Heidelberg University, Heidelberg

SRH Hochschule Heidelberg

Master's Thesis
Vision based Indoor Navigation of an Unmanned Aerial Vehicle

Author:	Mandlik Saurabh
Matriculation Number:	11013278
Address:	Am Steingarten 12, 18 68169, Mannheim Germany
Email Address:	saurabhmandlik97@gmail.com
Supervisor:	Prof Dr. Achim Gottscheber
Supervisor:	Mr. Holger Dieterich
Begin:	01. December 2020
End:	30. June 2021

Abstract

This thesis presents the system for UAV which automate a task that can performed by humans in warehouses. The UAV can navigate and detect the objects in non-GPS indoor environment. This system can solve the many problems of inventory warehouses. It is safer, cheaper and faster than humans to do the inventory work such as scan the barcode, separate out the boxes through detection method etc. AprilTag marker is used for localization and position through Real-Sense Depth Camera. The performance of the developed system was used in Raspberry Pi 4 microcomputer, Robot Operating System (ROS) framework, and computer vision approach for detecting the objects. Mission Planner software was used as a ground control station to control the UAV.

Declaration

Statement of Authenticity

The author of this report declares that he has prepared the submitted work himself, unassisted and without any other resources other than those indicated. All the direct and indirect cited information from other sources (including electronic sources) is duly acknowledged without exceptions. The material , in this or similar form , has not been previously submitted , either in full or in part , for other exams at this or any other academic institution. The enclosed Master Thesis is the same version as the version evaluated by the supervisors.

Heidelberg, Germany
30 June 2021

Acknowledgement

It is an honor to present the thesis report on “Vision Based Indoor Navigations of an UAV”. I have received a lot of support throughout the writing of this thesis.

First, I would like to thank my Dean, Prof Dr. Achim Gottscheber at SRH Hochschule Heidelberg for providing me with continuous support and steering me in the right direction whenever I needed it. The door to Prof. Gottscheber was always open whenever I have some queries and doubts regarding thesis.

I would like to acknowledge and continual support from my supervisor Mr. Holger Dieterich who consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed it. The working experience with him was unforgettable as he always stood behind me to clear my silliest doubts and encourages me to do new things. This thesis would not have been possible without his guidance, support and patience.

Theses acknowledgment is incomplete without mentioning my colleagues from the Automation Department of Heidelberg University for having the friendly working atmosphere. Special thanks to lab technician Mr. Frank Stolzenberger who helped me to provide some hardware related stuff.

Last but not least, I express my gratitude to thank my parents to give me constant support and encouragement during my entire study years. I am forever thankful for the unconditional love and support throughout the entire thesis process and every day.

Table of Contents

Chapter 1
Introduction

1.1 Motivation:

Indoor autonomous navigation of UAVs has been actively studied in robotics community. Indoor applications have less boundary condition compared to outdoor applications. The main purpose is to make the inventory process completely autonomous. Indoor navigation of an UAV is used in different field such as in warehouse operations, where it can be used in inventory management, intra-logistics and inspection and surveillance etc.

In manufacturing and production companies, various tasks are gradually being automated to be performed by robots instead of human labours. Drone technology's increasing use in the sector is radically changing the business paradigm and redrawing industrial landscapes. Infrastructure, transportation, insurance, media, entertainment, telecommunications, agriculture, mining, and security are all affected by this trend. Drone-based solutions are best suited to industries that value mobility and good data quality. The take-off/landing system is one of the most important parts in the development of autonomous systems based on UAVs. It should be able to operate independently and with the required precision, but the electronics on board the UAV should be as light as feasible and consume as little computational power as possible.

Motivation of the thesis is to automate a task that is currently performed by humans. The indoor navigation is more safe, cheaper and faster than lifting a human to scan the barcode in warehouses.

1.2 Objective

The objective of the thesis is:

- Find an appropriate localization method to enable autonomous navigation of an UAV.
- Navigate the UAV in a given relative position to a objected in the live video stream.

1.3 Performance factors

Some performance factors had considered while implementing this model.

Power consumption must be the important factor in UAVs applications. If the system will become portable, flexible then power must be kept low. Here, the StromPi battery unit gives portable supply to on board UAV computer.

Robustness is getting from the minimum errors and less disturbance of the moving objects.

Scalability is basically depends on the how large is the working area. This project is developed in indoor environment as it requires large area to cover the required task and detect the multiple objects.

Complexity is related to the project design i.e., particular system. It should be easy to install in new environment. It must not take too much time to calibrate some technical things and so on.

Portability describes how easy to move this hardware from one place to another. In short, it depends on weight, size and power module.

Accuracy is an important parameter while detecting the objects in indoor environment. Because of light conditions, the objects might get some time to detect by UAV.

Latency is when dealing with a moving target, latency is critical since a large delay will result in a significant level of inaccuracy. The period in seconds between when the system gets data and when it is completed processing it is known as latency.

1.4 Thesis organization

The structure of this thesis is as follows.

- Chapter 1 gives the introduction and background of thesis topic. It’s become easy to understand the topic objective and basic information of the project topic from chapter 1.

- Chapter 2 reviews the indoor navigation systems used by UAV. This chapter helps to find the ongoing and previous research has been done in this related indoor navigation field. The second chapter has a way to get some solutions to implement this thesis topic.

- Chapter 3 explain the how first objective of the thesis has achieved using deep learning method. In that, different object detection techniques have been explained.

- Chapter 4 gives information about the overall system architecture of UAV. Here, all the hardware related work and used components has used in the UAV is explained. This chapter presents the general overview of hardware components and software design.

- Chapter 5 presents the UAV configurations and steps required to initialize the system. Here, ROS nodes, topics, fiducial marker and position estimation of an UAV have discussed.

- Chapter 6 shows the results that carried out from few experiments. The issues and their solutions that occurred during experiments have discussed in this section.

- Chapter 7 presents the conclusion of the work conducted and discussed about future work.

