

Master of Aerospace Engineering Research Project

Autonomous Robotic Aerial Vehicle's Control in Complex Environments

S3 Project report

Author(s): [Karthik Mallabadi, Ryan Xavier Dsouza](#)

Due date of report: [30 / 03 / 2022](#)

Actual submission date: [29 / 03 / 2022](#)

Starting date of project: 25/Jan/2021

Duration: 14 Months

Tutors: [Raghuvamsi Deepthimahanthi, Yves Briere](#)

This document is the property of the ISAE SUPAERO and shall not be distributed
nor reproduced without the formal approval of the tutors.

Table of Contents

1	Introduction	1
2	Semester 3 section	2
2.1	Context and key issues.....	2
2.2	Work done during Semester 2	2
2.3	Work to be done during Semester 3	4
3	Investigation Methods	5
3.1	Vehicle Components.....	5
3.1.1	Air Module	5
3.1.2	Ground Module	6
3.2	Software Modules	6
3.3	Communication Algorithm	8
3.4	Simulation Set-up for the Hybrid Vehicle	9
3.5	Safe Landing Algorithm	10
3.5.1	Energy Estimation	11
3.5.2	Safe Landing Planner	12
3.5.3	Simulation Set-Up	13
3.6	Real-time Vehicle Setup	16
3.6.1	Pixhawk and Raspberry Pi connection	16
3.6.2	Offboard Mode Vehicle Setup	18
3.6.3	MoCap System Configuration for Position Estimation.....	19
4	Results and analysis.....	20
4.1	Simulation Results for Communication Model	20
4.2	Simulation Results for Safe Landing Algorithm	25
4.3	Testing of Hybrid Vehicle	29
5	Conclusion and perspectives	32
6	References	33

Declaration of Authenticity

This assignment is entirely our own work. Quotations from literature are properly indicated with appropriated references in the text. All literature used in this piece of work is indicated in the bibliography placed at the end. We confirm that no sources have been used other than those stated.

We understand that plagiarism (copy without mentioning the reference) is a serious examinations offence that may result in disciplinary action being taken.

Date:29/03/2022

Signature



Ryan Xavier Dsouza



Karthik Mallabadi

Abstract

Unmanned Aerial Vehicles (UAV) have a wide spectrum of applications such as surveying or delivery, while rovers are commonly used in space and cross terrain missions. However, the combination of both paves' way to a new field of research. The main goal of this project is to program an existing hybrid vehicle which will be fully autonomous and can move both in ground and air.

In order to implement the hybrid model, a switching module between the air and ground module is implemented. The working of this model is discussed and is validated by performing simulations in Gazebo environment. Safety of the vehicle is crucial and in order to satisfy this aspect, a safe landing algorithm which is able to make decision on whether to fly or land based upon the battery information will be simulated in ROS environment.

Prior to the real time testing of the vehicle, communication between the Flight controller unit (Pixhawk) and the onboard computer (Raspberry Pi) is established. A detailed explanation of the communication model will be discussed. Results achieved from the offboard control on the hybrid vehicle prototype will also be presented.

Finally, the report will conclude with the key findings of this work and also the future aspects of this project.

Keywords : UAV, Rover, Communication, Safe Landing

1 Introduction

In recent years, rovers and UAVs have been of interest due to their wide spectrum of applications. Rovers are generally used in space and cross terrain missions. These vehicles are highly efficient and can easily traverse all types of terrains whilst being robust in all conditions. Due to this there is a large demand in space exploration, in battlefield and disaster zone type conditions.

On the other hand, Unmanned Aerial Vehicles have risen in popularity for several reasons. The main ones being: there is no onboard pilot, quite easy to use, comparatively inexpensive, and most of all versatility. Its versatility comes from the fact that the operator can easily optimize the UAV for a large range of missions from photography to reconnaissance to agricultural applications and many more. UAVs possess high agility which allows them to commute places faster but at an expense of consuming more power.

The main focus of this project is to program an autonomous hybrid model of UAV and rover which combines the capabilities of both systems. To achieve this goal, firstly, an obstacle avoidance and path-planning for the vehicle is necessary. Based on the path generated by the avoidance algorithm, a decision will be made by the vehicle to whether fly or use the ground module. This decision will depend on three key parameters: power, time and energy. The choice between the modules will be carried out by the switching module.

In case the rover lacks power to follow the path, the air module takes over. Moreover, when it comes to saving energy, the ground module is preferred, as long as the avoidance can be done on the ground. On the other hand, air module is preferred when there is a time constraint, as it is faster when compared to the ground module. Secondly, there is a need to address the safety of the vehicle. The vehicle should land on its own in the following cases: mission is completed, battery emergency, switch to ground module. While doing so, the vehicle should have the capability of detecting the terrain underneath it so that the damages are avoided while landing.

This document outlines a brief summary of what has been done in the previous semester, the main objectives for this semester, the process followed in achieving those objectives and the key findings along with conclusions and future scope.

2 Semester 3 section

2.1 Context and key issues

This work has been previously conducted by Anton Sambalov, Sakshi Chaudhary and Nandhini Raghunathan, who have designed and assembled a vehicle composed by a ground and an air module. Furthermore, Anton Sambalov also worked on the set up of different types of equipment as well as on the communication between them. Posteriorly, some simulations of required algorithms are also performed [1].

Previous year, Ricardo Rodriguez continued the project and has setup a baseline path-planning and obstacle avoidance algorithm for the air module along with the complete obstacle avoidance algorithm for the ground module [2] [3]. In the previous semester, notable progress has been achieved in the path-planning module and the safe landing module [4]. A brief summary of the the work done in the previous semester is given in section 2.2. The main focus is then to work on the switching module which controls the switch between air and ground module and also improvise the safe landing module implemented in the previous semester. For the vehicle to be autonomous, there is a need to establish communication link between both the controllers of ground and air module.

One of the main issues involved in this work is to implement a decision making model based on the constraints of power, time and energy within the switching module. Next challenge will be to develop a safe landing algorithm, which detects flat areas underneath the vehicle suitable for the landing in case of emergencies.

2.2 Work done during Semester 2

In the previous semester, a significant progress in the project is achieved. A path-planning algorithm for the air module is chosen and implemented based on the comparison study performed on several path-planning algorithms available in the literature. A*, Dijkstra, and D* Lite algorithms are investigated and a comparative study is carried out. Results are shown in Figure 1.

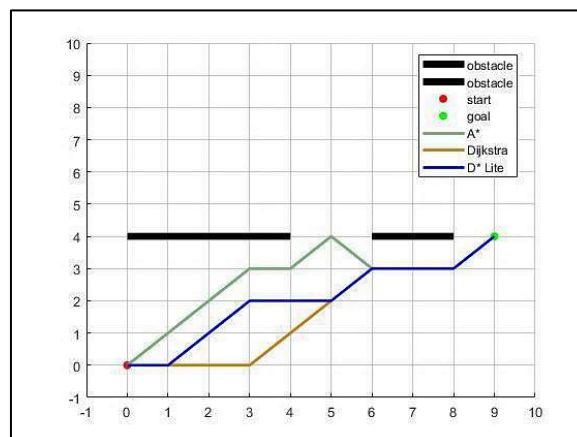


Figure 1 2D map showing paths generated from A*, Dijkstra, and D* Lite algorithms

It is observed that A* algorithm had the longest path among the three algorithms. This is because of the heuristic estimation in the algorithm which tries to reduce the gap between the current node and the goal node. Hence it starts moving in diagonal in the beginning. In between two obstacles, the current node is first reached according to the heuristic and then it encounters an obstacle, then it tries to avoid the obstacle and moves towards the goal. Since no other search branches present nearby, the path generated is longer. Dijkstra and D* Lite have same path length but the search criteria are different. From the comparison study, it is clear that D* Lite algorithm better suited our application as it had ability to adapt to dynamic environment.

The selected D* Lite algorithm is extended to 3D environment. Two versions of the algorithm are considered. In version 1, two neighbor nodes, one up and one down, are added to the search criteria of the algorithm and the heuristics function are updated accordingly. In version 2, The heuristic function is further updated to adapt to the 3D distance between the nodes. The neighbouring nodes are updated such that every node surrounding a given node is searched while computing the path.

It is observed in Figure 2 that, the path generated by version 2 is longer than that of version 1, nevertheless, it is the preferable one because ,the vehicle requires less energy to follow this path. In the path from version 1, when vehicle is near the goal point, it spends more energy to kill the momentum in x- and y-axes and then reduce the altitude to the goal point. But in version 2, the vehicle kills the momentum after it reaches the goal point.

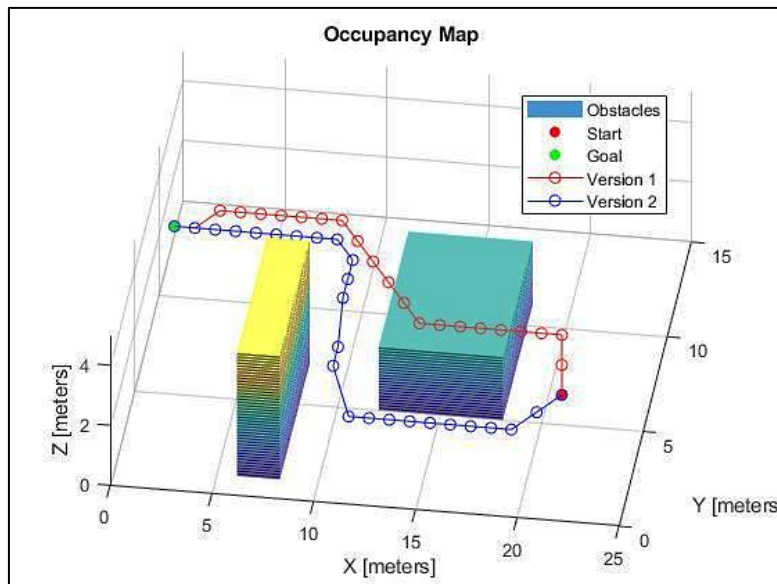


Figure 2 3D map showing path generated for different start and goal point

Next, an algorithm focusing on the safety of the vehicle is implemented. This algorithm forces the vehicle to land whenever the battery level falls below the threshold value (40%). This algorithm helps to avoid crashes due to critically low battery and also protects the vehicle from

getting damaged while landing due to rough or irregular terrains. The safe landing planner is simulated using QGC. A mission is defined by specifying waypoints in QGC. Rviz which is another simulator compatible with ROS is used to observe the behaviour of the vehicle. The results obtained using this algorithm are shown in Figure 3

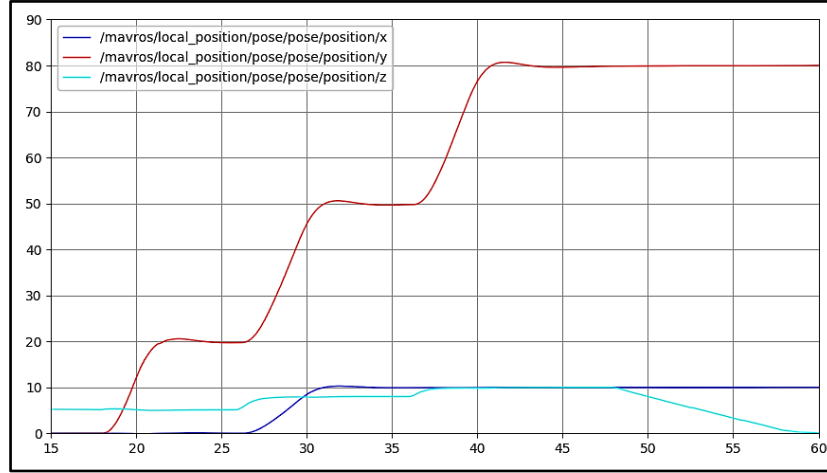


Figure 3 Position of the vehicle

When the simulation starts, the battery percentage is 1, which means it is fully charged. The waypoints are followed until the available battery percentage reached the 40% threshold. Around $t=45$ seconds, the battery falls below this threshold and the z-coordinate of the vehicle's position decreases until it becomes zero (see Figure 3), in other words, the vehicles lands.

2.3 Work to be done during Semester 3

The main focus of this semester is to develop a communication algorithm which performs the switching between air and ground module. The switching is done based on the user-defined waypoints (x, y, z co-ordinates) for the vehicle. The ground module is executed whenever the z co-ordinates is less than 0.5m. Air module is implemented when the z input is greater than 0.5m. However, in the future, decision to switch shall be performed depending on the energy, time and power constraints. In the previous semester, vehicle is told to land whenever the battery dropped down below 40%, however the detection of terrain underneath the vehicle while landing is not performed. Hence, the next objective is to upgrade the safe landing algorithm with the ability to detect terrain underneath the vehicle and land only when the ground is flat. Establishing communication between the flight controller unit (PX4) and the onboard computer (Raspberry Pi) is another task that is to be carried out. Final objective of this work is to test this communication set-up on the real-vehicle and fly the vehicle with user-defined waypoints.

3 Investigation Methods

3.1 Vehicle Components

As previously mentioned, the robotic aerial vehicle is a hybrid system composed of air module which is a quadcopter and a ground module which consists a rover. Both modules have different controllers; Raspberry Pi micro-controller controls the ground module and also communicates with the air module, while Pixhawk PX4 flight controller controls the air module. The full model is shown in Figure 4. The components used in both air and ground module are mentioned in the following two sections.

