”
- V. Now, go to “Aircraft” > “Select Aircraft” > “Custom Aircraft” > “Quadcopterx - flightaxis” > “OK.”

**Fig 35: Custom QuadcopterX model****Fig 36: Importing the QuadCopterX .RFX file**

### 9.3 Configuring the Dual PC Setup

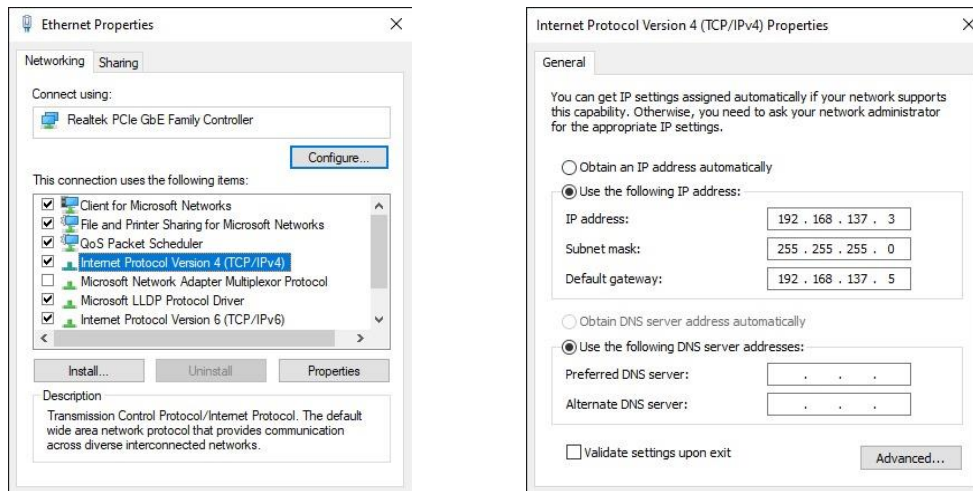
This technique spreads the processing requirements between two PCs: one Windows machine running RealFlight and the physics/flight graphics, and another Linux PC or Linux VM running the SITL models. It also allows you to test and use locally generated code, rather than only the master branch, used by Mission Planner SITL. The layout of the dual PC setup is shown in fig.



**Fig 37: dual PC setup**

Configure the dual PC setup according to the following steps:

- i. Connect the two PCs using an ethernet cable.
- ii. On the linux PC, go to “Network” > “wired setting” > click on the settings icon > go to “IPv4” > enter the IP address, eg: 192.168.137.2 > netmask: 255.255.0.0 > gateway: eg: 192.168.137.4 > click “apply” > turn off and then turn back ON the wired connection to make the changes permanent.
- iii. In the terminal, write: `ip a` to check the IP address.
- iv. On the windows PC, go to “settings” > “network and internet” > “Status” > “Change adapter options” > right click on “Ethernet” > “Properties” > “Internet protocol version 4 (TCP/Ipv4)” > select “use the following IP address” > enter the IP address same as the gateway of the linux PC i.e., 192.168.137.4 > same subnet mask as the linux pc “255.255.0.0” > enter the gateway same as the IP address of the linux PC “192.168.137.2”



**Fig 38: IP configuration on windows PC**

- v. Now both the PCs are connected to each other.
- vi. To check the IP of the windows pc, open the command prompt and enter: `ipconfig`



## 9.4 Running RealFlight Model Using SITL

Once both the PCs are successfully connected and Flightaxis is enabled on realflight. In the linux terminal (The IP address of the windows PC is to be used here):

```
❖ cd ardupilot
❖ cd ArduCopter
❖ sim_vehicle.py -f flightaxis:192.168.137.4 --map -console
```

Now the MAVproxy console will open, it should not show any errors. The ArduCopter console should look like fig. And the Map must have the quadcopter icon.

The SITL is successfully connected to the QuadcopterX model.

In the MAVproxy console:

- Go to “Parameter” > “Load” > select the file that you downloaded earlier from [https://github.com/ParasIntern-SwapnilMathur/SITL\\_Models/blob/master/RealFlight/Released\\_Models/Multicopters/QuadCopterX/QuadCopterX.param](https://github.com/ParasIntern-SwapnilMathur/SITL_Models/blob/master/RealFlight/Released_Models/Multicopters/QuadCopterX/QuadCopterX.param)
- First change the format of the file to “.param” and then Load it into MAVproxy console.

To operate the quadcopter using SITL, in the linux terminal within the `sim_vehicle.py` command, enter:

mode	All the available modes will appear.
mode guided	Guided mode is activated
arm throttle	the propellers start to rotate in realflight.
takeoff 50	the copter takes off and achieves 50 metres altitude which can be seen in the MAVproxy console.

If the copter fails to arm, in the MAVproxy console > “Parameter” > “Editor” > search “ARMING\_CHECK” > unselect all or type “0” > “write” > and then try arming again. If the issue persists, change the parameters of that specific error by searching it in the editor and try arming again OR restart the mission.

### 9.5 Square Path Using the command line

If a square path has to be provided to the copter, the following commands are used:

- Connect the RealFlight copter with the SITL using the commands mentioned above.
- After the communication between SITL and RealFlight is successful and the copter has entered “STABILIZE” mode, In the terminal:
- mode guided
- arm throttle
- takeoff 50

module load message

message SET_POSITION_TARGET_LOCAL_NED 0 0 0 1 3576 200 0 -10 0 0 0 0 0 0 0	Copter flies 200 meters North at an altitude of 10 meters.
message SET_POSITION_TARGET_LOCAL_NED 0 0 0 1 3576 0 200 -10 0 0 0 0 0 0 0	Copter flies 200 meters East at an altitude of 10 meters.
message SET_POSITION_TARGET_LOCAL_NED 0 0 0 1 3576 -200 0 -10 0 0 0 0 0 0 0	Copter flies 200 meters South at an altitude of 10 meters.
message SET_POSITION_TARGET_LOCAL_NED 0 0 0 1 3576 200 0 -10 0 0 0 0 0 0 0	Copter flies 200 meters West at an altitude of 10 meters.