Chapter 2
Literature Survey

This chapter reviews the theory and survey related to indoor positioning of UAV. This section introduces a several methods to obtain a robust autonomous navigation [1], autonomous landing on unmanned surface vehicle [2], and various applications of UAV used in indoor environment [3].

Review of state-of-the-art Indoor UAV Platforms

From past few years, there has been number of innovations in warehouses for indoor navigation system. Although it’s a challenging task and following research has helped to choose the appropriate method require to complete this thesis topic.

2.1 Autonomous warehouse inventory management [4] research explains the working of unmanned aerial vehicle (UAV) and unmanned ground vehicle (UGV). The UGV is basically a ground reference to get the position for UAV. This approach is somehow similar to this thesis topic. They have presented a novel technique for the automation of warehouse task. They used UGV as a carrying platform in GPS denied environment while UAV is used as the mobile scanner to scan the barcode on goods. The UAV has mounted a camera on front side which will scan the barcode. The UGV starts to navigate the rows of racks which carrying the UAV. In the given rack, there are coded markers has pasted which indicates the ID of appropriate rack number as shown ion fig. 1.

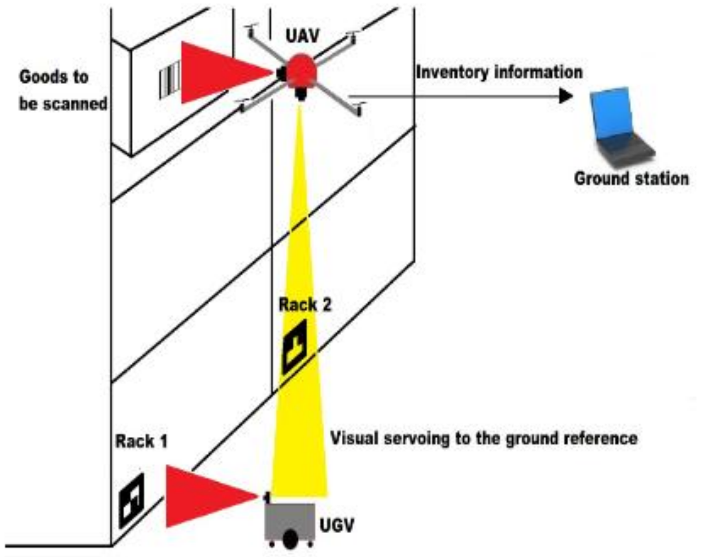


Fig. 2.1 Autonomous Inventory Scanning Method [4]

The UGV is also having the front facing camera to detect the marker. When UGV detect the marker, it will stop at that moment and UAV will start to fly vertically for scanning the barcodes placed on goods. When UAV reaches at top, UGV changes its position and moves to next rack marker. As UGV is ground reference for UAV, it will follow UGV. Now, the UAV is located on top of second rank. It will start to scan the barcode from top to bottom. The process will go until the whole row is completely scanned.

The UAV also has a down facing camera to detect the marker which is placed on upper side of UGV for vision based target tracking. When the whole row of racks has scanned successfully, the UAV will land on UGV from the Augmented Reality (AR) marker and it will recharge its batteries before moving forward to next row.

2.2 The author from [1] presented **the robust navigation of UAVs in warehouses** using another method from previous research. They have developed low-cost sensing system using Extended Kalman Filter (EKF). In their research paper, they have mainly focused on the three things: 1) neglect the outliers, inherent drift using Mahalanobis norms, 2) incorporation of visual SLAM by introducing pseudo-covariance, 3) recognition of floor lanes to get position and yaw measurement.

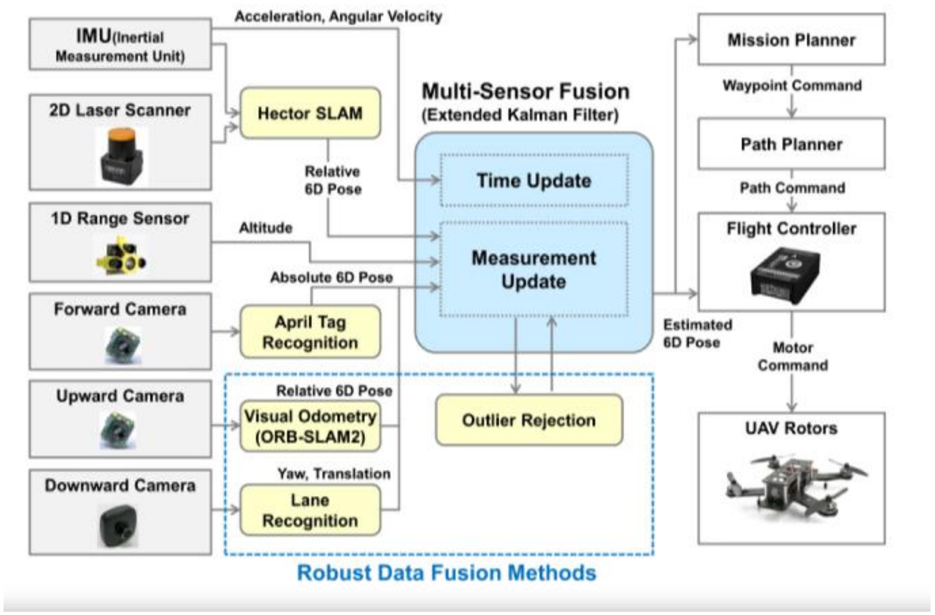


Fig. 2.2 Robust Data Fusion Methods [1]

In this research paper, author suggests multi-sensor fusion method with the estimated pose and applies Extended Kalman filter (EKF) using three cameras, 2D laser scanner, 1D range sensor, and IMU as shown in fig. 2. EKF will acquire the data from sensors in the form of relative and absolute angles, position and orientation. The SLAM method provides the 6D relative positions i.e. x, y, z, roll, pitch and yaw of the UAV. While absolute positions will get from April Tag marker detection algorithm.