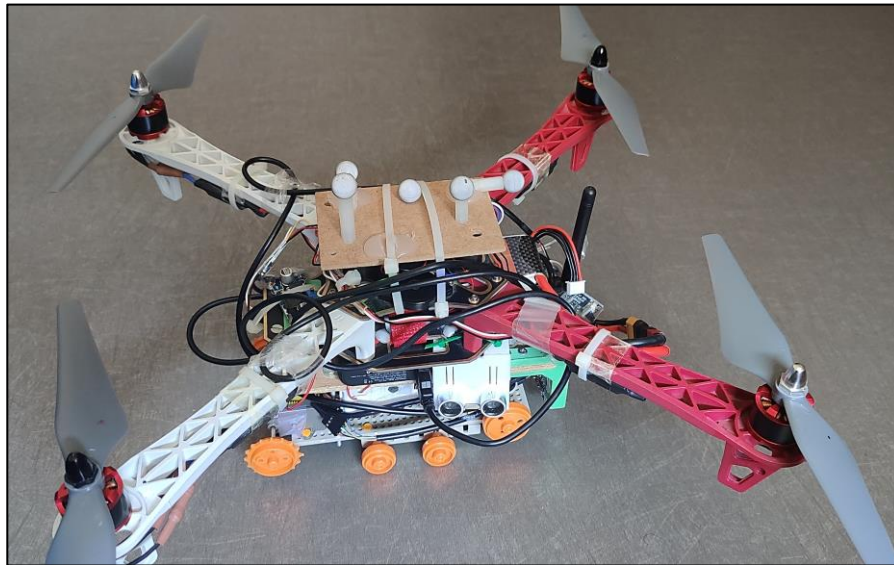


Figure 4 Full assembled hybrid model

3.1.1 Air Module

The air module of a vehicle consists of the components listed below:

- Frame: F450 Drone frame kit, which is a X-type frame with 450mm wheelbase is used for this project.
- Power Distribution Board (PDB): Included in F450 Drone frame kit.
- Motors: Vehicle has four of the 2212 920kV brushless DC motors, meaning they have a stator width of 22mm and height of 12mm.
- Propellers: Propellers of type 9450 are used (9.4-inch diameter, 5.0-inch pitch).
- Electronic Speed Controllers (ESC): Four of the 30A ESCs are used in this project.
- Battery: Floureon 3S, 11.1V, 3000mAh, 30C LiPo battery is used to power up the vehicle.
- Flight Controller: The vehicle is equipped with a Pixhawk, which serves as a flight controller.

3.1.2 Ground Module

The ground module is comprised of a rover that has a base similar to a tank. The rover consists of continuous track which provides maximum control on rough terrain and furthermore, prevents the vehicle from getting stuck. The ground module is made of the following components:

- Frame: Tamiya universal plate set is used as a frame.
- Tracks and Wheels: Ground module is based on Tamiya 70100 track and wheel set.
- Gearbox: Tamiya 70097 twin-motor gearbox consists of the gearboxes and two independent brushed DC motors, which are used to drive two shafts. Motors run on 3-6V.
- Power System: Floureon 3S, 11.1V, 3000mAh, 30C LiPo battery.
- Motor Driver: The TB6612FNG motor driver is able to control speed and direction of the two rear motors. Having a supply range of 2.5V to 13.5V, it suits our motors.

Computer: Raspberry Pi is used as the onboard computer to control the ground module and communicate with the pixhawk. A 10000 mAh power bank to power the Pi is used in the vehicle.

3.2 Software Modules

This section details out the software systems that are used throughout the project work. The Robot Operating System (ROS) is used as a framework. It consists of libraries and tools which allow to design and coordinate robot software. A ROS-based process is called a node and different nodes communicate between each other by sending messages via ROS. The concept of node will be very important throughout the whole content of this report.

Finding a suitable simulation environment for testing algorithms on the vehicle is very crucial as it allows us to validate our algorithms in a simulated environment before actually testing the vehicle in the real world. Furthermore, having the luxury of observing and analyzing the performance of algorithms in 3D is very important for understanding how the robot is going to behave in the real world. The selected simulation environment is the open-source 3D simulator Gazebo [11] which is compatible with ROS. It provides a robust physics engine which is able to detect collisions between the robot and the surrounding environment, providing the user with a precise estimation of what would happen in reality. It also enables the utilization of sensor models, which provide the sensor data to the robot. For example, the sensor data coming from the Gazebo sensor models can be published on a ROS topic. Topics are the medium through which ROS nodes exchange messages between each other. Therefore, a publisher node can publish data on a topic and other subscriber nodes can then subscribe from that same topic in order to receive information.

QGroundControl (QGC) is a ground control station which provides full flight control and mission planning for any MAVLink enabled vehicle. It allows us to monitor the vehicle in real-time

by continuously providing data about the position, velocity, orientation of the vehicle and also information about the power and battery consumption. QGC gives a real time map based on the GPS location, and this is important in the project to test the safe landing algorithm. As the main idea behind the safe landing algorithm is to detect flat areas where the vehicle can land safely, the map generated by QGC provided a kind of simulation environment when it comes to ground surface.

As stated before, a Pixhawk flight controller is used to control the air module. Gazebo allows the simulation of drones which are controlled by the PX4 flight control software. The PX4 autopilot communicates with the system through a ROS node called MAVROS, which uses the MAVLink communication protocol. For example, an obstacle avoidance node collects information from the PX4 autopilot regarding the current vehicle state, using the MAVLink protocol. It then sends the data to the flight controller with the computed setpoints, which the vehicle will then follow.

At this point, the whole configuration of PX4 and ROS has been explained. The relationship between each component is shown in Figure 5. It can be seen, PX4 Software in the Loop [10] communicates with the simulator, which is Gazebo, in order to receive sensor data from the simulated world and then send motor and actuator commands. It can also communicate with a Ground Control Station (QGC). The Offboard API represented in Figure 5 is ROS.

As previously stated, Raspberry Pi is used as an onboard computer to control the ground module. Another reason why Pi is used is briefed in this section. Due to the fact that ROS is only supported on Linux, a Virtual Machine with Ubuntu OS is used for simulations. Having installed Ubuntu on the Raspberry Pi, makes the process easier as all the nodes being simulated can directly be implemented on the real vehicle. The only differences being that the actual sensors will be feeding data to the PX4 controller instead of Gazebo sensor models and the motor commands will be computed by the PX4 flight controller.

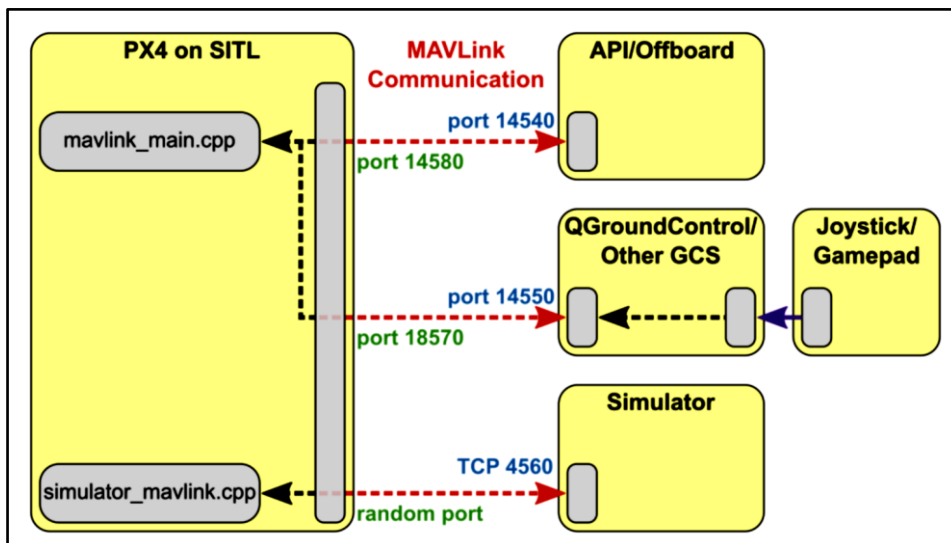


Figure 5 Communication architecture for the vehicle

Flight Review, a web-based tool is used to analyse the results of simulation and tests. A log file is created after each mission and it is retrieved through QGC tool which contains various flight data, battery information and other key information of the completed mission. The log file from the QGC is uploaded in the Flight Review tool and the results are analysed.

3.3 Communication Algorithm

The communication algorithm is part of the onboard computer which focuses on the selection of either the ground module or the air module of the hybrid vehicle. This algorithm is responsible for making decisions on whether the vehicle should use the ground module or the air module for its maneuvering. It is also used to make sure that only one among the two modules is active at a given time during the maneuvering. For the selection of the module, the algorithm depends on the given constraints on power, time and energy. It also depends on the next waypoint available for the vehicle to be followed. For this semester, the decision process of the algorithm is focused on the next waypoint available for the vehicle.

The flowchart of the algorithm can be seen in Figure 6. The decision process depends on the value of the z-coordinate of the next waypoint. If the value is greater than 0.5m, the algorithm switches to air module and reaches the goal point. It switches to the ground module if the value is less than 0.5m. To make sure the algorithm does not run into an infinite loop, a stopping criterion is introduced. When *Ctrl+c* is given as input, the algorithm stops. The algorithm is implemented in ROS using the node *air_ground_hybrid_module.py*.

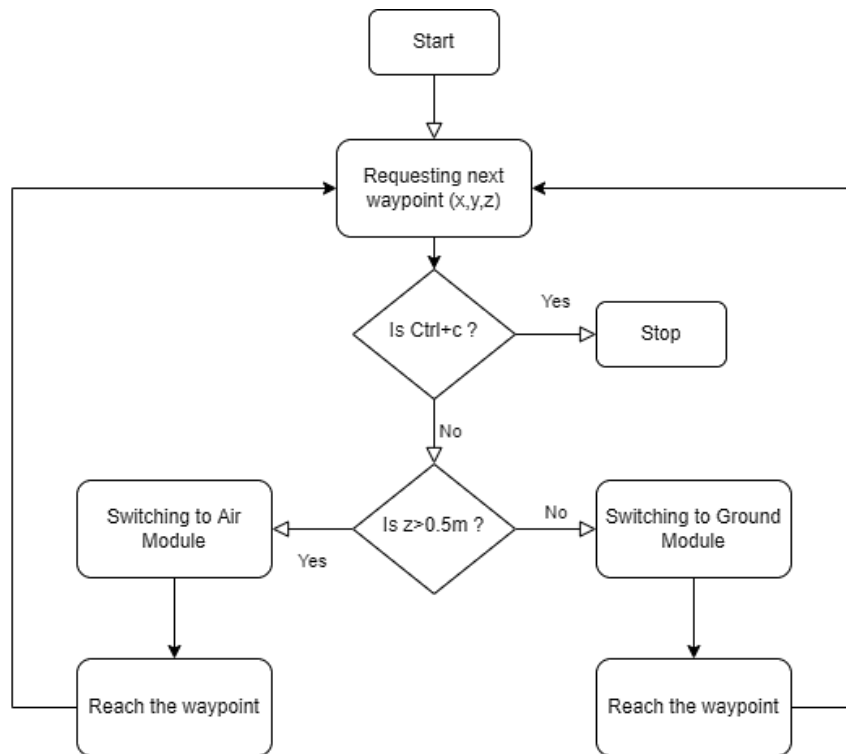


Figure 6 Communication algorithm flowchart

3.4 Simulation Set-up for the Hybrid Vehicle

The hybrid vehicle model for the Gazebo simulation is created in the previous year by Ricardo [2]. It is found during the simulation tests that the model had some vibration and drifting issues. To counter these issues, the weight and inertia values of the vehicle required some modifications. The new weight of the model is 1.95 kg which is close to the real vehicle which is 1.85 kg. The final model of the vehicle is displayed in Figure 7.

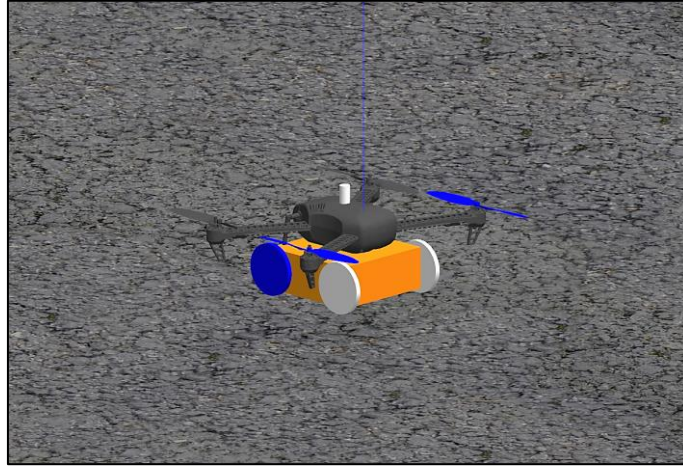


Figure 7 Gazebo Hybrid Model

Once the hybrid model is setup without any issues, the communication algorithm is tested. The launch file `mavros_posix_sitl2.launch` is launched which launches px4, mavros, gazebo and connects to QGC via MAVLink. The vehicle model can be seen in the gazebo environment. In a new terminal, the `air_ground_hybrid_module.py` ROS node is launched which launches the communication algorithm (see Figure 8).

```
karthik@karthik: ~/catkin_ws/src/hybrid_automata/dd_bot
File Edit View Search Terminal Tabs Help
/home/karthik/PX4-Autopilot/launch/mav... x karthik@karthik: ~/catkin_ws/src/hybrid_... x
karthik@karthik:~$ roscd dd_bot
karthik@karthik:~/catkin_ws/src/hybrid_automata/dd_bot$ python air_ground_hybrid
_module.py
Enter the next waypoint (x,y,z):
```

Figure 8 `air_ground_hybrid_module` ROS node launch

The communication between the nodes and rostopics can be observed in the *rqt_graph* as below (see Figure 9).