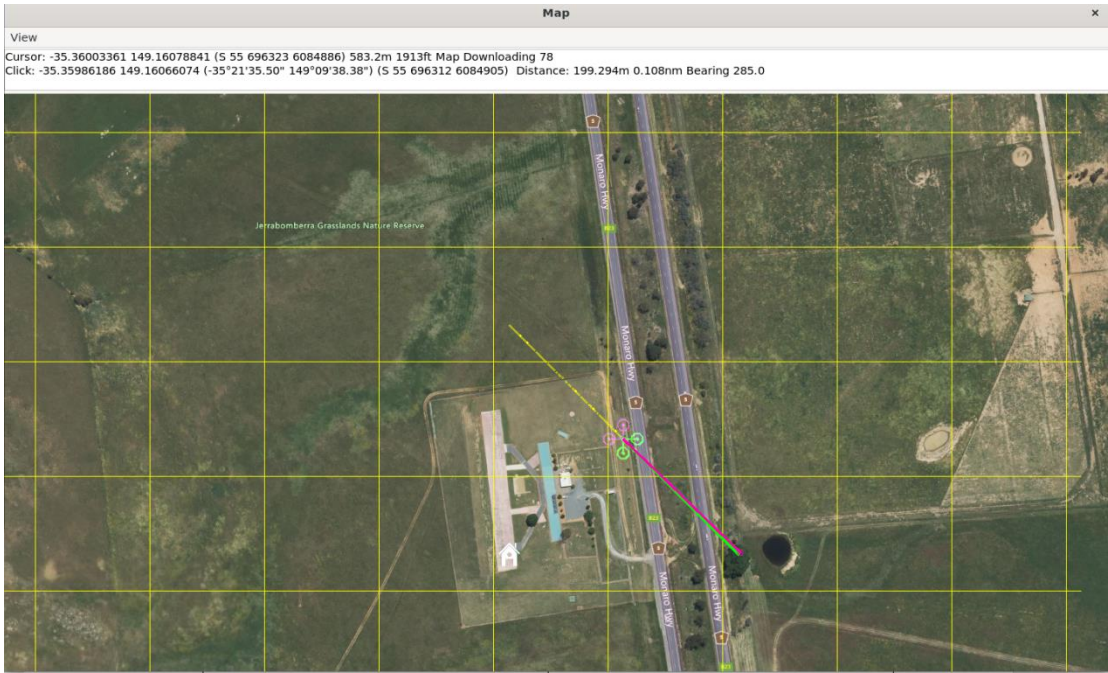


Fig 39: Square Path mission

## 9.6 Camera Survey Mission with Alti Transition VTOL Quadplane

Alti transition is a hybrid UAV which can fly as a quadcopter and can be transitioned to fly as a plane. The objective of this module is to carry out a survey mission using the Alti Quadplane. For this, we need to add and configure a Gimbal cam module with the QuadPlane:

- Download the .RFX file on the windows PC with RealFlight of the Alti model from: [https://github.com/ParasIntern-SwapnilMathur/SITL\\_Models/blob/master/RealFlight/Released\\_Models/QuadPlanes/Alti\\_Transition\\_QuadPlane/AltiTransition\\_with\\_FPV\\_gimbal\\_EA.RFX](https://github.com/ParasIntern-SwapnilMathur/SITL_Models/blob/master/RealFlight/Released_Models/QuadPlanes/Alti_Transition_QuadPlane/AltiTransition_with_FPV_gimbal_EA.RFX)
- Download the parameter file on the Linux PC and save it in “.parm” format using: [https://github.com/ParasIntern-SwapnilMathur/SITL\\_Models/blob/master/RealFlight/Released\\_Models/QuadPlanes/Alti\\_Transition\\_QuadPlane/Alti.param](https://github.com/ParasIntern-SwapnilMathur/SITL_Models/blob/master/RealFlight/Released_Models/QuadPlanes/Alti_Transition_QuadPlane/Alti.param)



**Fig 40: Alti QuadPlane in RealFlight**

1. To configure the Gimbal camera, go to “Aircraft” > “Edit”
2. Select “Physics” then select “airframe” and click on “components” from the options menu above. A dropdown list will appear, click on “Add” and select “Camera” from the dropdown list.
3. edit the “x”, “y” and “z” coordinates > “Save”
4. Connect the SITL with Realflight using the steps mentioned earlier.
5. Enter the following terminal commands:
  - i. mode guided
  - ii. arm throttle