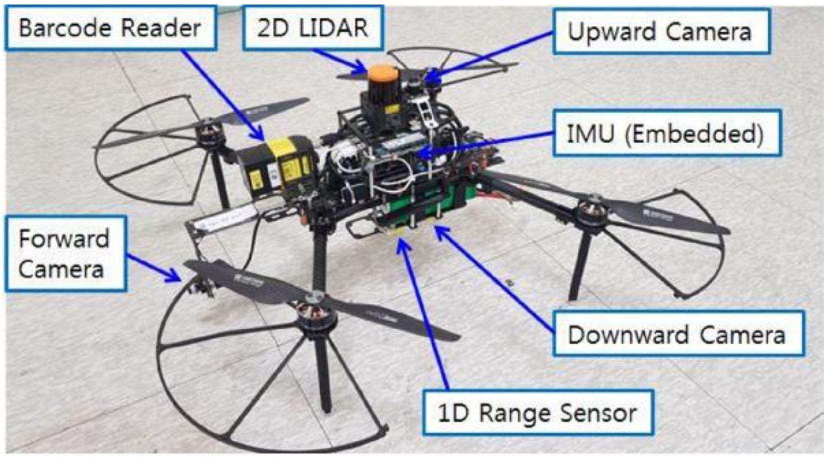


Fig. 2.3 The sensing system of our autonomous UAV [1]

The 1D range sensor is used as an altitude measurement to measure the distance up to 14 meters from floor to UAV. The forward camera will detect the April Tag maker to get the absolute 6D pose data of UAV. The EKF will sends the estimated UAV poses to flight controller and ground station as shown in Fig.2.

2.3 The research paper [5] explained about the UAV landing on mobile collaborative ground robot reference on the basis of IR marker detection. They have used two robots i.e., unmanned aerial vehicles (UAV) and unmanned ground vehicles (UGV) for precise localization. Using the help of 2D LIDAR sensors, camera and ultrasonic system they will get the precise localization.

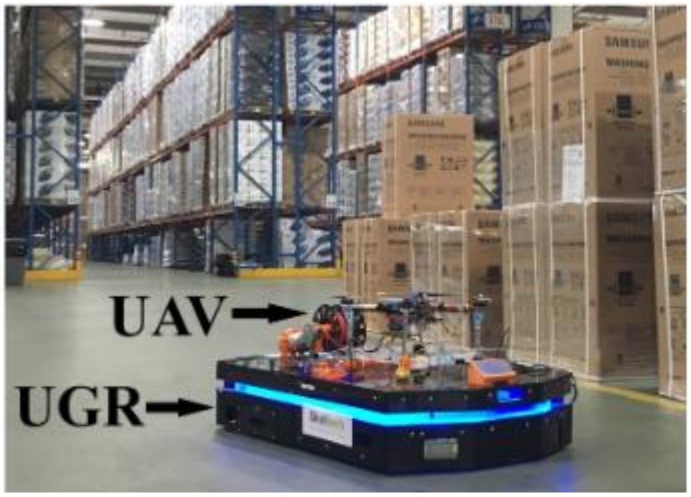


Fig. 2.4 System of two collaborative robots [5]

They have placed camera on mobile robot and IR markers are on the UAV. The localization is possible with fusing two IR markers patterns i.e., small and big marker. The small marker will help to take-off and land the UAV on robot and the bigger will use to calculate the height above 1 m. The ultrasonic sensors are connected to on board computer of the UGV to solve the problem of precise position estimation on high altitudes.

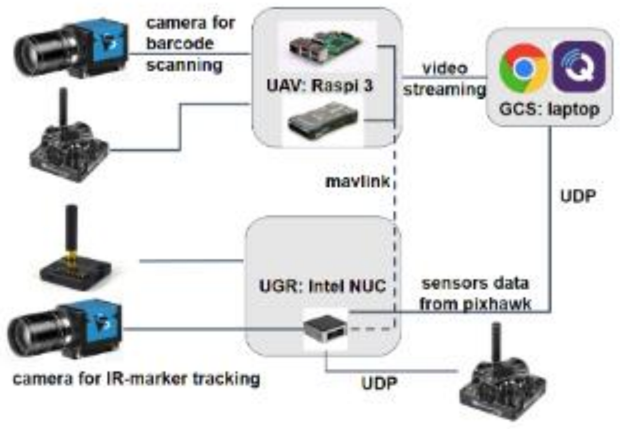

camera for barcode scanning
UAV: Raspi 3
video streaming
GCS: laptop
mavlink
UDP
UGR: Intel NUC
sensors data from pixhawk
camera for IR-marker tracking
UDP

Fig. 2.5 LPE using visual information [5]

The above figure is the communication setup of UAV and UGV where camera to be used for scanning the barcode is connected to UAV on board computer i.e., raspberry pi. The UAV can calculate the global coordinates using its coordinate relative to UGV.

2.4 The authors from [6] presented real-time implementation of object detector and tracking system for AR Drone 2 using Single Shot Detector (SSD) (see section 3.2.4) neural network. Their aim is to detect and track the target object using drone camera. Using the front camera, they have developed the tracking algorithm which calculate the parameters such as roll, pitch, yaw and altitude. However, these parameters are controlled by PID controller which takes the input as a position and distance of the target object.

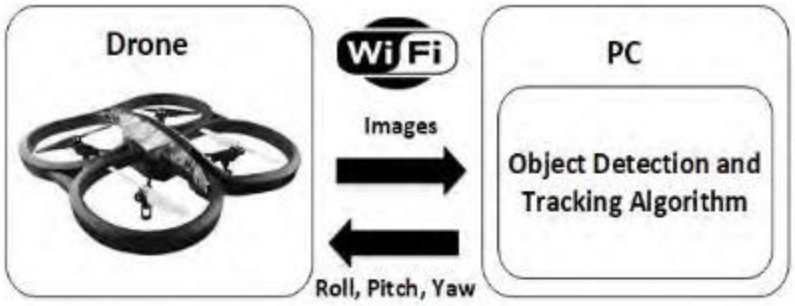


Fig. 2.6 Drone PC communication [6]

For detecting the object, drone sends the captured image to PC via Wi-Fi using front camera. The PC will train the images in CNN which used single shot detector (SSD) architecture. When the object is detected in an image, it returns the bounding box of the detected object. Using that bounding box, it will find the center of the detected object. This center will be use as an input to PID controllers. The benefit of using the SSD architecture is that object will detected with high accuracy while reducing the computational time.