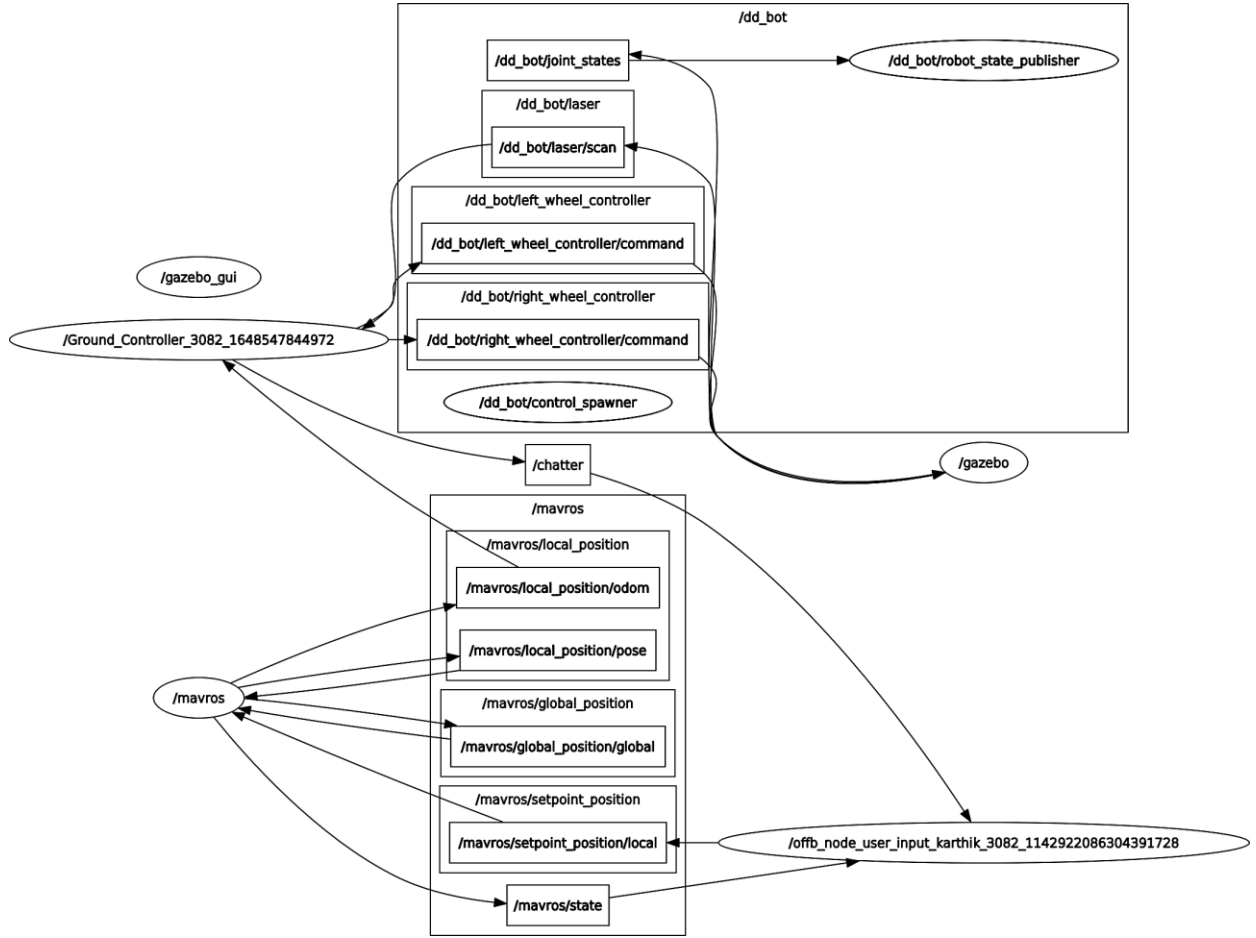


Figure 9 Communication architecture between ROS nodes

3.5 Safe Landing Algorithm

The safe landing algorithm is a part of the path-planning algorithm mainly focusing on the safety part of the vehicle. In this semester, the algorithm has been developed separately by specifying pre-defined waypoints. However, as a future scope, the algorithm shall be integrated with the path-planning module where the waypoints are generated by the latter and are sent to the safe landing algorithm.

Implementation of safe landing algorithm is crucial for the safety of the vehicle as it avoids crashes due to critically low battery and also protects the vehicle from getting damaged while landing due to rough or irregular terrains. The algorithm mainly performs two functions: the energy estimation to reach the next waypoint and the safe landing planner which detects flat areas while landing.

The flowchart which depicts the working of the algorithm is shown in Figure 10,

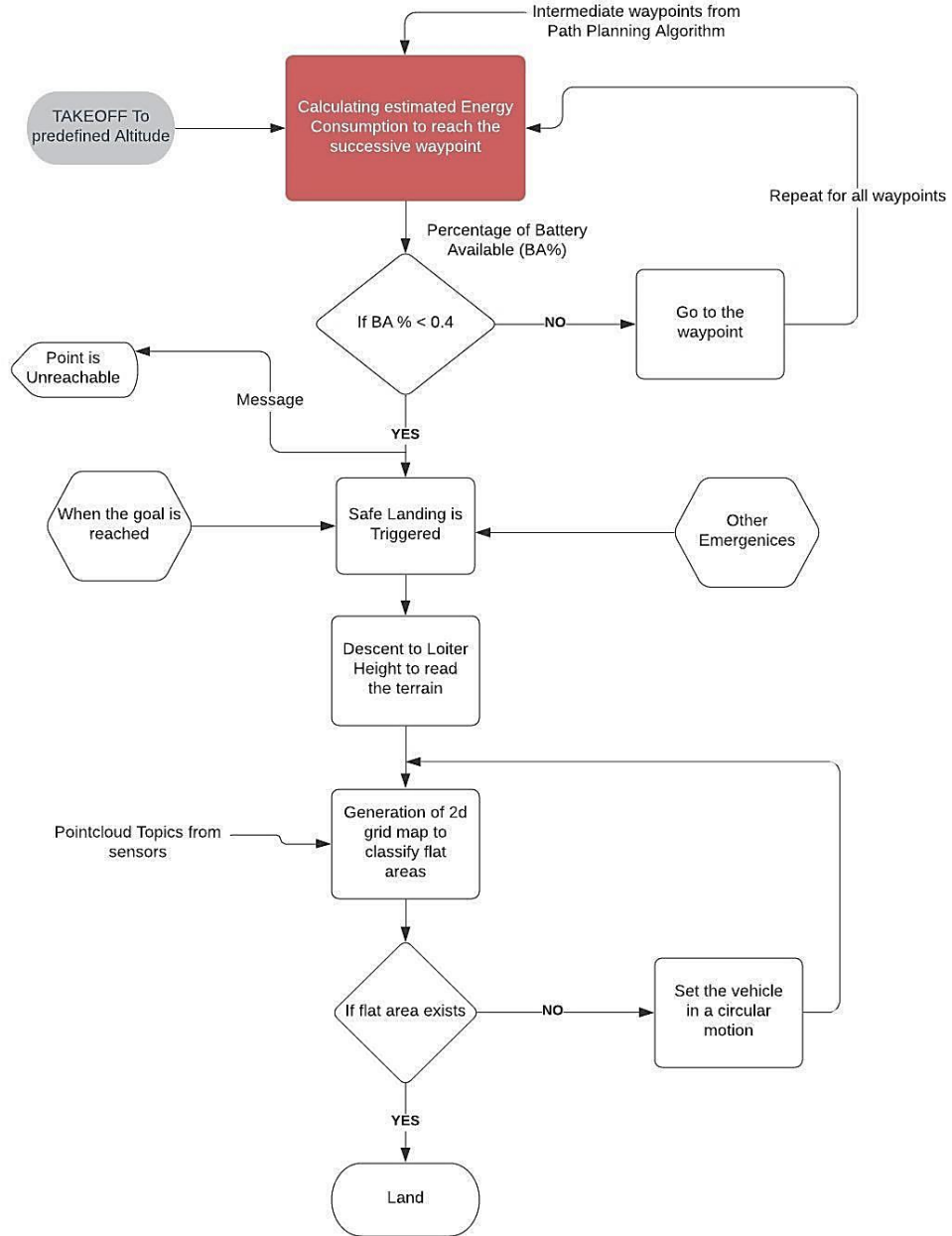


Figure 10 Safe landing algorithm flowchart

3.5.1 Energy Estimation

The first task of the safe landing algorithm once the vehicle has taken off to a specified altitude and waypoints from the path-planning algorithm are received is to estimate the energy in terms of battery capacity. This estimated energy is the energy that will be consumed by the vehicle to reach the next waypoint. Once this estimation is done, the algorithm will then check if the waypoint can be reached with the available battery. If there is sufficient amount of energy remaining, the vehicle will move towards the waypoint or else the safe landing planner will be triggered and the vehicle lands. The working of safe landing planner will be discussed in the next section.

The energy estimation is done based on the current consumption data and the time required to reach the next waypoint from the current position. The current consumption data will be provided by the power control module of the PX4-Autopilot. The time required to reach the waypoint is calculated using the ground speed of the vehicle and the distance to next waypoint. Ground speed is computed by the GPS module mounted on the vehicle and the distance to the waypoint is computed using the shortest distance formula,

$$Distance = \sqrt{((x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2)}$$

Where, (x_0, y_0, z_0) represent the current position of the vehicle and (x_1, y_1, z_1) is the location of the next waypoint.

The computed estimated energy to reach the waypoint is expressed in terms of battery percentage since the mavros node of the PX4-Autopilot publishes battery information in terms of percentage. The battery which we will be using on the vehicle for testing is a 3S 11.1V 3000 mAh LiPo battery with a C-rating of 30. In reality, only 80% of the battery is effective because, if the battery level falls below 20% the battery sustains a permanent damage. Once the energy is expressed in terms of battery percentage, a check is done if the available battery percentage published by the PX4 is higher than the required percentage to go to the next point. If there is enough energy to go to the point, the vehicle will follow the path computed by the path-planning algorithm. If the battery supply is less, then the safe landing planner will take the control over the vehicle and land it safely.

However, in this semester energy estimation has not been performed due to the lack of information about the current consumption data. Accurate information about current consumption can be obtained by performing flight tests and analyzing the flight logs for the same. Once this data is known, an estimate of energy can be made by using the ground speed of the vehicle and distance data as discussed previously.

In this work, the safe landing algorithm is triggered whenever the available battery level falls below 40 % of it's full charge. However, in the future estimate of energy to the successive waypoints can be done by performing the steps mentioned in the previous paragraph.

3.5.2 Safe Landing Planner

As explained in the previous section, the safe landing planner will be triggered whenever the battery level falls below the critical threshold (40%). The safe landing planner takes control over the vehicle everytime the vehicle switches to LAND mode. The pilot can activate the planner manually if there is some erroneous data that is being monitored in real time, which corresponds to component or system failure that is being monitored in real time.

The safe landing planner detects flat terrains by analysing the pointcloud data from the sensors. The vehicle descends to a loiter height, which is 5m for this work. Once it reaches this altitude it studies the terrain underneath the vehicle via the pointcloud data. Pointcloud topics are nothing but the co-ordinates (x, y, z) of the surface being monitored by the downward facing camera sensors. These sensors are available in the Gazebo simulation environment. Once this information is received the planner generates a 2d grid of several bins using the x and y components of the pointcloud data.

For each of the bins present in the grid, the mean and standard deviation of the z component of the pointcloud data are computed. Standard deviation tells us how far the data is spread from the mean value. In this case, since we are computing the deviation and mean of the z values for each bin, it gives an estimation of whether the surface is regular or irregular. If the detected surface is flat, the vehicle will land safely. If an irregular terrain is detected, then the vehicle is set in an outward spiral movement to detect the neighbouring areas and find a suitable spot for landing.

3.5.3 Simulation Set-Up

To run the simulation, in a new Ubuntu terminal a launch file named *safe_landing_offb.launch* (see *Annex 1*) is launched which launches the px4, mavros, establishes connection to the gazebo simulation environment, connects to the ground control station QGC and also starts the *offb_node* and the *Safe_landing_planner_node*. Once the mavros is launched, several mavros topics that are being published by the px4 and topics to which the node which contains the algorithm can subscribe are generated. The launch file also tells ROS to use the iris quadcopter model along with the downward facing camera. It also launches a visualization environment known as Rviz (ROS Visualization) where the behaviour of the vehicle in a simulated environment can be visualized.

In *offb_node*, a publisher is created in order to publish the desired position setpoints, as well as clients which request arming of the vehicle and mode change to Offboard mode. **The difference between clients and publishers is that clients send some data but also expect a response message which in this case is the current state of the vehicle.** It should be noted that the Offboard mode serves for the case when setpoints are being provided by MAVROS running on a companion computer (RPI), which is our case.

PX4 has a timeout of 500ms between successive Offboard commands. If this timeout is exceeded, the autopilot will enter Failsafe mode, which lands the vehicle xy position. Hence, the publishing rate needs to be faster than 2 Hz, which already accounts for possible latencies. Before publishing anything, the *offb_node* waits for the connection to be established between MAVROS and the autopilot. Then, a set of setpoint commands are sent to the vehicle prior to engaging the

Offboard mode. In the code, there is a switch to Offboard mode, after which the vehicle is armed and ready to fly. The QGC display shows (see Figure 11) the flight mode switched to Offboard mode and the vehicle is flying (vehicle is armed the motors are spinning).