iii. takeoff 50

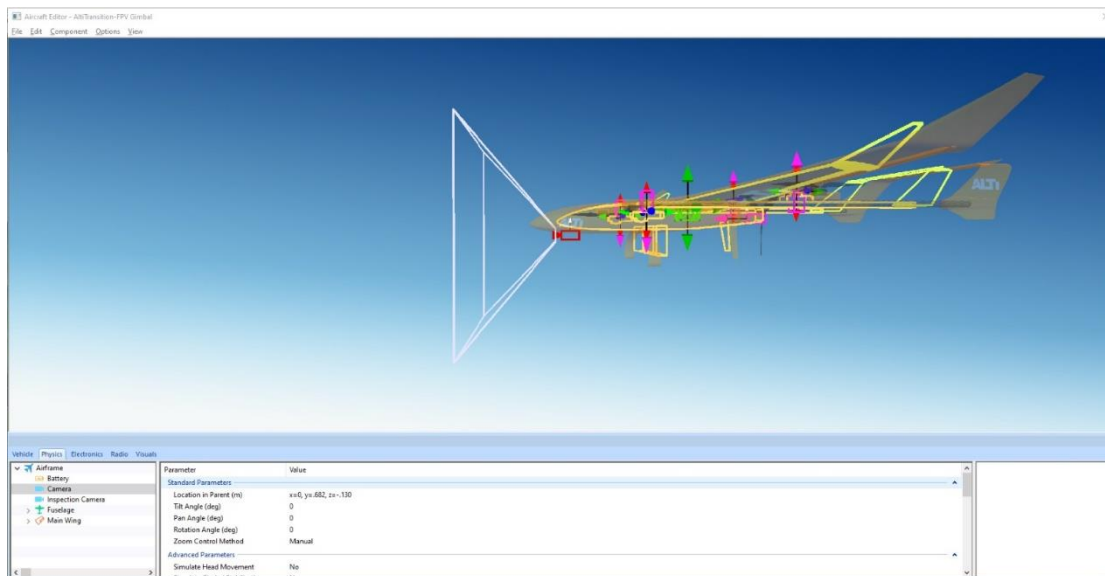
iv. The QuadPlane will take off in copter mode.

If you need to run it in the plane mode, change the directory from Ardupilot to AruPlane using:

v. cd -

vi. cd ArduPlane

vii. and then arm it.



**Fig 41: Setting up Gimbal Camera**

- ❖ To create a mission using waypoints, right click anywhere on the map and select the option “Mission” > “Draw” > enter desired altitude and press “Ok” > add waypoints and draw the mission.
- ❖ Choose the waypoints for take-off, RTL, and landing by right clicking on the waypoint > “Mission” > desired option.
- ❖ Click on “Save” > save the file in “.wp” or “.txt” format.
- ❖ In the terminal: `wp load filename.txt`
  - To start the mission:
  - `mode Auto`

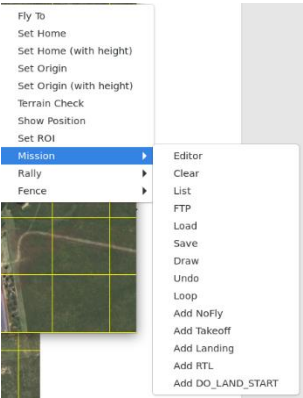


Fig 42: adding and configuring waypoints

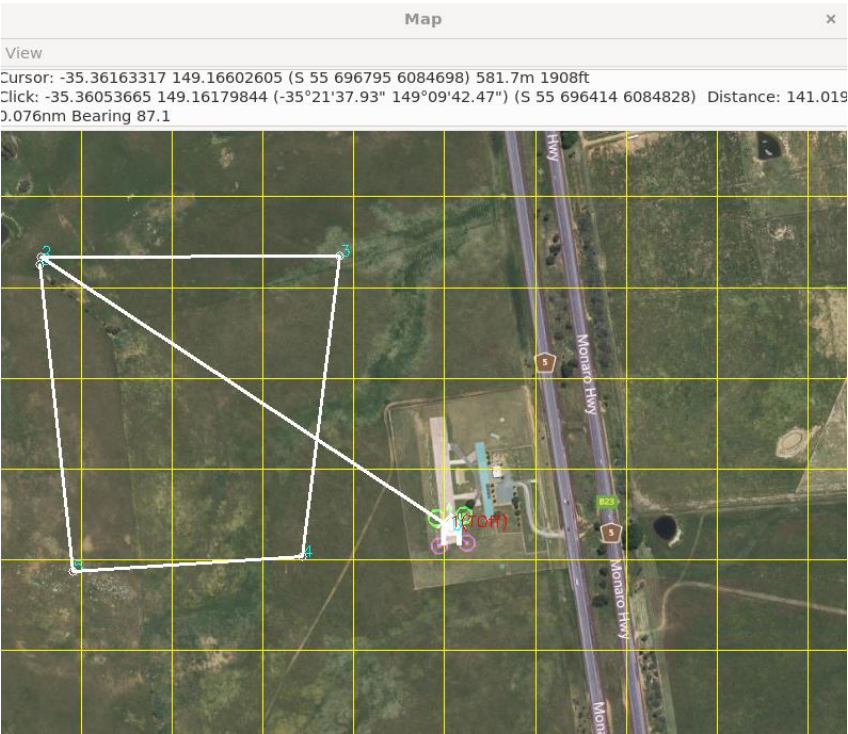


Fig 43: waypoint survey mission

## Chapter 10: Flight Dynamic Model of an Aircraft

To determine the physical forces acting on a simulated aircraft, such as thrust, lift, and drag, a flight dynamics model (FDM) is utilized. It aids in comprehension of the movement, stability, and control of the aircraft and is based on engineering and physics principles. Six degrees of freedom are commonly present in a model, and they are as follows:

- movement along the x-axis (roll)
- movement along the y-axis (pitch)
- movement along the z-axis (yaw)
- movement about the x-axis (roll rate)
- movement about the y-axis (pitch rate)
- movement about the z-axis (yaw rate)

This can be seen in figure 1.