Chapter 3
Computer vision preliminaries

Computer vision is a branch of artificial intelligence that teaches computers to read, understand, and interpret images. Machines can reliably recognize and classify items using digital images from cameras and movies, as well as deep learning models, and then detect the appropriate identified objects. Computer vision offers various advantages that make it useful in the AI area, such as making processing power more affordable and accessible, and allowing new algorithms like convolutional neural networks to take advantage of hardware and software capabilities.

Deep learning enables computational models with several processing layers to learn and represent data at multiple levels of abstraction, simulating how the brain receives and analyzes multimodal information, and so implicitly capturing intricate data structures. Deep learning approaches have recently sparked interest due to their ability to beat prior state-of-the-art techniques in a variety of tasks, as well as the amount of complicated data from various sources (e.g., visual, audio, medical, social, and sensor). Autonomous systems, facial recognition, self-driving automobiles, picture and speech recognition, categorization, and object detection are just a few of the applications. Unmanned Aerial Vehicles (UAVs), which are becoming a popular choice for a variety of applications, are one of the most promising Deep Learning systems.

3.1 Deep learning and Neural Networks:

Deep learning methods have outperformed prior state-of-the-art machine learning techniques in various disciplines in recent years, with computer vision being one of the most prominent examples. Deep learning allows computational models with multiple processing layers to learn and represent data at multiple levels of abstraction, mirroring how the brain processes multimodal input information and implicitly capturing sophisticated data structures.

Deep learning (DL) in Artificial Intelligence (AI) has recently gained a significant interest. Autonomous systems, facial recognition, self-driving automobiles, picture and speech recognition, categorization, and object detection are just a few of the applications. Unmanned Aerial Vehicles (UAVs), which are becoming a popular solution for a variety of applications, are one of the most promising Deep Learning systems.

Recently, Convolutional Neural Networks (CNNs) have achieved great results indifferent fields of recognition, detection, and classification, especially in computer vision. CNNs are a type of deep neural network that is used to evaluate visual images. CNN is regarded as a very strong technology in the field of object identification and categorization. CNNs are hierarchical models inspired by biology that may be taught to perform various detection, identification, and segmentation tasks.

Object detection basically has two type’s namely specific objects and generic objects. There is an issue with matching problem in specific objects in generic object type, we have to detect instances of some predefined objects.

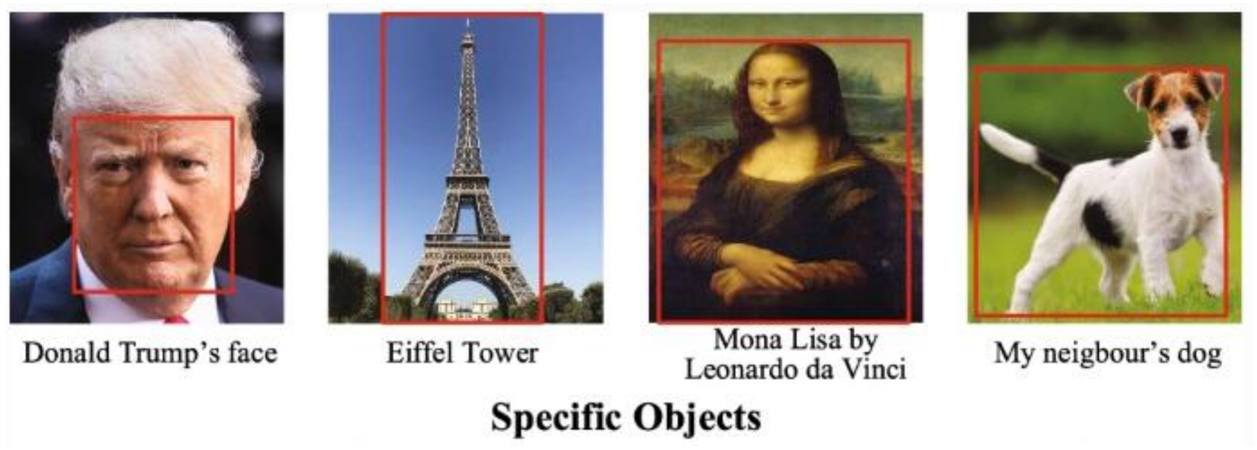


Fig 3.1 Specific Objects for Object Detection [7]

The most representative models of deep learning, convolutional neural networks (CNNs), are able to leverage the basic qualities underlying real signals: translation invariance, local connection, and compositional hierarchies.

3.2 Detection Frameworks

As mentioned earlier in introduction part that there are several object detection frameworks are present. In this chapter, the different CNNs are explained.

3.2.1 Region Based Computational Neural Network (R-CNN):

The RCNN method suggests a set of boxes in the image and tests to see if any of them contain any objects, rather than acting on a vast number of regions. To extract these boxes from a picture, RCNN employs selective search (these boxes are called regions).