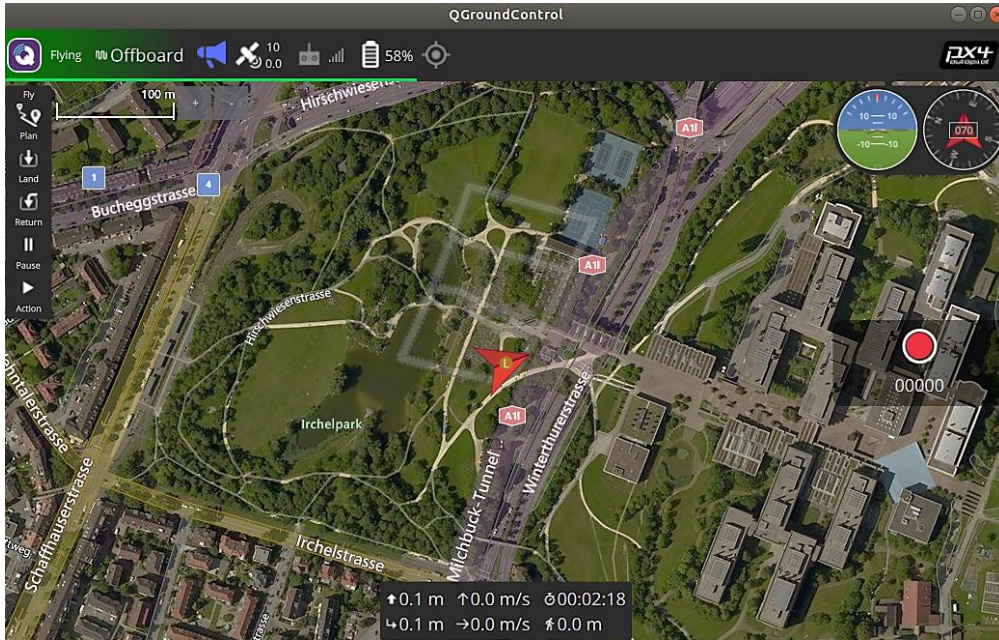


Figure 11 Vehicle switched to offboard mode in QGC

Once this is achieved in a new terminal the *pos_teleop_node* is implemented by using the `roslaunch` command. The *pos_teleop_node* is the one which asks the user for the waypoints. Once the user inputs the desired values, a publisher topics called `/chatter` is generated by this node which publishes the waypoints given by the user. This topic is the subscribed by the *offb_node* which is already running. The *offb_node* subscribed to the `/chatter` topic then publishes onto the mavros topic `/mavros/setpoint_position/local` which causes the vehicle to move to the desired position. The communication between these nodes can be seen in figure shown below. The incoming arrows to a node represents a topic that is subscribed by that node and the outgoing arrows from a node denote a topic that is being published by that node.

A condition is set in the *offb_node* telling the vehicle to land if battery information published by the topic `/mavros/battery` falls below 40%. Once this condition is satisfied the *safe_landing_planner_node* performs the functions mentioned in section 3.5.2. It can be seen from the Figure 12 that this node is subscribed to the topic `/camera/depth/points` being published by the *gazebo* node. This topic contains the point-cloud data as discussed previously.

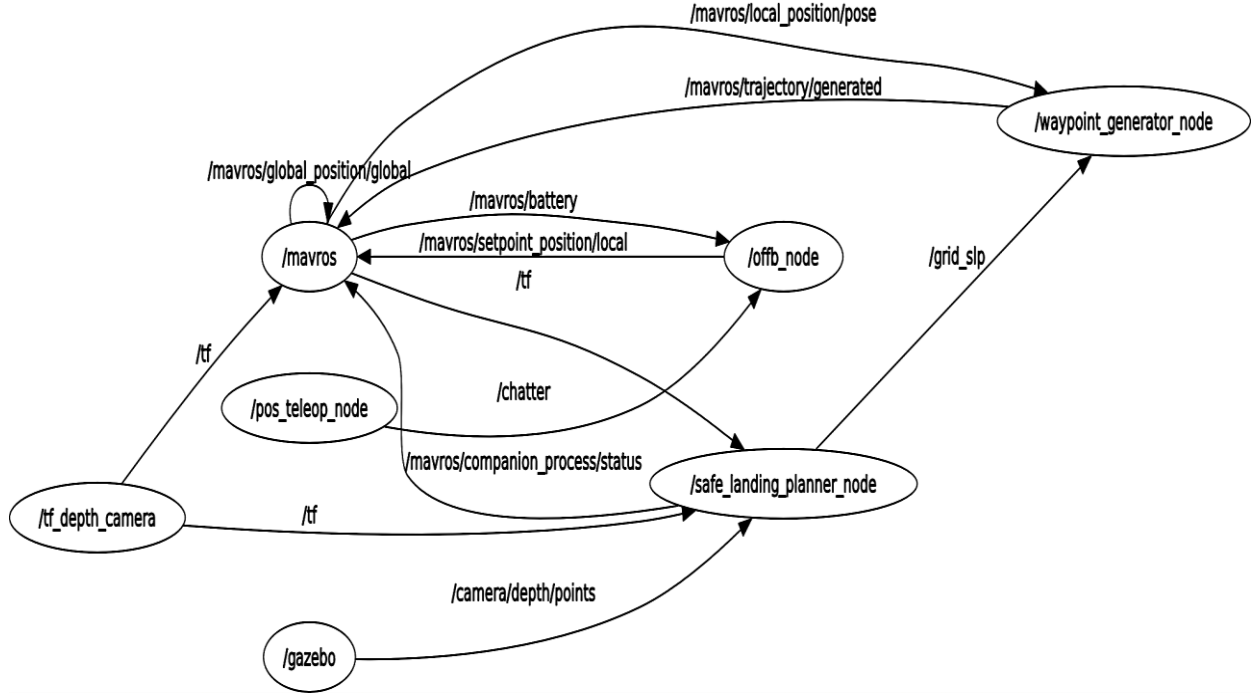


Figure 12 Communication between ROS nodes

Once the point-cloud data is received, the *safe_landing_planner* node publishes a topic named */grid_slp* which contains the information about the 2d-grid that is generated based on the point-cloud data. This topic is subscribed by the */waypoint_generator_node* which analyses the generated 2d-grid and classifies the terrain as flat or irregular. If the terrain is irregular then this node computes waypoints such that the vehicle moves in an outward spiral motion and detects areas which are flat for the vehicle to land safely.

The safe landing planner is simulated using QGC as it contains predefined maps. A mission is defined by specifying waypoints using the */pos_teleop_node*. Rviz which is another simulator compatible with ROS is used to observe the behaviour of the vehicle. It provides a view of your robot model, capture sensor information from robot sensors, and replay captured data. It can display data from camera, lasers, from 3D and 2D devices including pictures and point clouds.

The default battery model used by QGC and Gazebo only deplete to 50% of its capacity by default, thus being implemented to never run out of energy. The parameters of the battery that is being used in the hybrid vehicle as mentioned in section 3.1.1 are defined in QGC (see Figure 13) to have better and realistic results.

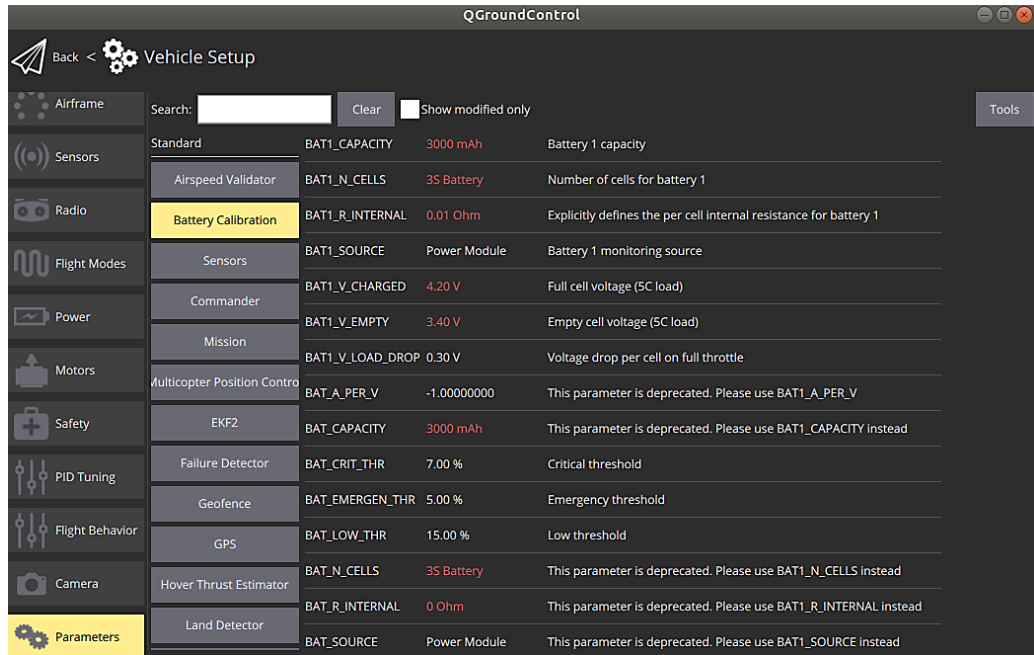


Figure 13 Battery Calibration in QGC

3.6 Real-time Vehicle Setup

3.6.1 Pixhawk and Raspberry Pi connection

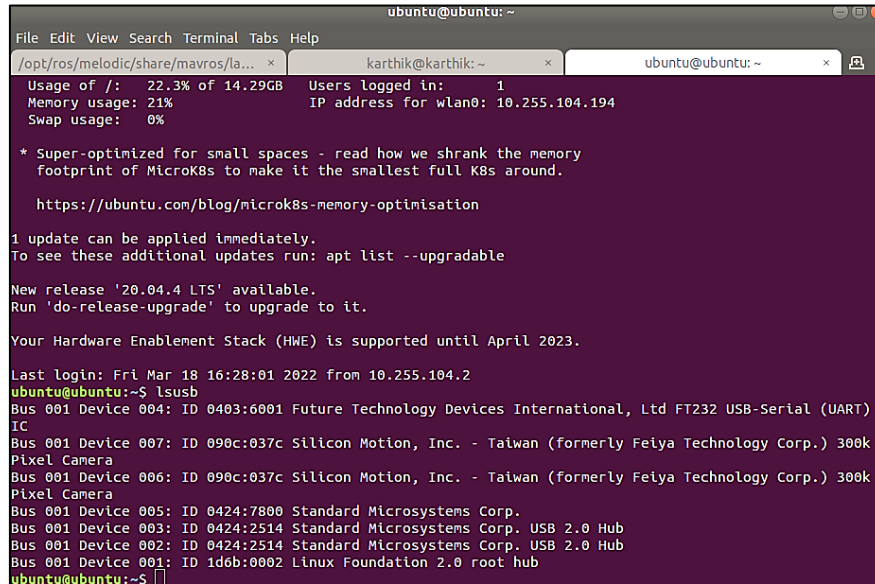
The MAVLink connection setup between the onboard computer (Raspberry Pi) and the flight controller (Pixhawk) is previously done by Ricardo [2]. In this setup, the GPIO pins on the Pi are used for the communication. While running tests with the vehicle, it has been found that the connection to the flight controller is not reliable, keeps disconnecting, and also it does not have surge protection. The better alternate connection and its setup using FTDI Chip USB-to-serial adapter is recommended in [5]. The hardware connection is shown in Figure 14. The Telem 2 port of the Pixhawk is connected to the FTDI cable. And the FTDI cable is connected to Raspberry Pi through USB.

TELEM2		FTDI	
1	+5V (red)		DO NOT CONNECT!
2	Tx (out)	5	FTDI RX (yellow) (in)
3	Rx (in)	4	FTDI TX (orange) (out)
4	CTS (in)	6	FTDI RTS (green) (out)
5	RTS (out)	2	FTDI CTS (brown) (in)
6	GND	1	FTDI GND (black)

Figure 14 Pixhawk to FTDI cable connection

For the software setup in onboard computer, the default name of a USB FTDI would be like `/dev/ttyUSB0`. If the system has more than one FTDI cable, it will register as `/dev/ttyUSB1`. To avoid confusion, a symlink from `ttyUSB0` to a friendly name is created, which depends on ID of vendor and product.

Using `lsusb` the vendor and product IDs are retrieved as displayed in Figure 15.



```

ubuntu@ubuntu: ~
File Edit View Search Terminal Tabs Help
/opt/ros/melodic/share/mavros/la... x karthik@karthik: ~ ubuntu@ubuntu: ~
Usage of /: 22.3% of 14.29GB Users logged in: 1
Memory usage: 21% IP address for wlan0: 10.255.104.194
Swap usage: 0%

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.

Last login: Fri Mar 18 16:28:01 2022 from 10.255.104.2
ubuntu@ubuntu:~$ lsusb
Bus 001 Device 004: ID 0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART)
IC
Bus 001 Device 007: ID 090c:037c Silicon Motion, Inc. - Taiwan (formerly Feliya Technology Corp.) 300k
Pixel Camera
Bus 001 Device 006: ID 090c:037c Silicon Motion, Inc. - Taiwan (formerly Feliya Technology Corp.) 300k
Pixel Camera
Bus 001 Device 005: ID 0424:7800 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
ubuntu@ubuntu:~$

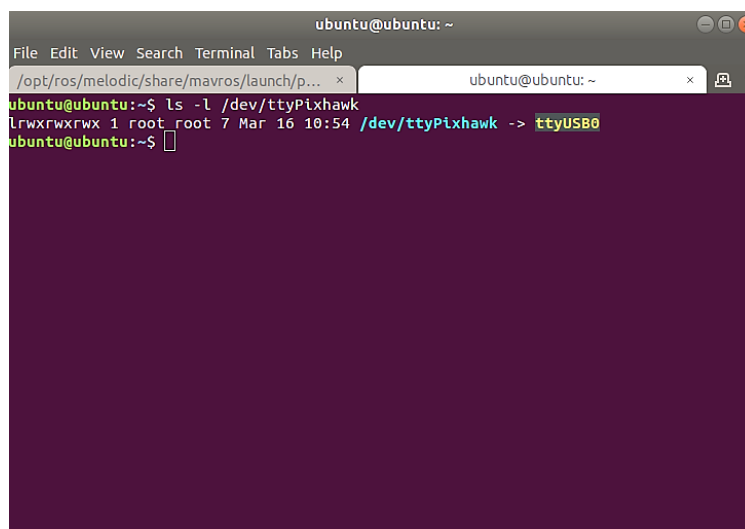
```

Figure 15 Vendor and Product ID information

The ID corresponding to the FTDI cable is “ID 0403:6001”. A new UDEV rule file is created in the directory `/etc/udev/rules.d/99-pixhawk.rules` with following information.