These degrees of freedom each correspond to a particular rotation or motion that an aircraft is capable of. The aircraft's linear motion in three dimensions is represented by the first three degrees of freedom, whereas the latter three degrees of freedom correspond to other motions.

In addition to the six degrees of freedom, the flight dynamics model may also include other factors such as wind, initial mass, and atmospheric conditions.

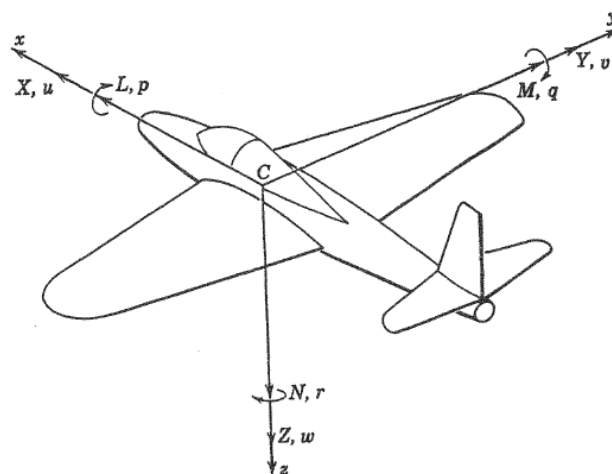


Fig 44: Degrees of Freedom of an Aircraft

### 10.1 Flight Dynamic Equations



### 10.1.1 Six Degrees of Freedom

1. Newton's Laws: Newton's second law in vector form:

$$\vec{F} = \sum_i \vec{F}_i \quad m \frac{d}{dt} \vec{V}$$

$\vec{F} = [F_x \ F_y \ F_z]$  and  $\vec{V} = [u \ v \ w]$ , then

$$F_x = m \, du/dt$$

$$F_y = m \, dv/dt$$

$$F_z = m \, dw/dt$$

2. Using Calculus, this concept can be extended to rigid bodies by integration over all particles.

Angular momentum of a rigid body can be found as:

$$\vec{H} = I \omega_I$$

where  $\omega_I = [p, q, r]^T$  is the angular rotation vector of the body about the centre of mass.

- p is rotation about the x-axis.
- q is rotation about the y-axis.
- r is rotation about the z-axis.
- I is defined in an Inertial Frame.

The matrix I is the Moment of Inertia Matrix. The moment of inertia matrix is defined as:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

$$I_{xy} = I_{yx} = \int \int \int xy \, dm \quad I_{xx} = \int \int \int (y^2 + z^2) \, dm$$

$$I_{xz} = I_{zx} = \int \int \int xz \, dm \quad I_{yy} = \int \int \int (x^2 + z^2) \, dm$$

$$I_{yz} = I_{zy} = \int \int \int yz \, dm \quad I_{zz} = \int \int \int (x^2 + y^2) \, dm$$

So

$$\begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

where pI, qI and rI are the rotation vectors as expressed in the inertial frame corresponding to x-y-z.

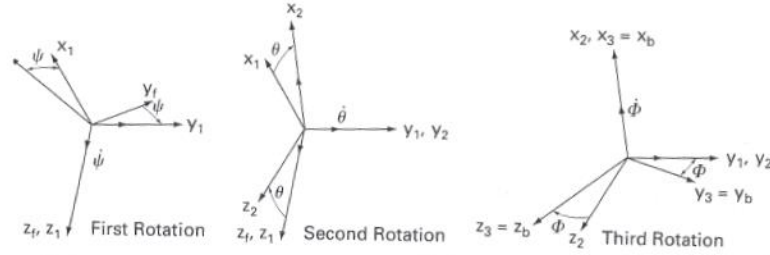
3. Equation of Motion:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix}$$

and

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx}\dot{p} - I_{xz}\dot{r} - qpI_{xz} + qrI_{zz} - rqI_{yy} \\ I_{yy}\dot{q} + p^2I_{xz} - prI_{zz} + rpI_{xx} - r^2I_{xz} \\ -I_{xz}\dot{p} + I_{zz}\dot{r} + pqI_{yy} - qpI_{xx} + qrI_{xz} \end{bmatrix}$$

4. Euler Angles: The term Euler Angles refers to the angles of rotation ( $\phi$ ,  $\theta$ ,  $\psi$ )



**Roll Rotation ( $\phi$ ):**

$$R_1(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix}$$

**Pitch Rotation ( $\theta$ ):**

$$R_2(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$

**Yaw Rotation ( $\psi$ ):**

$$R_3(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5. The rate of rotation of the Euler Angles can be found by rotating the rotation vector into the inertial frame.

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

6. This can be inverted as:

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

Where,

L Rolling Moment Net Moment in the positive p-direction

M Pitching Moment Net Moment in the positive q-direction

N Yawing Moment Net Moment in the positive r-direction



## 10.2 JSBSim

JSBSim is an open-source flight dynamics model (FDM) that compiles & runs under many operating systems, including Microsoft Windows, Apple Macintosh, Linux, IRIX, Cygwin (Unix on Windows), etc. The FDM is essentially the physics/math model that defines the movement of an aircraft, rocket, etc., under the forces and moments applied to it using the various control mechanisms and from the forces of nature. JSBSim has no native graphics. It can be run by itself as a standalone program, taking input from a script file and various vehicle configuration files. It can also be incorporated into a larger flight simulator implementation that includes a visual system. Its Features include:

- Fully configurable flight control system, aerodynamics, propulsion, landing gear arrangement, etc. through XML-based text file format.
- Configurable data output formats to screen, file, socket, or any combination of those.
- Non-linear 6 DoF (Degree of Freedom)
- Accurate Earth model including:
  1. Rotational effects on the equations of motion (Coriolis and centrifugal acceleration modelled).
  2. Oblate spherical shape and geodetic coordinates according to the WGS84 geodetic system.
  3. Atmosphere modelled according to the International Standard Atmosphere (1976).