First, let's define selective search and how it distinguishes between different regions. An object is made up of four regions: different scales, colors, textures, and enclosure. Selective search recognizes these trends in the picture and suggests different regions based on them. Here's how targeted quest works in a nutshell:

Here, Image should be the first input:

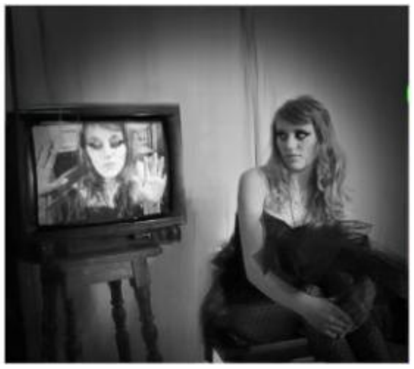


Fig 3.2 Input Image (A) [8]

Then, it generates initial sub-segmentations so that it has multiple regions from this image:

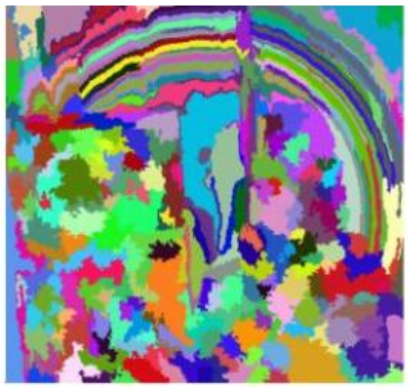


Fig 3.2 (B) Segmented image [8]

Following that, the approach joins the related sections to make a larger zone.

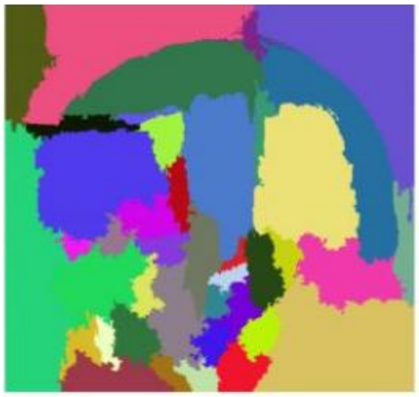


Fig 3.2 (C) Image with RoI [8]

- Lastly, these regions then produce the final object locations (Region of Interest).

The following is a quick rundown of the steps taken by RCNN to detect objects:

1. Start with a convolutional neural network that has already been educated.
2. The model is then retrained. The last layer of the network is trained using the amount of classes that need to be detected.
3. Get the Region of Interest for each picture as the third stage. Then, to match the CNN input size, restructure all of these regions.
4. Train SVM to identify objects and history after we get the regions. One binary SVM is trained for each class.
5. Finally, for each defined entity in the picture that train a linear regression model to produce tighter bounding boxes.

Here is the input image,

Fig 3.3 (A) Input image for RCNN [8]

The Regions of Interest (ROI) are then created using a proposal approach:

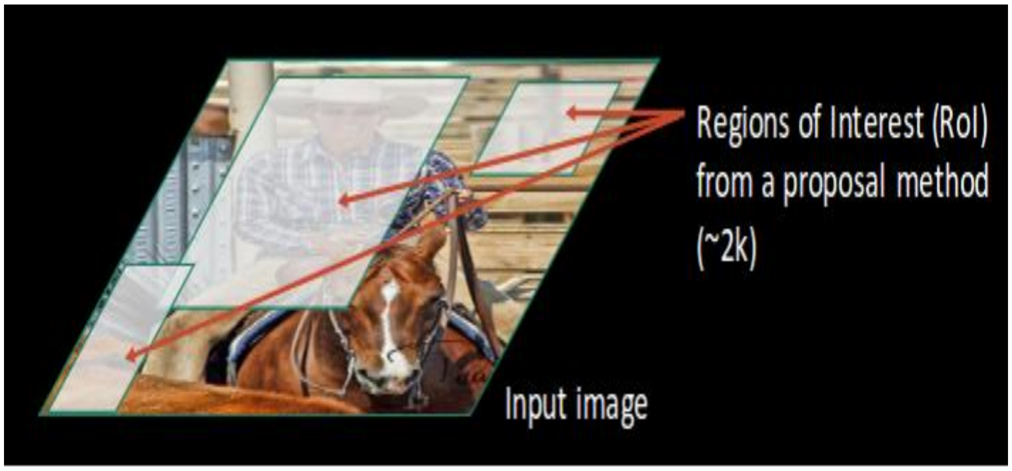
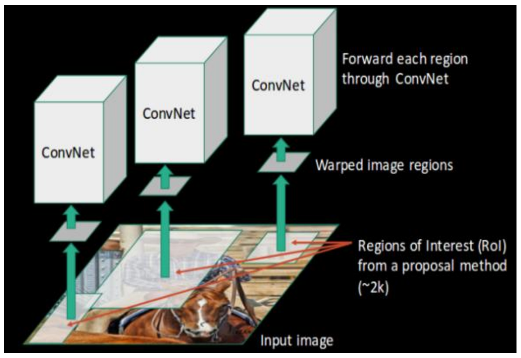


Fig 3.3 (B) Image with ROI [8]

After that, all of these regions are reshaped according to the CNN's input, and each region is transferred to the ConvNet.


ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

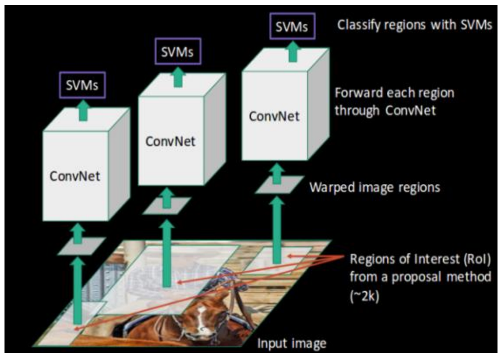

SVMs
SVMs
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Fig 3.3 (C) Generate ConvNet [8]

Fig 3.3 (D) Classify regions with SVMs [8]

SVMs are used to segment these regions into separate groups after CNN extracts features for each region. Lastly, a bounding box regression (Bbox reg) is utilized to predict the bounding boxes for each supplied field.

Problems with RCNN

RCNN method has its own set of limitations. The following steps make training an RCNN model both costly and time-consuming:

• Using a selective scan, extracting 2,000 regions for each picture
• Use CNN to extract features for each picture field. The number of CNN features would be N*2,000 if we have N videos.