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", SYMLINK+="ttyPixhawk"
```

After adding the above file, a system reboot is performed. Then the connection can be used through serial port `/dev/ttyPixhawk` in the scripts as displayed in Figure 16.



```

ubuntu@ubuntu: ~
File Edit View Search Terminal Tabs Help
/opt/ros/melodic/share/mavros/launch/p... x ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ls -l /dev/ttyPixhawk
lrwxrwxrwx 1 root root 7 Mar 16 10:54 /dev/ttyPixhawk -> ttyUSB0
ubuntu@ubuntu:~$

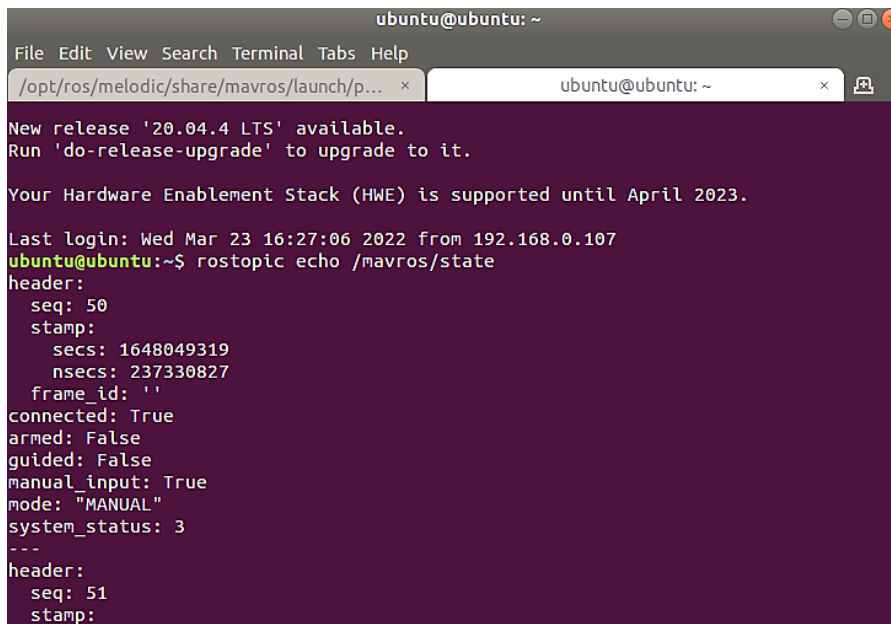
```

Figure 16 `/dev/ttyPixhawk` Serial Port Access

The user of the system is added to *tty* and *dialout* groups with following code to avoid executing the scripts as *root*.

```
usermod -a -G tty ros-user
usermod -a -G dialout ros-user
```

The connection is validated by checking mavros connection state of the vehicle. The `rostopic /mavros/state` gives information of the connection to the flight controller as shown in Figure 17.

A terminal window titled 'ubuntu@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). It shows system updates and a terminal command. The command 'rostopic echo /mavros/state' has been executed, resulting in a JSON message. The message indicates the vehicle is connected, not armed, in manual mode, and has a system status of 3. The message is displayed twice, separated by a dashed line.

```
ubuntu@ubuntu: ~
File Edit View Search Terminal Tabs Help
/opt/ros/melodic/share/mavros/launch/p... x ubuntu@ubuntu: ~ x
New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.