### 10.2.1 Installing the Software

#### 1. Installing Required Software

You need to have the [Git software](#) installed & [CMake](#)

#### 2. Downloading from [GitHub](#)

`git clone https://github.com/JSBSim-Team/jsbsim.git jsbsim-code` (**HTTPS mode**)

`git clone git@github.com:JSBSim-Team/jsbsim.git jsbsim-code` (**SSH mode**)

#### 3. Building with CMake. It can produce files to build JSBSim with GNU make. It is preferable to build JSBSim in a separate directory.

> `cd jsbsim-code`

> `mkdir build`

```
> cd build
```

4. First, you should invoke CMake and then execute make

```
> cmake ..
```

```
> make
```

This will compile the various classes, and build the JSBSim application which will be located in build/src.

5. Installing JSBSim

```
> make install
```

### 10.2.2 Running JSBSim

- Run the Command **JSBSim**

```
kp@kp:~/Downloads/gitg-3.32.1$ JSBSim

JSBSim version 1.2.0.dev1 Apr  4 2023 11:00:38

Usage: jsbsim [script file name] [output file names] <options>

options:
--help    returns this message
--version returns the version number
--outputlogfile=<filename> sets (overrides) the name of a data output file
--logdirectivefile=<filename> specifies the name of a data logging directives file
        (can appear multiple times)
--root=<path> specifies the JSBSim root directory (where aircraft/, engine/, etc. reside)
--aircraft=<filename> specifies the name of the aircraft to be modeled
--script=<filename> specifies a script to run
--realtime specifies to run in actual real world time
--nice    specifies to run at lower CPU usage
--nohighlight specifies that console output should be pure text only (no color)
--suspend specifies to suspend the simulation after initialization
--initfile=<filename> specifies an initialization file
--catalog specifies that all properties for this aircraft model should be printed
        (catalog=aircraftname is an optional format)
--property=<name=value> e.g. --property=simulation/integrator/rate/rotational=1
--simulation-rate=<rate (double)> specifies the sim dt time or frequency
        If rate specified is less than 1, it is interpreted as
        a time step size, otherwise it is assumed to be a rate in Hertz.
--end=<time (double)> specifies the sim end time

NOTE: There can be no spaces around the = sign when
      an option is followed by a filename
```

**Fig 45:**

#### Information of JSBSim and its commands

- Run the command JSBSim --script=scripts/c1723.xml in source directory
- As we can see the program gives output on bases of various parameters like:
  - i. Aircraft Metrics
  - ii. Mass and Balance
  - iii. Ground Reactions
  - iv. Propulsion
  - v. Navigation, etc.

```

kp@kp:~/Khush/jsbsim$ JSBSim --script=scripts/c1723.xml

JSBSim Flight Dynamics Model v1.2.0.dev1 Apr  4 2023 11:00:38
[JSBSim-ML v2.0]

JSBSim startup beginning ...

Reading Aircraft Configuration File: Cessna C-172 Skyhawk II
Version: 2.0

This aircraft model is a BETA release!!!

This aircraft model probably will not fly as expected.

Use this model for development purposes ONLY!!!

Description:  Models a 1982 Cessna 172P.
Model Author:  Tony Peden
Creation Date: 1999-01-01
Version:      $Revision: 1.91 $

Aircraft Metrics:
WingArea: 174.000000
WingSpan: 36.000000
Incidence: 0.000000
Chord: 4.900000
H. Tail Area: 21.900000
H. Tail Arm: 15.700000
V. Tail Area: 16.500000
V. Tail Arm: 15.700000
Eyepoint (x, y, z): 37.000000 , 0.000000 , 48.000000

```

Fig 46: Aircraft Metrics of Cessna C -172 Shyhawk II

For Ardupilot:

Run the Command

> sim\_vehicle.py -v ArduPlane -f jsbsim:pc7 --console --map

```

kp@kp:~/ardupilot$ sim_vehicle.py -v ArduPlane -f jsbsim:pc7 --console --map
SIM_VEHICLE: Start
SIM_VEHICLE: Killing tasks
SIM_VEHICLE: Starting up at SITL location
SIM_VEHICLE: WAF build
SIM_VEHICLE: Configure waf
SIM_VEHICLE: "/home/kp/ardupilot/modules/waf/waf-light" "configure" "--board" "sitr"

```

Fig 47: Building of Sim\_vehicle.py and JSBSim

```
ArduPlane
----- JSBSim Execution beginningOpened JSBSim control socket
... -----

start_engine (Event 0) executed at time: 0.000833

Start: Tuesday April 25 2023 11:13:39 (HH:MM:SS)
Ground Trim

Trim successful
Trim Results:
    Altitude AGL: 2.71 wdot: 4.08e-04 Tolerance: 1e-03 Passed
    Pitch Angle: 0.03 qdot: 3.06e-07 Tolerance: 1e-04 Passed
    Roll Angle: -0.00 pdot: -1.45e-09 Tolerance: 1e-04 Passed

Trim (Event 1) executed at time: 0.010000

GEAR_CONTACT: 0.010000 seconds: NOSE 1
GEAR_CONTACT: 0.010000 seconds: LEFT_MAIN 1
GEAR_CONTACT: 0.010000 seconds: RIGHT_MAIN 1
validate_structures:489: Validating structures
Loaded defaults from Tools/autotest/default_params/plane-jsbsim.parm
```

Fig 48: JSBSim connected with ArduPilot

## 10.3 MATLAB Simulation

MATLAB Simulink provides a set of tools for building and simulating flight dynamic models. To get started with creating a flight dynamic model in MATLAB Simulink, follow these steps:

1. Start by creating a new Simulink model.
2. Open the Simulink Library Browser and navigate to the Aerospace Blockset library.
3. From the Aerospace Blockset library, select the blocks required for building your flight dynamic model. For example, you may need blocks for modeling aircraft motion, atmospheric conditions, control systems, and sensors.
4. Assemble the blocks into your model using the Simulink Editor as show in figure vi, vii, viii and ix.