The entire object detection mechanism with RCNN is divided into three models:
1. Function extraction using CNN
2. Object recognition using a linear SVM classifier
3. Bounding box tightening using a regression model

3.2.2 Fast R-CNN

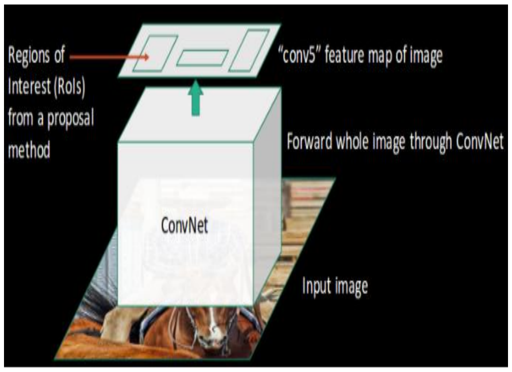
Feed the input image to the CNN in Quick RCNN, which then generates the convolutional feature maps. These maps are used to extract proposal regions. The proposed regions are then molded into a specific size using a RoI pooling layer before being supplied into a fully linked network.

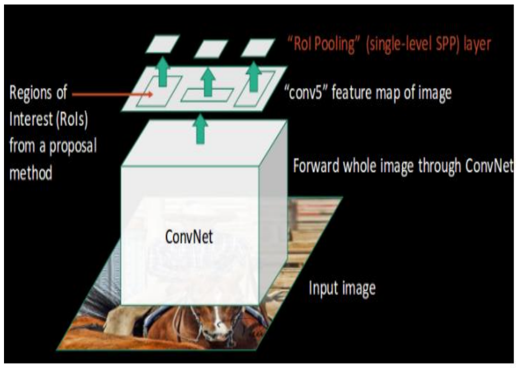
To make the idea clearer, let's break it down into steps:

1. As with the previous two techniques, we start with a picture.
2. A ConvNet is used to build the Regions of Interest from the image.
3. On all of these regions, a RoI pooling layer is added to reshape them according to the ConvNet's feedback. After that, each area is linked to a larger network.
4. To output groups, a softmax layer is used on top of the fully connected network. A linear regression layer is used in parallel with the softmax layer to produce bounding box coordinates for predicted groups.

Rather than using three separate models (as in RCNN), Quick RCNN employs a single model that extracts features from regions, divides them into classes, and At the same time, create boundary boxes for the defined groups.

Below is the visualization of each step that discussed above


Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Fig 3.4 (A) Getting RoI from Image [8]


"RoI Pooling" (single-level SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Fig 3.4 (B) RoI Polling layer [8]

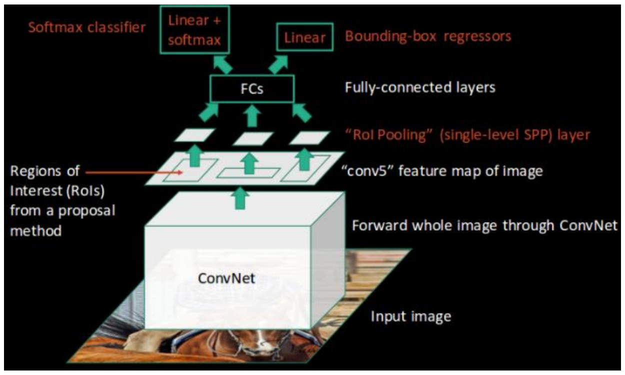

Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" (single-level SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image

Fig 3.4 (C) Softmax layer produce the bounding box [8]

Problems with Fast RCNN

There are some issues with Quick RCNN.
It also proposes using selective search to find the Regions of Interest, which is a slow and time-consuming process.
Detecting objects takes around 2 seconds per image, which is much faster than RCNN.
However, when dealing with massive real-world datasets, even a Fast RCNN becomes slow.

3.2.3 Faster R-CNN:

Faster RCNN is a variant of Fast RCNN that has been tweaked. The main difference is that Fast RCNN generates Regions of Interest using selective search, whereas Faster RCNN uses RPN (Region Proposal Network). RPN takes picture feature maps as input and produces a list of object proposals, each ranked for objectness.

In a Faster RCNN technique, the following stages are commonly followed:

1. Pass an image to the ConvNet, and it will return the image's feature map.

2. on these function maps, a region proposal network is implemented.
The object proposals are returned, along with their objectness ranking.

3. These proposals are subjected to a RoI pooling layer, which reduces the size of all proposals to the same level.

4. The ideas are transported to a completely connected layer with a softmax layer and a linear regression layer at the top to categorize and output the bounding boxes for artifacts.

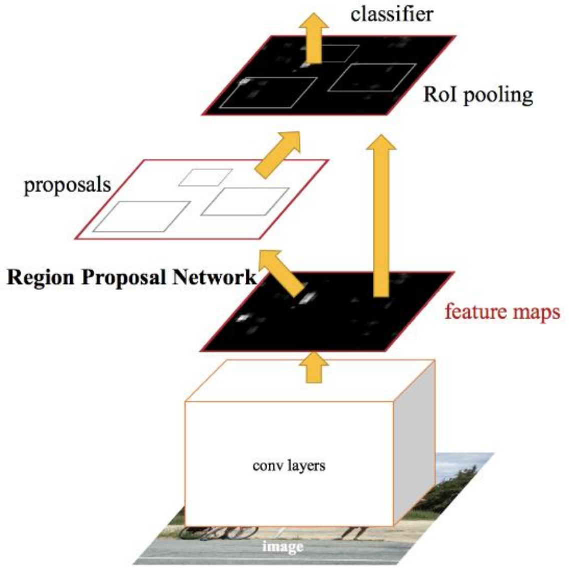


Fig 3.5 (A) Faster RCNN Layered Structure [8]

First, discuss about the Regional Proposal Network (RPN). RPN is the region in the image where an object might be located. After classifying the objected area as foreground class, while the area where the object is not present is labeled as background class.
Anchor boxes are created by RPN using a sliding window. Anchor boxes are predefined bounding boxes that come in a variety of sizes and shapes.

Once anchor boxes generated, next step is to find out IoU (Intersection over union).