Last login: Wed Mar 23 16:27:06 2022 from 192.168.0.107
ubuntu@ubuntu:~$ rostopic echo /mavros/state
header:
  seq: 50
  stamp:
    secs: 1648049319
    nsecs: 237330827
  frame_id: ''
connected: True
armed: False
guided: False
manual_input: True
mode: "MANUAL"
system_status: 3
---
header:
  seq: 51
  stamp:
```

Figure 17 Message published on topic `/mavros/state`

3.6.2 Offboard Mode Vehicle Setup

Offboard mode is where the Pixhawk receives stream of setpoints for controlling vehicle attitude and movement through Raspberry Pi. The offboard mode supports only a set of MAVLink connections.

There are certain pre-conditions for activating the offboard mode.

1. The vehicle should receive a stream of setpoints before engaging this mode.
2. A feedback from external position sensor (for example, GPS and motion capture/MoCap system) is required for the activation of this mode.
3. The vehicle should continue to receive the setpoints at rate greater than 2 Hz to remain in the offboard mode.
4. Offboard mode requires an active connection to a remote MAVLink system (here, the onboard computer), if the connection is lost, the failsafe module is activated.

While performing tests, it is observed that the pre-flight conditions are violated due to a high horizontal position error of the GPS due to the cluttered indoor environment. Later it is found that the vehicle needs minimum of 12 GPS satellite locks when it is using GPS module as external position sensor. This requirement can only be satisfied while testing the vehicle in outdoors.

Alternatively, motion capture (MoCap) system is used as external position sensor for the real-time position feedback to the vehicle.

3.6.3 MoCap System Configuration for Position Estimation

MoCap is a 3D position and orientation estimation technique which uses external fixed cameras to track the vehicle in a given 3D space. The vehicle is tested for engaging offboard mode using the MoCap system facility available at ISAE-SUPAERO's Drone Lab.

To use MoCap system, certain parameters of the position estimation in the flight controller are modified as given in [5]. The Table 1 below gives depicts the modified parameters.

Table 1 Modified PX4 parameters for MoCap system

Parameter	Setting for External Position Estimation
EKF2_AID_MASK	Set to <i>vision position fusion and vision yaw fusion</i>
EKF2_HGT_MODE	Set to <i>vision</i>
EKF2_EV_DELAY	40 ms

After changing the above parameters, the vehicle reboot is performed. Now the vehicle uses feedback from MoCap system for offboard mode. The data from MoCap system is relayed to Pixhawk through MAVROS using the rostopic */mavros/vision_pose/pose*.

4 Results and analysis

This section talks about the results obtained from the simulation tests performed for the communication and safe-landing algorithms. The results from the real-time test carried on the hybrid vehicle to validate the communication set-up and offboard control are also presented.

4.1 Simulation Results for Communication Model

As mentioned in section 3.4, The *air_ground_hybrid_module* node requests for the next waypoint of the vehicle. The waypoint is given as input in the format (x,y,z). Depending on the condition of z-coordinate mentioned in section 3.4, the vehicle switches to either the ground module or the air module. The switching of the model is displayed in Figure 18 and Figure 19.

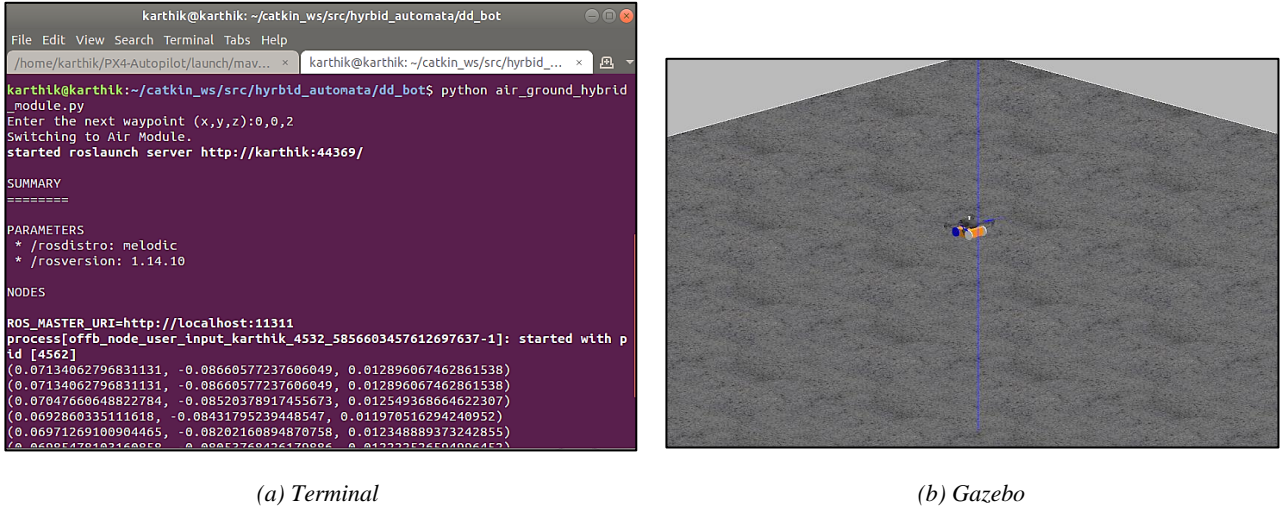


Figure 18 Vehicle switching to Air Module

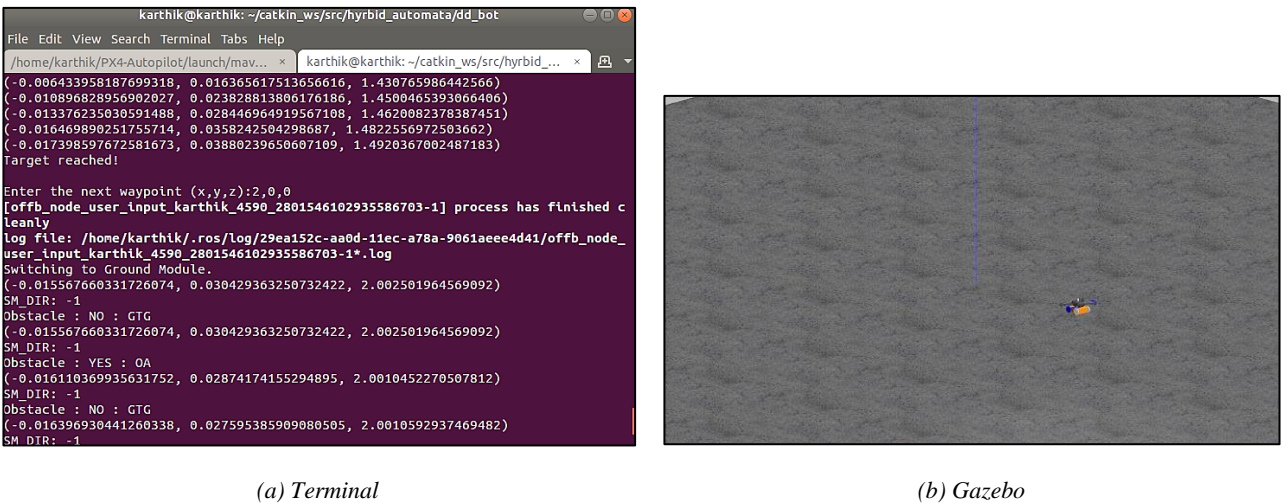


Figure 19 Vehicle switching to Ground Module

In Figure 18(a), (0, 0, 2) is given as the waypoint input. Here, the z-coordinate is greater than 0.5m and the vehicle switches to air module. The vehicle reaching the given waypoint is visualized in Gazebo as shown in Figure 18(b). As the vehicle reaches the target, the *air_ground_hybrid_module* node again requests for the next waypoint as shown in Figure 19(a).

(2, 0, 0) is given as new input. Then the vehicle switches to ground module as the z-coordinate is less than 0.5m, and reaches the target on ground (see Figure 19(b)). To terminate the communication node, *Ctrl+c* command is given as input when the next waypoint is requested.

The vehicle incorporates PID controller in both ground and air module to have a better performance in terms of responsiveness, reduced oscillations and negligible steady state error. The ground module controller and obstacle avoidance algorithm is explained in [2]. The PID control architecture depicted in Figure 20 for air module is in [5]. As the hybrid model of the vehicle is modified from [2], the air module is tuned for the quadcopter using the tuning set-up available in QGC. The tuning set-up of QGC is displayed in Figure 21.

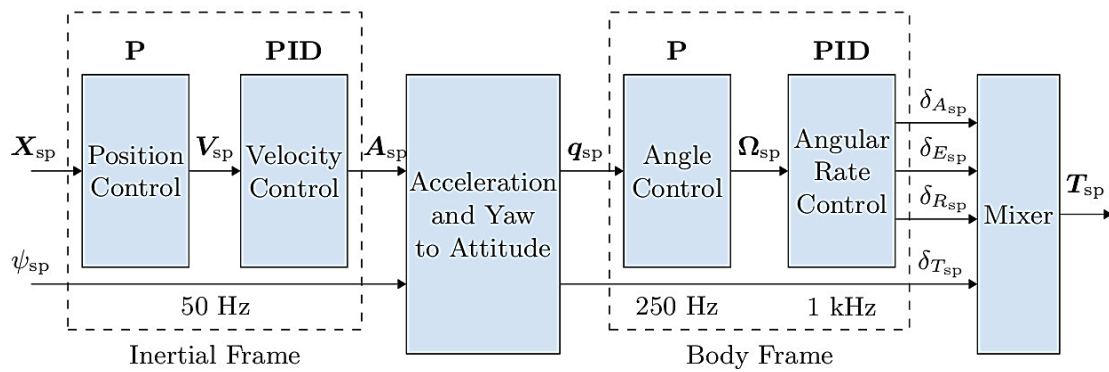


Figure 20 PID Control Architecture for Air Module

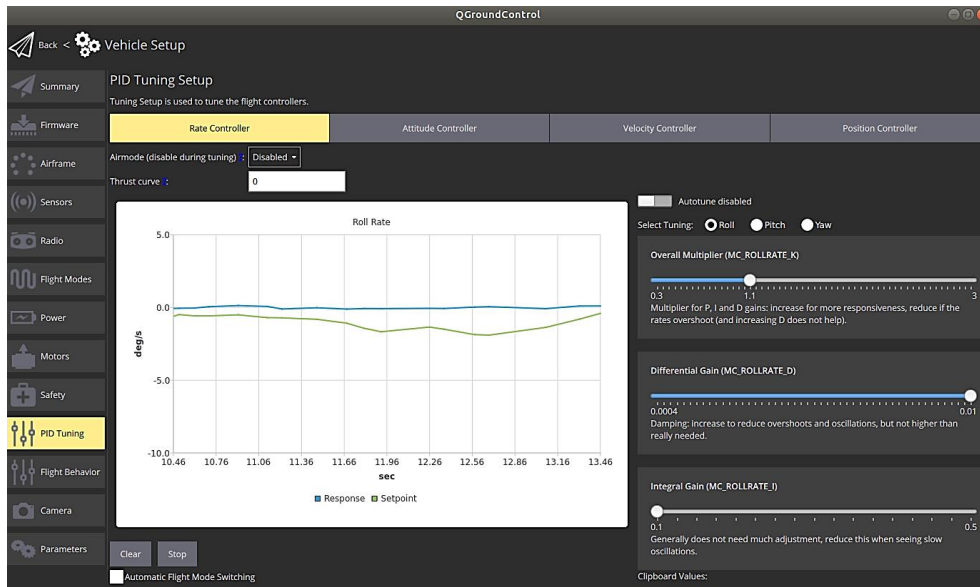


Figure 21 PID tuning set-up in QGC

The tuned values of PID controller of the hybrid vehicle model for better performance are listed in Table 2, Table 3, Table 4, and Table 5.

Rate Controller

Table 2 PID values for rate controller for hybrid model

Parameters	Tuned Values
MC_ROLLRATE_P	2.1
MC_ROLLRATE_I	0.2
MC_ROLLRATE_D	0.01
MC_PITCHRATE_P	2.4
MC_PITCHRATE_I	0.2
MC_PITCHRATE_D	0.01
MC_YAWRATE_P	0.8
MC_YAWRATE_I	0.16

Attitude Controller

Table 3 PID values of attitude controller for hybrid model

Parameters	Tuned Values
MC_ROLL_P	5.5
MC_PITCH_P	4.5
MC_YAW_P	1.4

Velocity Controller

Table 4 PID values for velocity controller for hybrid model

Parameters	Tuned Values
Horizontal Velocity	
MPC_XY_P	4
MPC_XY_I	3.4
MPC_XY_D	0.6
Vertical Velocity	
MPC_XY_P	6
MPC_XY_I	1.05
MPC_XY_D	0.45

Position Controller

Table 5 PID values for position controller for hybrid model

Parameters	Tuned Values
Horizontal Position	
MPC_Z_P	1.05
Vertical Position	
MPC_Z_P	1.15

These tuned values of PID gains for the controller are then updated to the simulation set-up and the simulation is run. For a realistic simulation, Gaussian noise is also introduced in the sensors. Using the flight log data from Pixhawk and Pixhawk flight review tool, the thrust, yaw angle, and position response in x, y and z directions of the vehicle are plotted as shown in following figures.

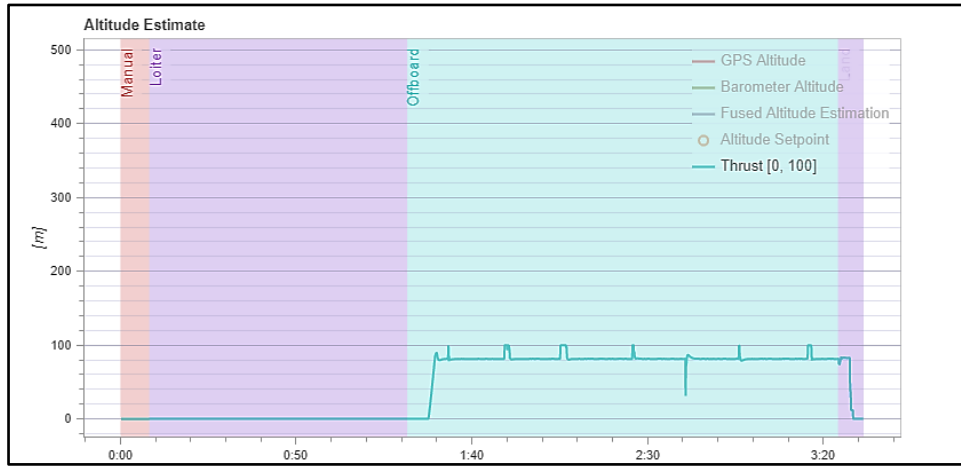


Figure 22 Thrust plot for Hybrid Model Simulation

In Figure 22, the thrust used by air module is depicted. It is observed that the nominal thrust required for hovering is around 80%. The reason for the high nominal thrust is that the vehicle is too heavy. It is also observed that during maneuvering, the thrust saturates which results in poor performance from the tuned controllers.

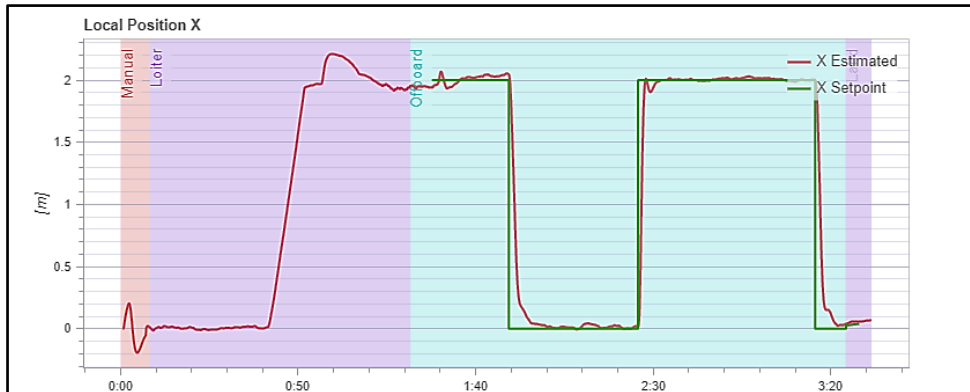


Figure 23 Position X plot for Hybrid Model Simulation

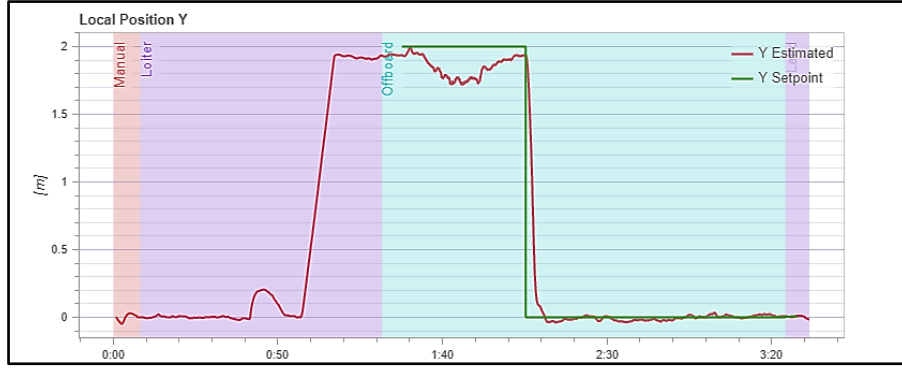


Figure 24 Position Y plot for Hybrid Model Simulation

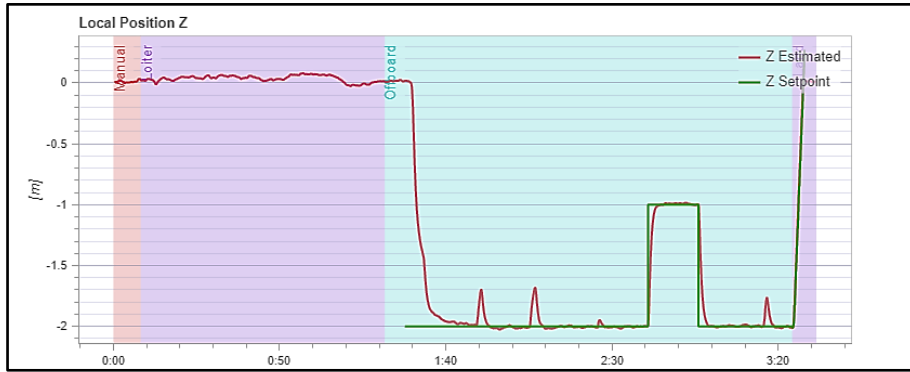


Figure 25 Position Z plot for Hybrid Model Simulation

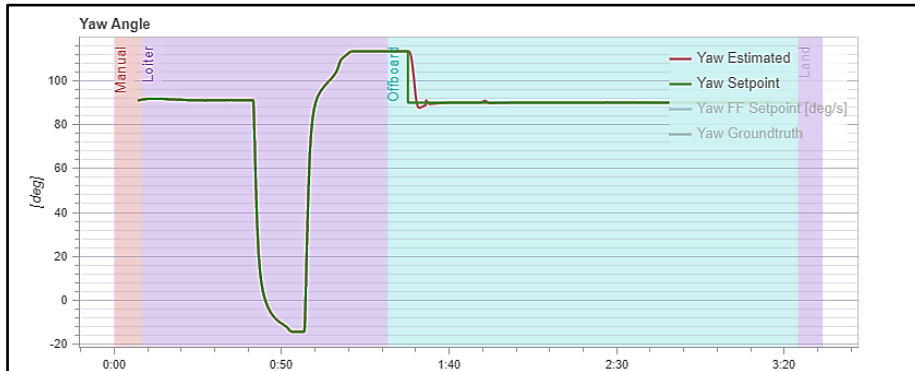


Figure 26 Yaw Angle plot for Hybrid Model Simulation

From the position and yaw angle plots in Figure 23, Figure 24, Figure 25, and Figure 26, taking into consideration the saturation of the thrust, the previously mentioned PID gains results in acceptable responses. The flight controller uses extended Kalman filter for the estimation of the position through noisy sensor measurement data. In the plots, the Loiter region represents the movement of the vehicle using ground module and the Offboard region represent the movement of the vehicle using air module. The disturbance observed in Position Y plot (Figure 24) at the beginning of the Offboard region is due to the correction of yaw during switching of the vehicle to air module. The air module supports a fixed heading manoeuvre, and the vehicle will correct itself to that heading when the air module is selected. The small disturbances observed in Position Z plot (Figure 25) are due to the saturation of the thrust.

4.2 Simulation Results for Safe Landing Algorithm

As mentioned in section 3.5.3, the *pos_teleop_node* is used to ask the user the waypoints for the mission. A set of waypoints are given as input to the safe landing planner and the mission is performed. Validity of the algorithm is tested by using the ground control station tool QGC which contains a database of pre-defined maps. The camera sensor attached to the quadcopter in gazebo environment then uses the QGC map to get pointcloud data and feed it to the safe landing algorithm. Rviz is used to visualize the behaviour of the drone and to check if the algorithm is working as expected, that is to say if it is able to switch to land mode when the battery falls below 40%. This tool also helps us to verify if the algorithm is detecting the terrain underneath the quadcopter during landing and making decisions accordingly. Gazebo is used to see the vehicle in the QGC world. The Figure 27 below shows the quadcopter flying in a gazebo world pre-defined in QGC.



Figure 27 Quadcopter flying in Gazebo world

The behaviour of the vehicle as seen in Rviz is shown in figure. As soon as the battery level in the vehicle falls below 40%, the algorithm decides the vehicle must land and it begins its descent (see Figure 28(a)). The vehicle descends to a height of 5m known as loiter height. At this level, the vehicle hovers and scans the ground underneath it by analysing the point-cloud data coming from the camera sensors. It can be seen from Figure 28(b) that at a particular waypoint in the QGC world, represented as a yellow sphere, the algorithm has classified the terrain as irregular or not suitable for landing. This is shown by a red colored 2d-grid. Next, the algorithm is set into a spiral outward motion and the vehicle moves to another waypoint computed by the algorithm as show in Figure 28(c). At this point, again the vehicle scans the ground and in this case the algorithm has deduces the terrain is flat. This is shown by the green colored 2d-grid in Figure 28(c). Once the vehicle knows the ground is suitable for landing, it begins its descent and this can be seen in Figure 28(d).

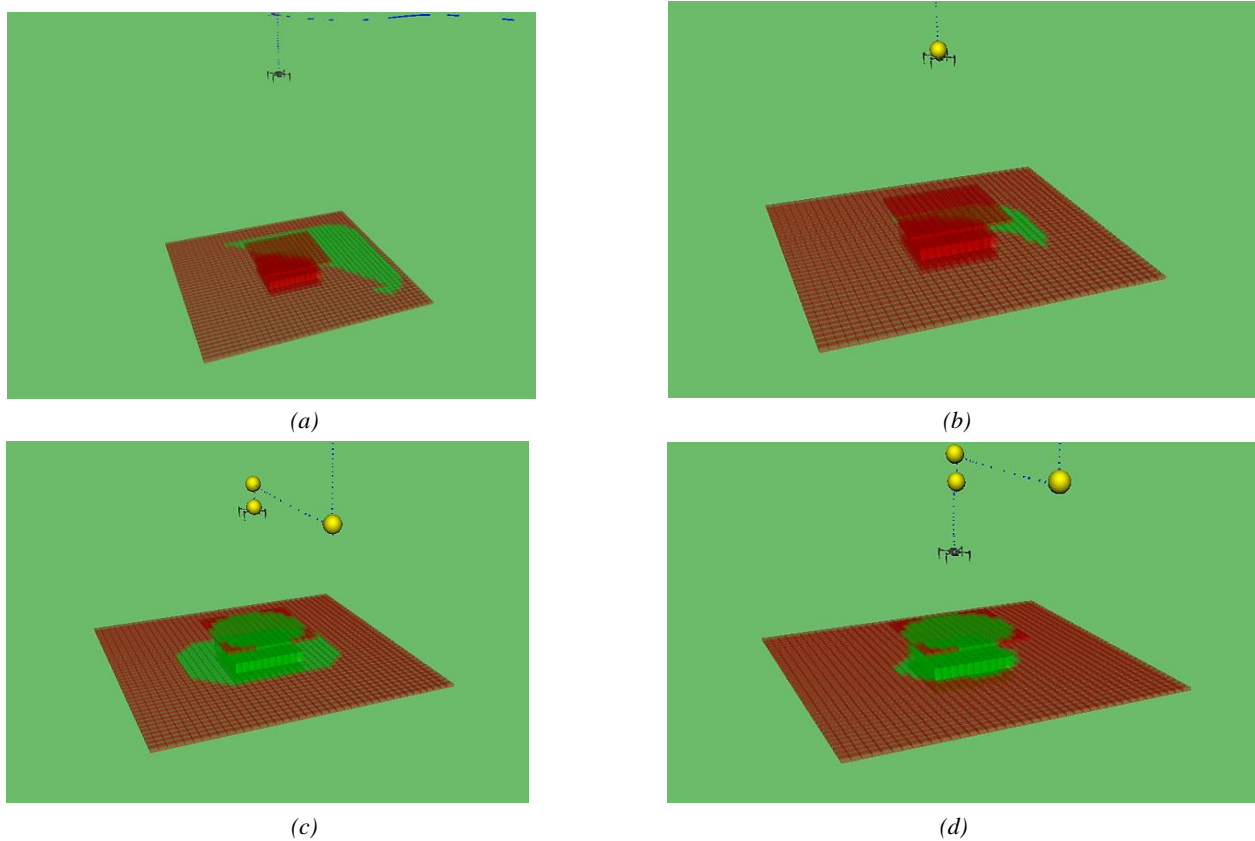


Figure 28 Validation of safe landing algorithm in Rviz

In order to have a better performance of the vehicle in terms of responsiveness, reduced oscillations and negligible steady state error, PID tuning for the quadcopter was performed using the tuning set-up available in QGC. The control architecture used in Pixhawk is discussed in the previous section. The tuned values of PID in order to achieve faster response, less overshoot and small steady state error are given in Table 6 , Table 7, Table 8 and Table 9.

Rate Controller

Table 6 PID values for rate controller

Parameters	Tuned Values