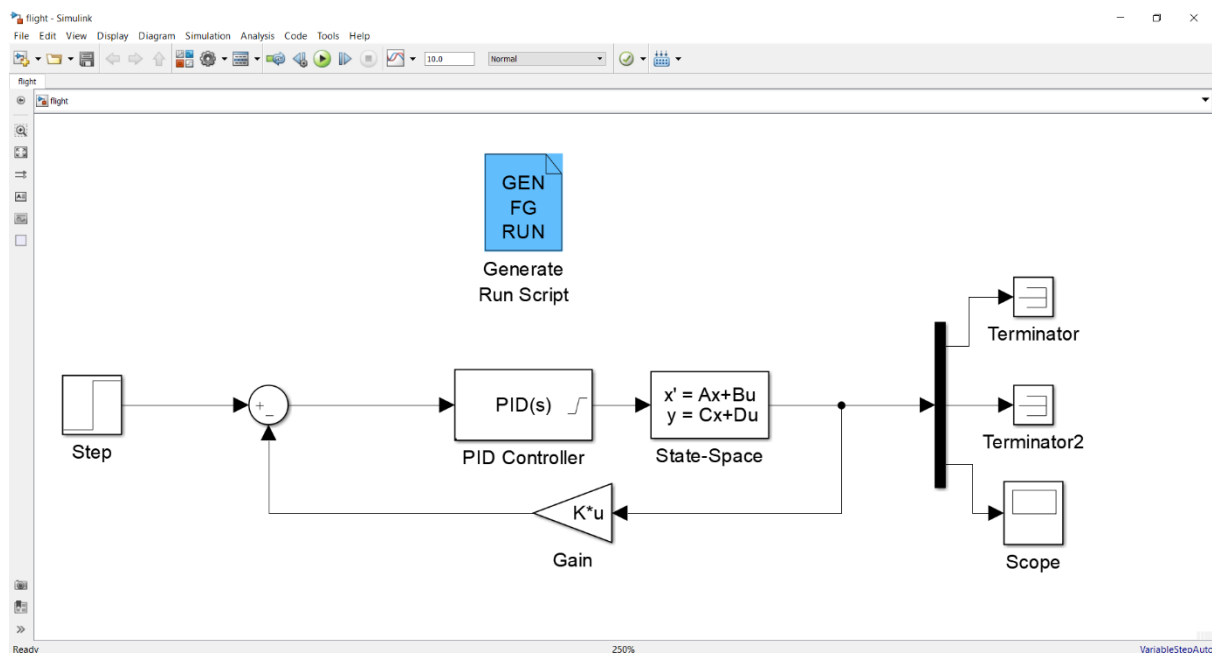


Figure 49: Simulink model

5. Configure the parameters of each block to suit your model requirements.

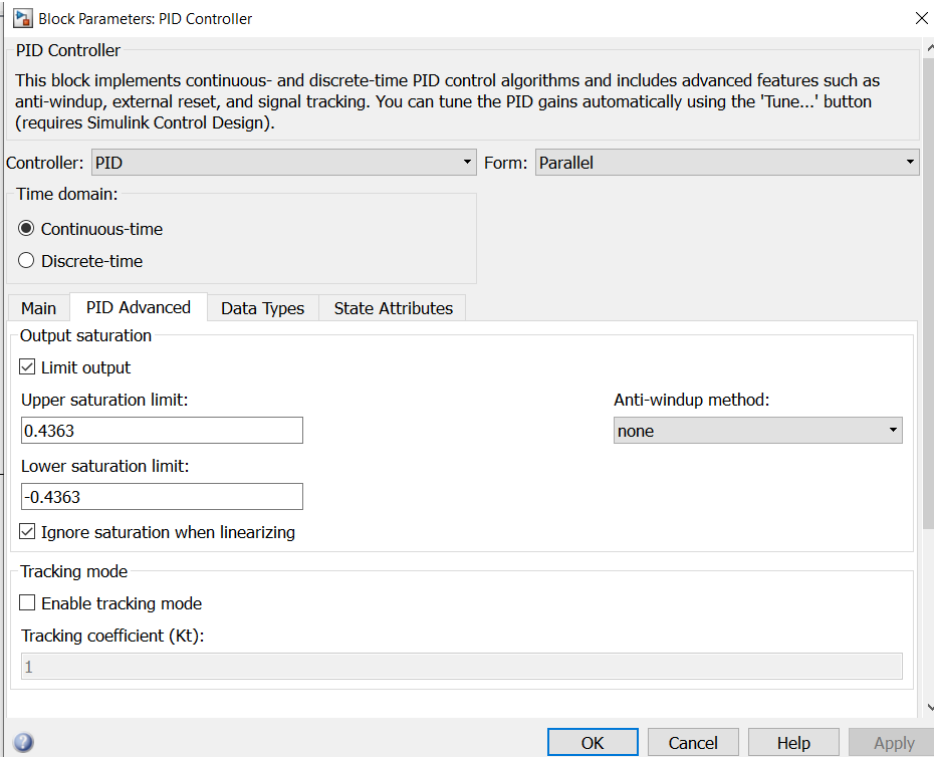


Figure 50: PID Controller configuration

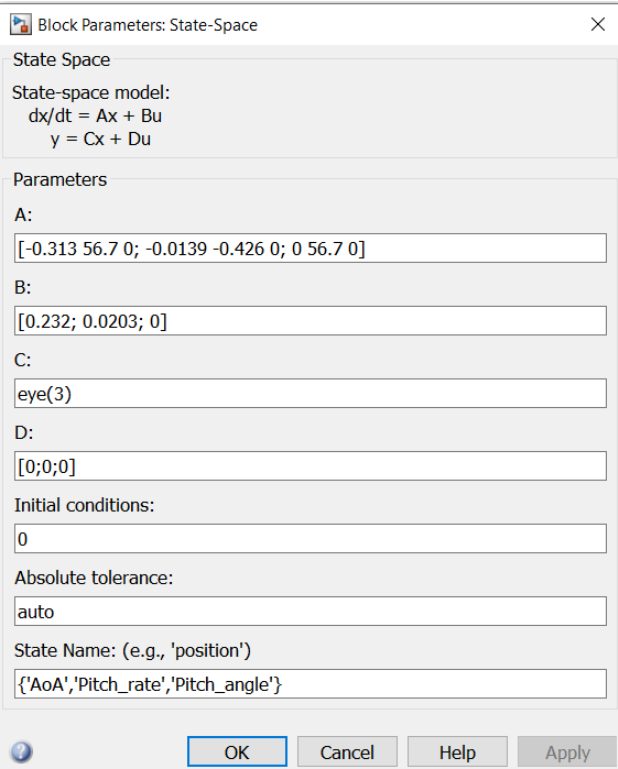