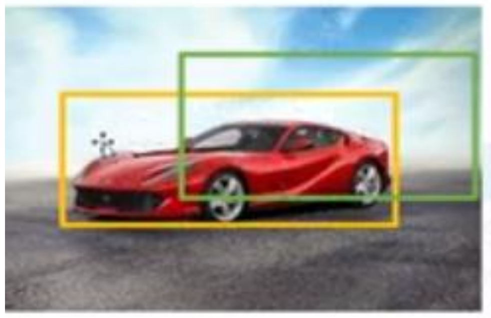


Fig 3.5 (B) Generate IoU

The overlapping region from the previous figure is referred to as IoU. If the overlapping region is greater than 50%, the object will be identified by the box. Foreground class refers to the anchor box with the highest IoU.

3.2.4 Single Shot Detector (SSD):

- Input image gives to VGG16 network to extract feature map.
- SSD makes 8732 predictions for every single class using 6 convolutional layers.
- This 6 CNN layers performed classification and detection task.

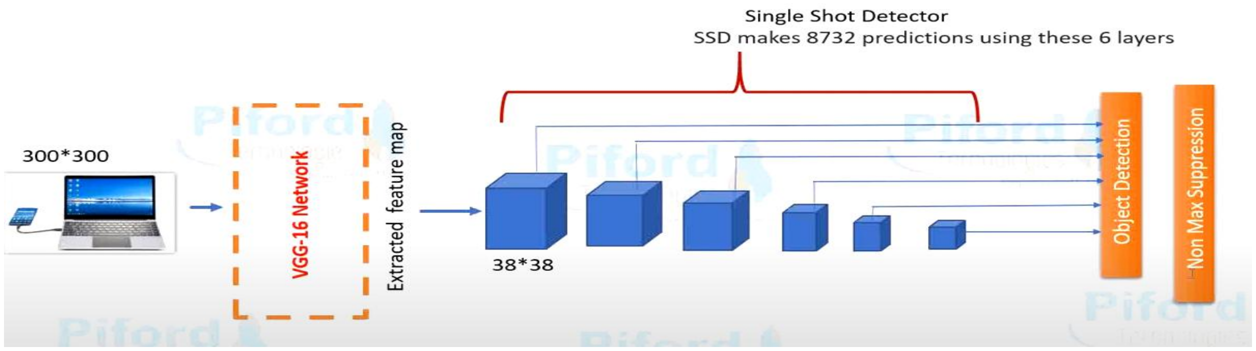


Fig 3.6 (A) SSD Structure

- We used non-max suppression just to remove duplicate predictions.

Let’s take an example for better understanding of SSD.

Below image is the input of SSD and it must have ground truth box for each image as it can see in figure yellow and green box.

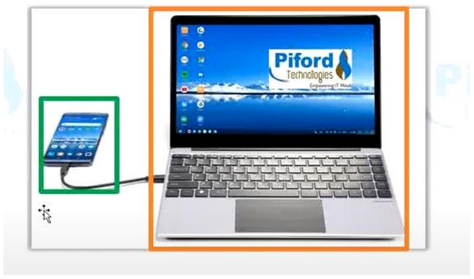


Fig 3.6 (B) Image with truth box

After this, convolutional layers are present here. The task of this CNN layers is to check boxes of different aspect ratios and each locations with different sizes.

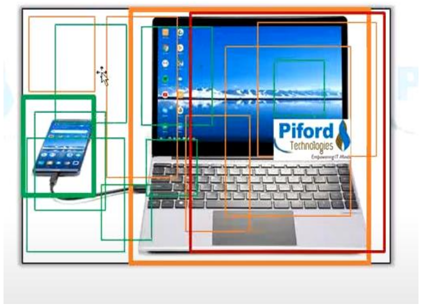


Fig 3.6 (C)

As discussed earlier, there are 8732 boxes per object in above figure. So for laptop there are 8732 bounding boxes and for mobile have also 8732 boxes. Here, it can see that there are multiple boxes are overlapping each other. We used IoU here to find out highest overlapping box. Using IoU concept we can detect object easily.

3.2.5 You Only Look Once (YOLO):

YOLO predicts the bounding box coordinates and class probability for these boxes using the entire image as a single case. The greatest advantage of YOLO is its tremendous speed; it can process 45 frames per second. YOLO is also aware of the concept of abstract object representation.

Following steps will help to understand the YOLO algorithm easily.
Suppose following image is the input for YOLO.

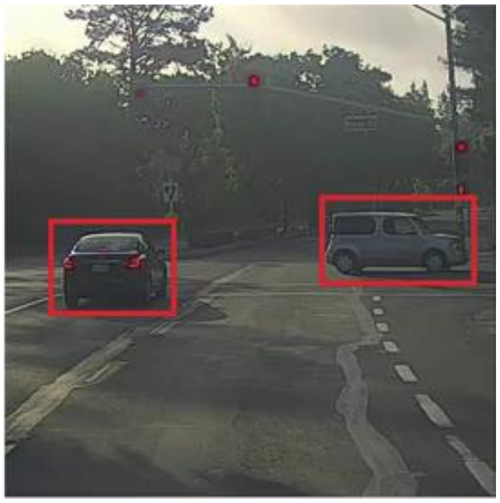


Fig 3.7 (A) Input image for YOLO [9]

The grid for YOLO framework would be 3 x 3.

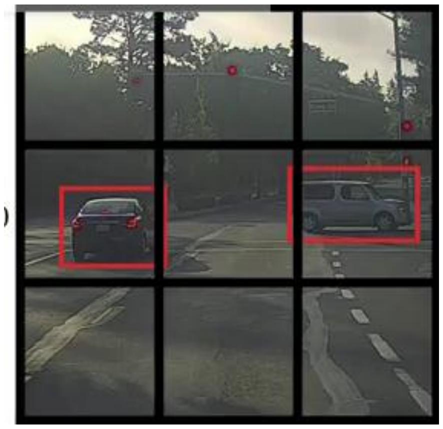


Fig 3.7 (B) Image with grid [9]

Each grid has an image localization and image classification. To comprehend the YOLO algorithm, one must first determine what is being expected. Finally, it can able to predict an object's class and the bounding box that defines its position.