MC_ROLLRATE_P	1.1
MC_ROLLRATE_I	0.125
MC_ROLLRATE_D	0.01
MC_PITCHRATE_P	1.1
MC_PITCHRATE_I	0.125
MC_PITCHRATE_D	0.01
MC_YAWRATE_P	1.05
MC_YAWRATE_I	0.08

Attitude Controller

Table 7 PID values of attitude controller

Parameters	Tuned Values
MC_ROLL_P	6.5
MC_PITCH_P	7
MC_YAW_P	3

Velocity Controller

Table 8 PID values for velocity controller

Parameters	Tuned Values
Horizontal Velocity	
MPC_XY_P	3.65
MPC_XY_I	5.6
MPC_XY_D	0.6
Vertical Velocity	
MPC_XY_P	4
MPC_XY_I	1.85
MPC_XY_D	0.25

Position Controller

Table 9 PID values for position controller

Parameters	Tuned Values
Horizontal Position	
MPC_Z_P	0.9
Vertical Position	
MPC_Z_P	1.65

These tuned values of PID gains for the controller are then updated to the simulation set-up and the simulation is run. The position response of the vehicle in x, y and z directions are shown in Figure 29, Figure 30 and Figure 31.

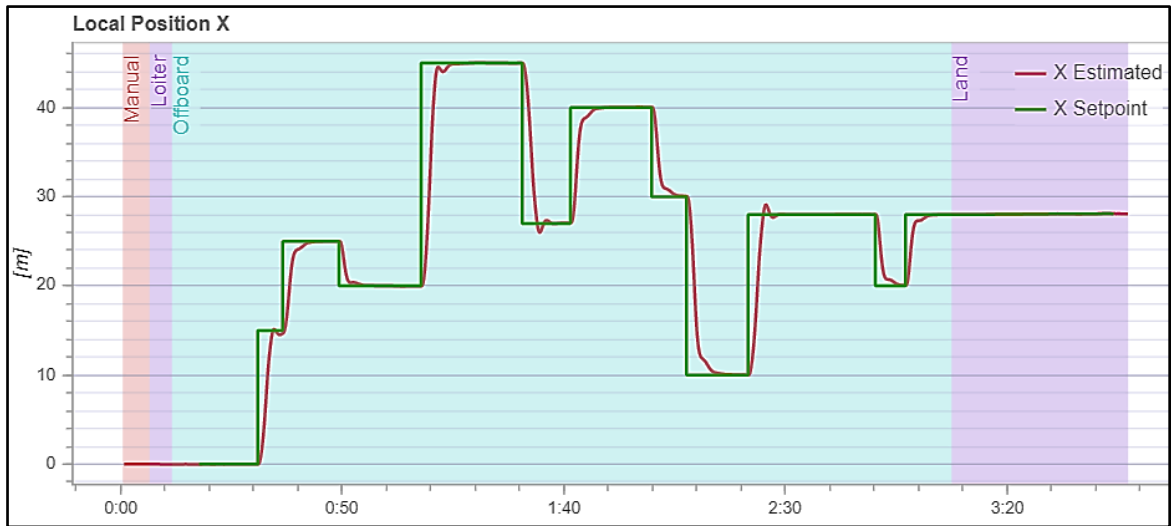


Figure 29 X position plot for safe landing simulation

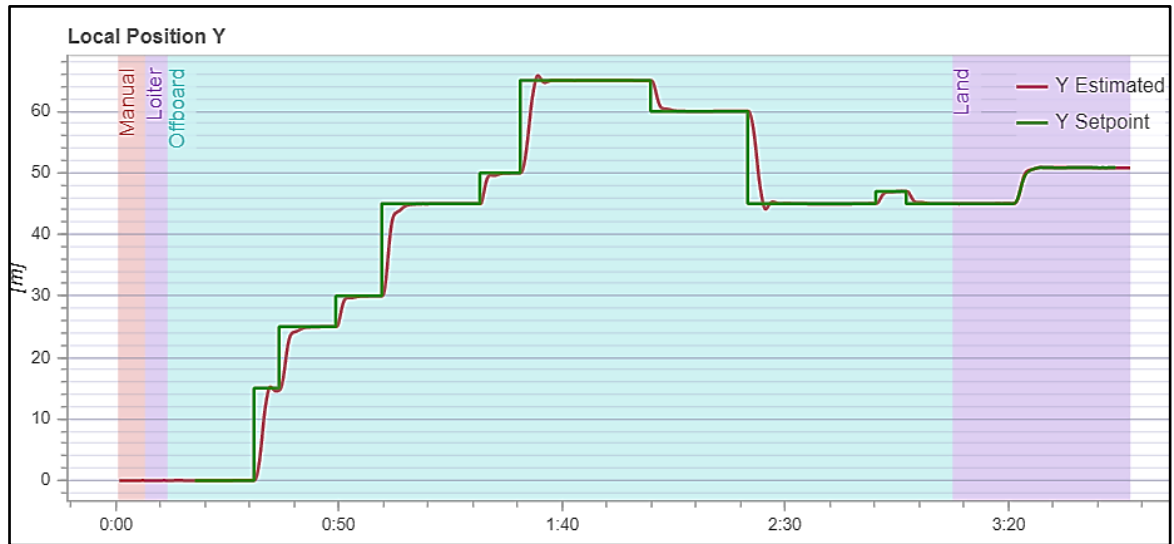


Figure 30 Y position plot for safe landing simulation

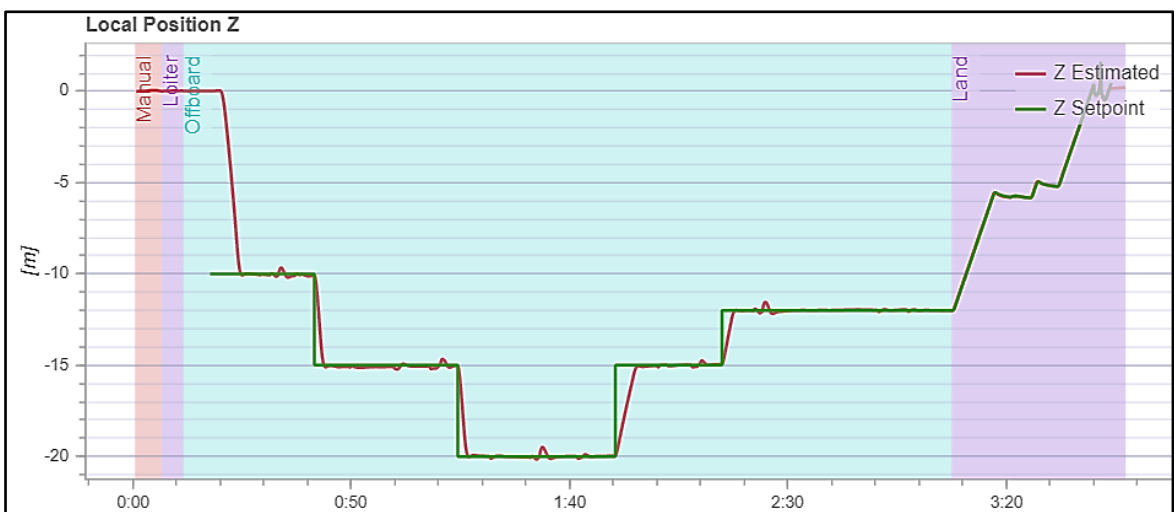


Figure 31 Z position plot for safe landing simulation

It can be seen from the above plots that the PID values mentioned previously results in an acceptable performance of the vehicle. In the offboard mode, the vehicle follows the waypoint inputs given by the user. After around three minutes of flying in offboard mode, the vehicle switches to land mode and the safe landing algorithm take control of the vehicle. The Figure 32 shows the battery percentage over the entire duration of flight. The battery falls below the critical threshold of 40% at around 3 minutes.

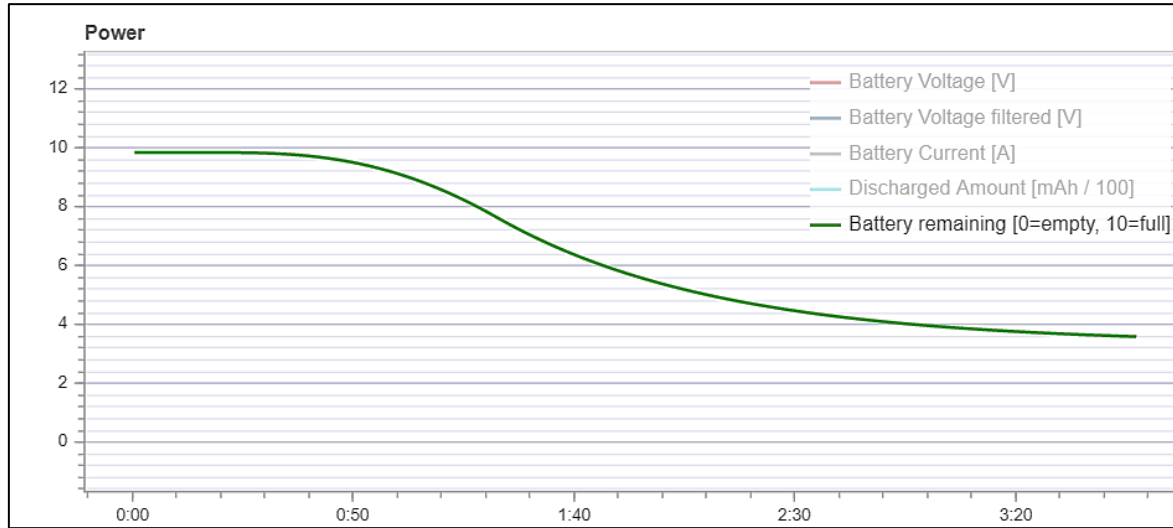


Figure 32 Battery percentage variation during the simulation

4.3 Testing of Hybrid Vehicle

The connection between the Raspberry Pi and Pixhawk is tested with the real vehicle in the Drone Lab. Two flight tests are performed and they are focused on testing the controller of the air module. **The default PID gains are used during the first flight test.** The aim of this test is to make the vehicle enter the Offboard mode, take-off and hold the position at (0, 0, 1).

Channel 5 on remote controller is configured to change the vehicle mode to Offboard mode and Manual mode. The vehicle is positioned in the flying zone of the lab (see Figure 33).

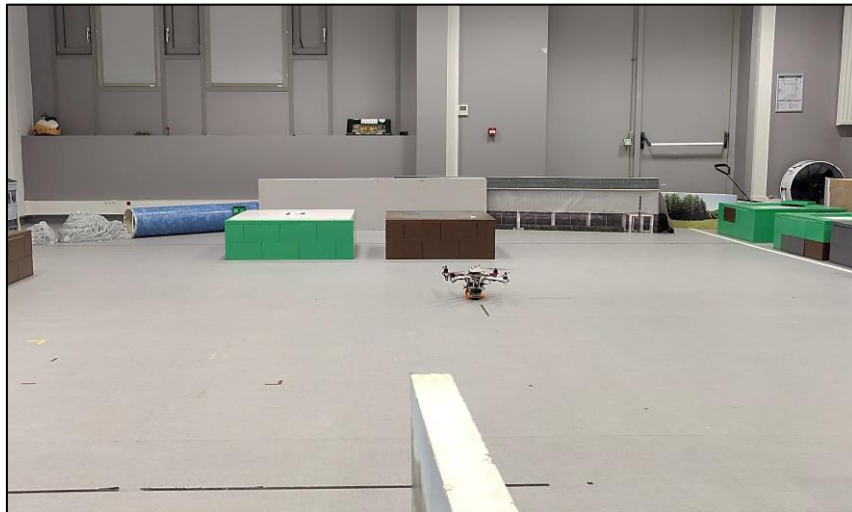


Figure 33 Vehicle Position in Flying Zone

The vehicle is armed manually and taken-off the ground. It is observed that, around 80% of thrust is utilized by the vehicle for hovering. Raspberry Pi is connected through SSH on a remote PC. The Pi is connected to Pixhawk through MAVROS. In a new terminal, the setpoints are published using the *offb_node*. Then the vehicle is switched to Offboard through remote controller.

It is observed that before reaching the given height ($z = 1\text{m}$), the thrust value applied to vehicle in Offboard mode reaches 100% and saturates. The thrust plot during the flight is displayed in Figure 34.



Figure 34 Thrust plot for First Test Flight

In Figure 35, position z plot is displayed. It is observed that due to the saturation in thrust, the vehicle is oscillating around the given setpoint and the controllers are unable to provide the required control input to hold the vehicle at the required setpoint.

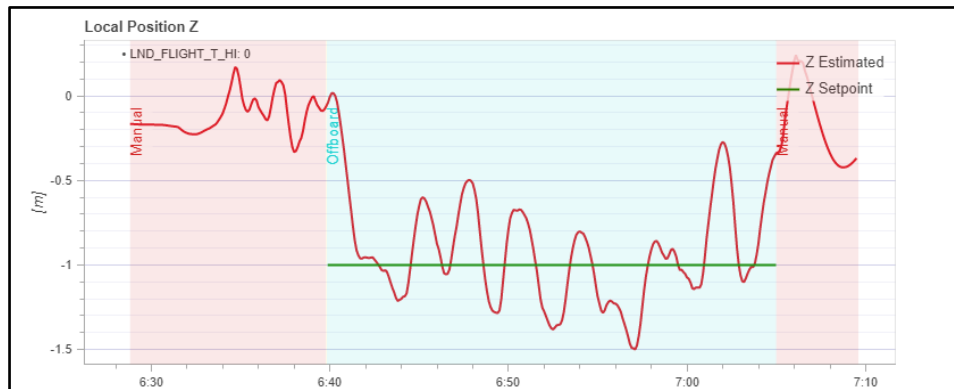


Figure 35 Position Z plot for First Test Flight

As the nominal hover thrust of the vehicle is around 80%, the power drawn from the battery is too high. As a result, the voltage of the battery is reduced drastically during the flight. The same can be observed in Figure 36. Initially, the battery is at 11.4V and during the flight the voltage is dropped to around 10V within 30 seconds. This results in very less endurance of the battery and the flight needs to be terminated early.

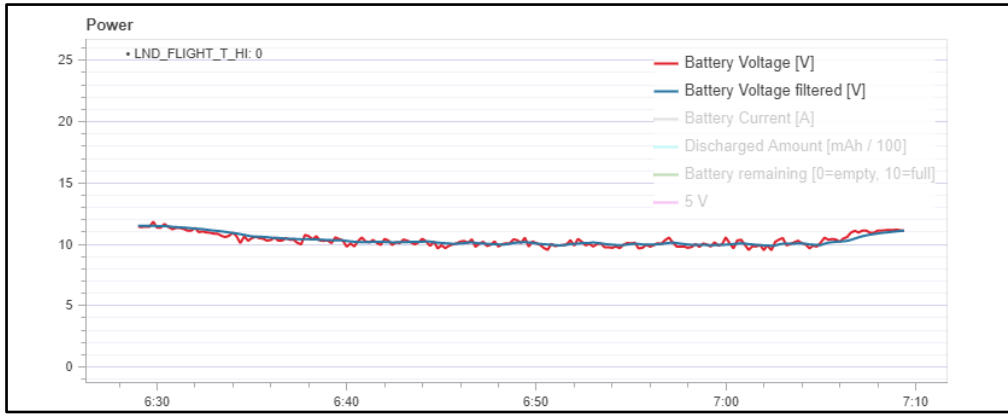


Figure 36 Battery Voltage plot for First Test Flight

As the goal of the first flight is to test the MAVROS communication between Raspberry Pi and Pixhawk and to enter the vehicle into Offboard mode, the test proved to be a success. The vehicle has a stable and better communication with the onboard computer. The vehicle is also able to enter the Offboard mode without any issues.

The second flight test is focused on updating the vehicle with the previously tuned PID gains from section 4.1. Also prior to the flight, some modifications are performed on the vehicle from the embedded team and additional sensors are introduced to the vehicle. Due to the added weights prior to second flight, it is observed that the maximum thrust generated by the vehicle is not sufficient for vehicle to take-off to the desired setpoint unlike the first flight. The thrust plot for the second flight test is displayed in Figure 37. Since the nominal hover thrust required is not achieved while flying the vehicle manually, the test for offboard control is not performed.

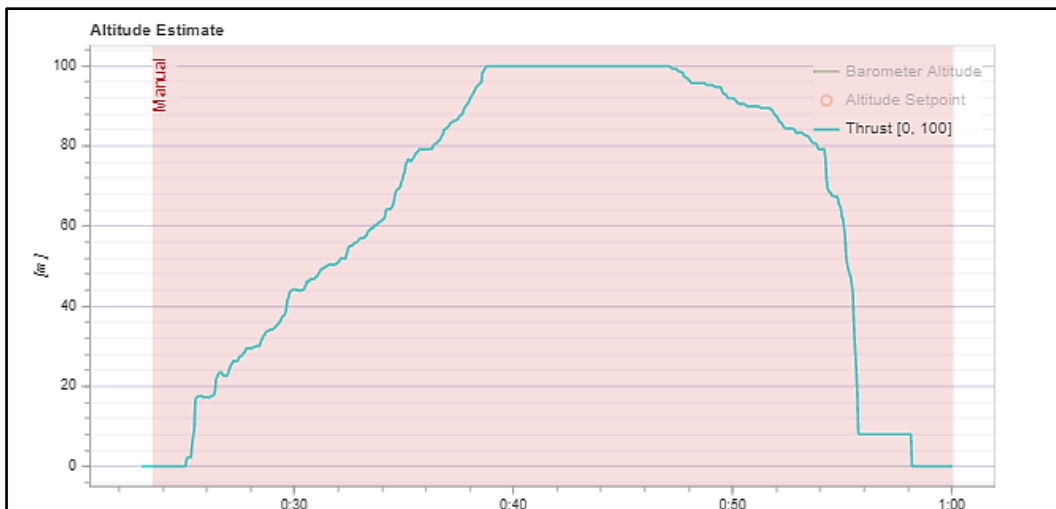


Figure 37 Thrust Plot for Second Flight Test

5 Conclusion and perspectives

ROS was chosen as an offboard API due to the fact that it is largely used in the robotics community thus having a vast number of libraries which are necessary for implementing all the algorithms. This also gave us the ability to create nodes simulating the PX4 autopilot software and allowing to create real life missions of predefined waypoints in QGroundControl, which can then be given to the algorithms.

A communication algorithm controlling the switch between the air and the ground module is implemented in the hybrid vehicle. The decision to switch is purely based on the waypoint given by the user. The algorithm is validated by running simulations in the gazebo environment. In order to achieve better performances, PID tuning is carried out for the hybrid vehicle and the vehicle response is analysed.

Next, a safe landing algorithm which addresses the safety aspect of the hybrid vehicle is developed. Whenever batter level drops down below a threshold, this algorithm decides the vehicle must land. While landing, the algorithm is capable of detecting the suitable areas underneath the vehicle to perform a safe landing by avoiding damages to the vehicle. The algorithm is validated by performing simulations using QGC maps and visualizing the performance in Rviz.

Communication between the flight controller and Raspberry Pi is established using a FTDI serial to USB adapter, which proved to be a better reliable medium for communicating. This set-up is validated by testing the vehicle in offboard mode in the drone lab. The real-time test of the hybrid vehicle showed that the onboard propulsion system is not able to provide sufficient nominal thrust for the current configuration of the vehicle.

The results achieved so far in this work provides a concrete foundation for the upcoming batches to further develop and upgrade the existing modules. Firstly, the communication module can be further optimized by adding constraints on time, energy and power, allowing the vehicle to decide on which module to prefer based on those constraints. Secondly, the safe landing algorithm needs validation in real-time environment and this can be achieved by integrating camera sensors at appropriate locations on the vehicle. Finally, in order to carry out a mission with acceptable endurance, there is a need to re-visit the current propulsion system configuration and make necessary modifications.

6 References

- [1] A. Sambalov, "Autonomous robotic aerial vehicle: S3 project report", March 2020.
- [2] R. Rodrigues, "Autonomous robotic aerial vehicle: S3 project report", March 2021.
- [3] B. Ismael, "Autonomous robotic aerial vehicle: S3 project report", March 2021.
- [4] K. Mallabadi, R. Dsouza "Autonomous Robotic aerial vehicle's control in Complex Environments: S2 project report", June 2021
- [5] "PX4 Development | PX4User Guide", Docs.px4.io [Online]. Available: <https://docs.px4.io/master/en/development/development.html>
- [6] N. Chow, G. Guban and A. Haque, "RADR: Routing for Autonomous Drones," *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Tangier, Morocco, 2019, pp. 1445-1450, doi: 10.1109/IWCMC.2019.8766530.
- [7] S. Islam and A. Razi, "A Path Planning Algorithm for Collective Monitoring Using Autonomous Drones," *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, USA, 2019, pp. 1-6, doi: 10.1109/CISS.2019.8693023.
- [8] M. Lupascu, S. Hustiu, A. Burlacu and M. Kloetzer, "Path Planning for Autonomous Drones using 3D Rectangular Cuboid Decomposition," *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, 2019, pp. 119-124, doi: 10.1109/ICSTCC.2019.8886091.
- [9] Elaf Jirjees Dhulkefl, & Akif Durdu. "Path Planning Algorithms for Unmanned Aerial Vehicles". *International Journal of Trend in Scientific Research and Development*, (2019) 3(4),359–362. <http://doi.org/10.31142/ijtsrd23696>.
- [10] Armin Hornung, Kai M.Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.
- [11] "Gazebo", GazeboSim.org. [Online]. Available: <http://gazeboSim.org/>
- [12] ZOMPAS A. Development of a three-dimensional path planner for aerial robotic workers (Master's thesis, University of Twente).
- [13] Zammit C, Van Kampen EJ. Comparison between A* and RRT algorithms for UAV path planning. In 2018 AIAA guidance, navigation, and control conference 2018 (p. 1846)
- [14] Sahoo RR, Rakshit P, Haidar MT, Swarnalipi S, Balabantaray BK, Mohapatra S. Navigational path planning of multi-robot using honey bee mating optimization algorithm (HBMO). *International Journal of Computer Applications*. 2011 Aug;27(11):1-8.
- [15] Jun M, D'Andrea R. Path planning for unmanned aerial vehicles in uncertain and adversarial environments. In *Cooperative control: models, applications and algorithms 2003* (pp. 95-110). Springer, Boston, MA.
- [16] Knispel L, Matousek R. A performance comparison of rapidly-exploring random tree and Dijkstra's algorithm for holonomic robot path planning. *Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology*. 2013:154-62.
- [17] Raheem FA, Hameed UI. Path planning algorithm using D* heuristic method based on PSO in dynamic environment. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*. 2018 Dec 11;49(1):257-71.
- [18] Kim J, Jo K, Kim D, Chu K, Sunwoo M. Behavior and path planning algorithm of autonomous vehicle A1 in structured environments. *IFAC Proceedings Volumes*. 2013 Jun 1;46(10):36-41.
- [19] Minh VT, Pumwa J. Feasible path planning for autonomous vehicles. *Mathematical Problems in Engineering*. 2014 Jan 1;2014.
- [20] "Documentation - ROS Wiki", Wiki.ros.org. Available: <http://wiki.ros.org/Documentation>.

Appendices

Annex 1: *safe_landing_offb.launch*