Figure 51: State Space Configuration

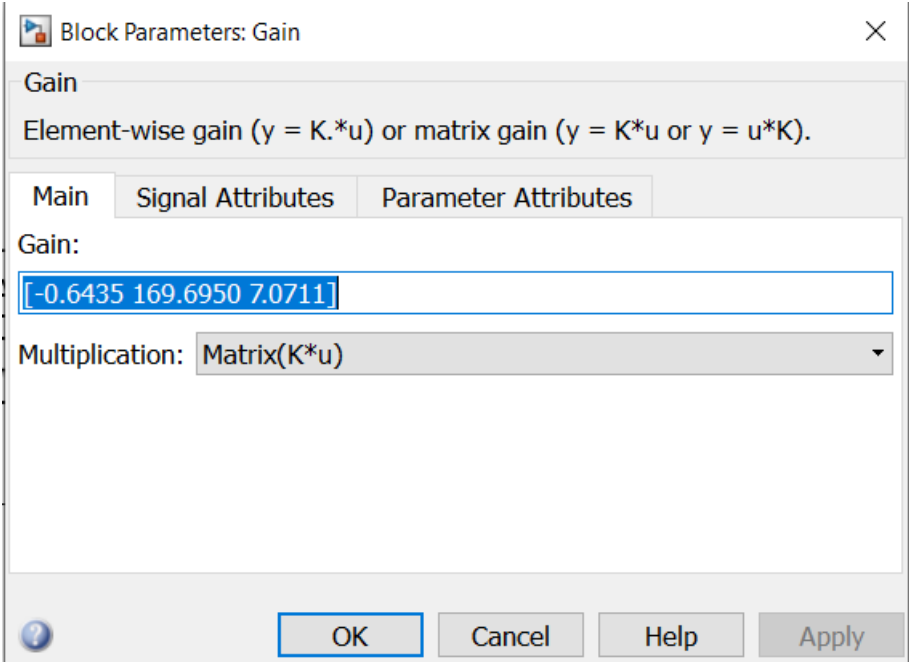


Figure 52: Gain Value

6. Connect the blocks in your model using signal lines to define the flow of data.
7. In the file tab, mention the latest flightgear base directory.
8. Run the generate file and click generate script.

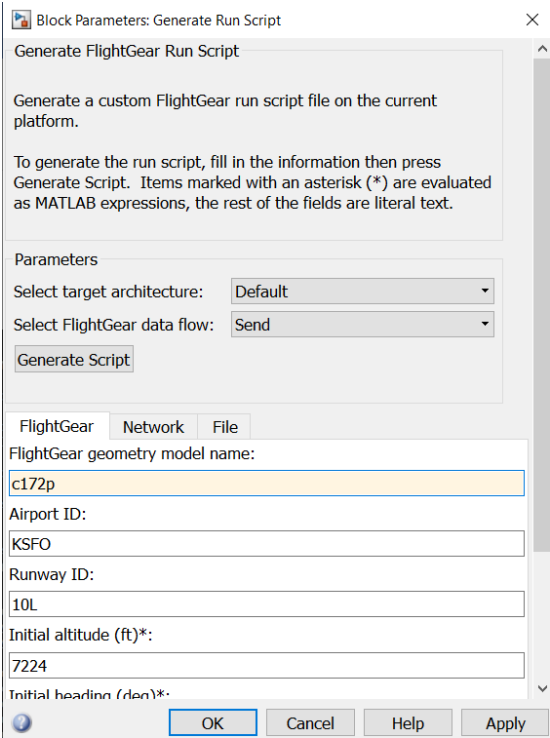


Figure 53: generating

run script

8. Edit the bat file according to the flight that you have selected. For the aircraft model c172p:

```

C:

cd C:\Program Files\FlightGear 2020.3

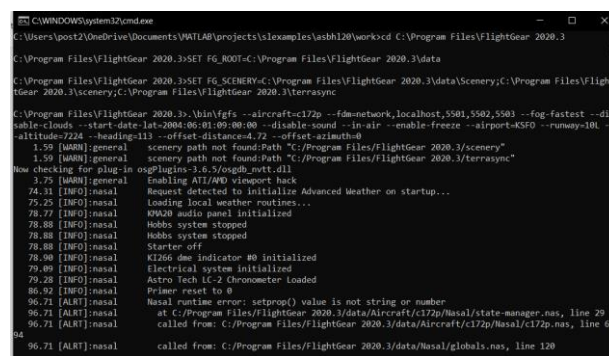
SET FG_ROOT=C:\Program Files\FlightGear 2020.3\data

SET FG_SCENERY=C:\Program Files\FlightGear 2020.3\data\Scenery;C:\Program
Files\FlightGear 2020.3\scenery;C:\Program Files\FlightGear
2020.3\terrasyne

.\bin\fgfs --aircraft=c172p --fdm=network,localhost,5501,5502,5503 --fog-
fastest --disable-clouds --start-date-lat=2004:06:01:09:00:00 --disable-
sound --in-air --enable-freeze --airport=KSFO --runway=10L --altitude=7224
--heading=113 --offset-distance=4.72 --offset-azimuth=0

```

## 9. Run the bat file.



```

C:\WINDOWS\system32\cmd.exe
C:\Users\post2\OneDrive\Documents\WATLAB\projects\slxamples\ash120\work\cd C:\Program Files\FlightGear 2020.3
C:\Program Files\FlightGear 2020.3>SET FG_ROOT=C:\Program Files\FlightGear 2020.3\data
C:\Program Files\FlightGear 2020.3>SET FG_SCENERY=C:\Program Files\FlightGear 2020.3\data\Scenery;C:\Program Files\FlightGear 2020.3\scenery;C:\Program Files\FlightGear 2020.3\terrasyne
C:\Program Files\FlightGear 2020.3>.\bin\fgfs --aircraft=c172p --fdm=network,localhost,5501,5502,5503 --fog-fastest --d1
enable-clouds --start-date-lat=2004:06:01:09:00:00 --disable-sound --in-air --enable-freeze --airport=KSFO --runway=10L --
altitude=7224 --heading=113 --offset-distance=4.72 --offset-azimuth=0
1.59 [WARN]:general scenery path not found:Path "C:/Program Files/FlightGear 2020.3/scenery"
1.59 [WARN]:general scenery path not found:Path "C:/Program Files/FlightGear 2020.3/terrasyne"
Now checking for plug-in osgPlugins-3.4.3/fgosg_nrt.dll
3.75 [WARN]:general Enabling ATI/AMD viewport hack
24.11 [INFO]:nasal Request detected to initialize Advanced Weather on startup...
75.25 [INFO]:nasal Loading local weather routines...
78.77 [INFO]:nasal KWA20 audio panel initialized
78.88 [INFO]:nasal Hobbs system stopped
78.88 [INFO]:nasal Hobbs system stopped
78.88 [INFO]:nasal Starter off
78.90 [INFO]:nasal KI266 dme indicator #0 initialized
79.09 [INFO]:nasal Electrical system initialized
79.28 [INFO]:nasal Astro Tech LC-2 Chronometer loaded
86.92 [INFO]:nasal Primer reset to 0
96.71 [ALRT]:nasal Nasal runtime error: setprop() value is not string or number
96.71 [ALRT]:nasal at C:/Program Files/FlightGear 2020.3/data/Aircraft/c172p/Nasal/state-manager.nas, line 29
96.71 [ALRT]:nasal called from: C:/Program Files/FlightGear 2020.3/data/Aircraft/c172p/Nasal/c172p.nas, line 6
96.71 [ALRT]:nasal called from: C:/Program Files/FlightGear 2020.3/data/Nasal/globals.nas, line 120

```

Figure 54: Command Window



Figure 54: Flightgear visualisation of flight dynamics



## References

- [1] Lu, W., Zhang, D., Zhang, J., Li, T., & Hu, T. (2017, May). Design and implementation of agasoline-electric hybrid propulsion system for a micro triple tilt-rotor VTOL UAV. In 2017 6th Data Driven Control and Learning Systems (DDCLS) (pp. 433-438). IEEE.
- [2] Cappello, F., Ramasamy, S., & Sabatini, R. (2016).
- [2] A low-cost and high performance navigation system for small RPAS applications. *Aerospace Science and Technology*, 58, 529-545.
- [3] Yage Wang, Mayao Gao. (2022, September). Attitude Control System of Quadcopter Based on Four-Element Method-Double Layer PID Algorithm
- [4] WANG P, Zhang ZF, Cao MC. Research status and development of multi-UAVS formation based on consensus. *Ship Electronic Engineering*. 2017; 37(09):1-9.
- [5] Zou YF, Liu W, Yan ZW. Attitude control system of quadrotor based on PID neural network. *Transducer and Microsystem Technologies*. 2020 Jan; 39(1): 68-71.
- [6] Chen HY, Liu XH, Wang PZ. Design of formation control system for small quadcopter based on STM32. *Modern Computer*. 2021 Jan; 2(1):115-120.