```
<launch>

  <arg name="world_file_name" default="simple_obstacle" />
  <arg name="world_path" default="$(find avoidance)/sim/worlds/$(arg world_file_name).world" />
  <arg name="pointcloud_topics" default="/camera/depth/points"/>

  <!-- Define a static transform from a camera internal frame to the fcu for every camera used -->
  <node pkg="tf" type="static_transform_publisher" name="tf_depth_camera"
    args="0 0 0 1.57 3.14 0 fcu camera_link 10"/>

  <!-- Launch PX4 and mavros -->
  <include file="$(find avoidance)/launch/avoidance_sitl_mavros.launch" >
    <arg name="model" value="iris_downward_depth_camera" />
    <arg name="vehicle" value="iris" />
    <arg name="world_path" value="$(arg world_path)" />
  </include>
  <!-- Launch local planner -->
  <node name="safe_landing_planner_node" pkg="safe_landing_planner" type="safe_landing_planner_node"
    output="screen">
    <param name="pointcloud_topics" value="$(arg pointcloud_topics)" />
    <param name="world_name" value="$(find avoidance)/sim/worlds/$(arg world_file_name).yaml" />
  </node>

  <node name="waypoint_generator_node" pkg="safe_landing_planner" type="waypoint_generator_node"
    output="screen" >
  </node>

  <node name="dynparam_slpn" pkg="dynamic_reconfigure" type="dynparam" args="load
safe_landing_planner_node $(find safe_landing_planner)/cfg/slpn.yaml" />
  <node name="dynparam_wpgn" pkg="dynamic_reconfigure" type="dynparam" args="load
waypoint_generator_node $(find safe_landing_planner)/cfg/wpgn.yaml" />

  <node name="rviz" pkg="rviz" type="rviz" output="screen" args="-d $(find
safe_landing_planner)/resource/safe_landing_planner.rviz" />

  <node name="offb_node" pkg="beginner_tutorials" type="offb_node" >
  </node>

</launch>
```