Bounding box can be defined into five parts:

i. pc: It shows probability of whether object is present in the bounding box.
ii. bx,by: Defines the center of bounding box
iii. bh and bw: height and width of bounding box
iv. c1,c2,c3,…cn: It represents the class of an object such as cars, bikes, pedestrian etc.

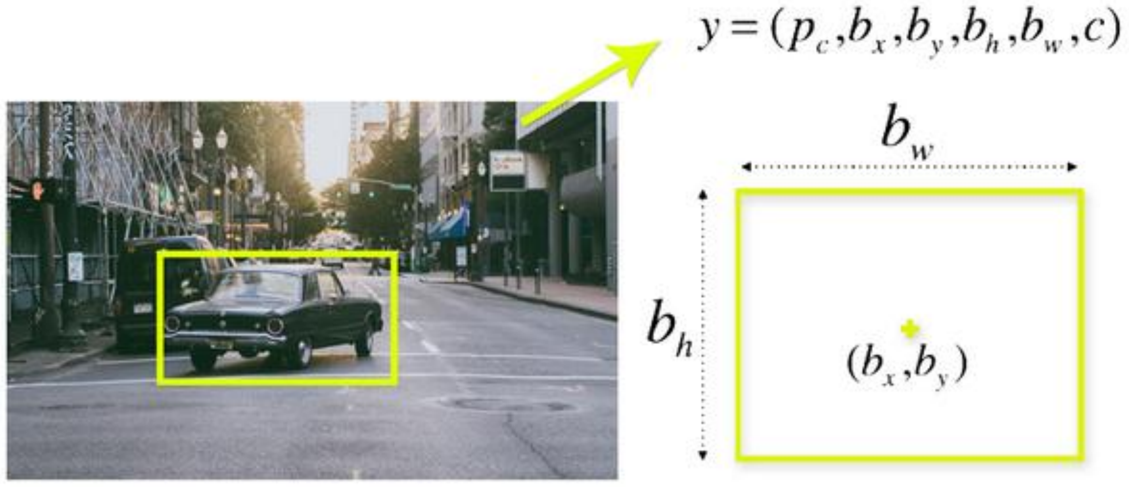


Figure 3.7 (C) Image with bounding box [9]

The algorithm to detect the object of the RCNN, Fast-RCNN and Faster-RCNN is on the basis region of interest of an image. And they defined these ROI using CNNs. But YOLO works differently. YOLO is not searching for region of interest. Instead, spill pictures into cells using a

19 x 19 grid. Each cell is in charge of forecasting five bounding boxes (in case there is more than one object in this cell).
As a result, for one picture, it gets 1805 bounding boxes.

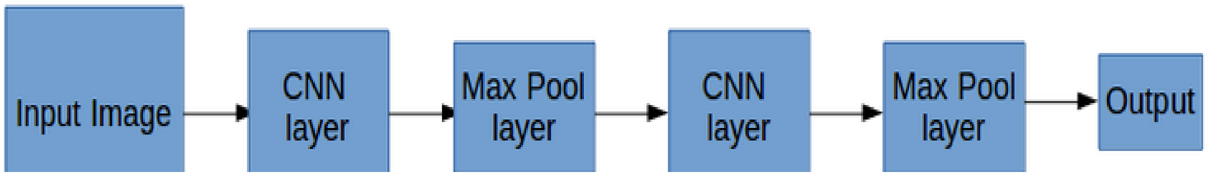


Figure 3.7 (D) YOLO layered Structure [9]

The majority of these cells and bounding boxes will be empty. As a result, we estimate the value pc, which is used in a method known as **non-max suppression** to delete boxes with low object likelihood and bounding boxes with the highest shared space.
How non-max suppression works?

While detecting the object in real-time, the object detection algorithm detects one object in many times. The detected object has surrounded with multiple bounding boxes as shown in Fig. 3.9 (E). Every image has pc value i.e. probability value. So, it selects the bounding box which has highest probability i.e. 0.9 in this case. Secondly, it takes highest IoU of the another box. Therefore, 0.6 and 0.7 probabilities are suppressed. This will goes until all the boxes after either selecting or compressing the bounding boxes.

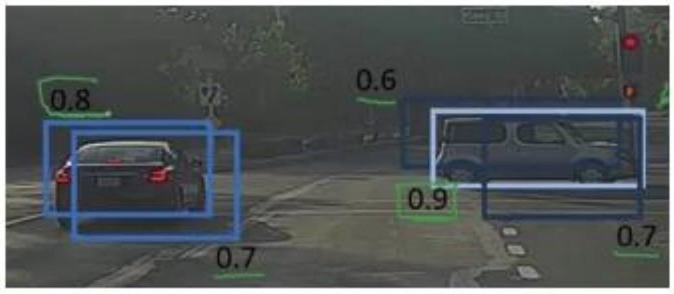


Figure 3.7 (E) Detected image [9]

Chapter 4
System Architecture Design For Unmanned Aerial Vehicle

As described in chapter 1, the purpose of this research is to investigate and develop a system for vision based navigation using UAV in the context of indoor environment.

Chapter 2 describes the literature review and also the materials and various theoretical methods applied in this research. Chapter 3 explains about the methods used in this research for predetermined object in video streams using computer vision approach.

This chapter presents the design of the system architecture for UAV. This section divided into two parts. First part covers the hardware design which included the UAV structure, serial connections and hardware setup. The second part gives the information about software which used to control the UAV.

Hardware Design

The system architecture for indoor navigation on board system as shown in fig 4.1 consists of quadcopter UAV (UAV name), an autopilot (Pixhawk), a microprocessor Raspberry Pi 4, Raspberry Pi camera, Intel Real-Sense depth camera D435, 4500 mAh Lipo Pack battery, Radio Control (Tx/Rx), StromPi Power and Battery Module WiFi connection and GSC (Mission Planner software) with additional component. Figure shows the hardware component attached to the UAV.

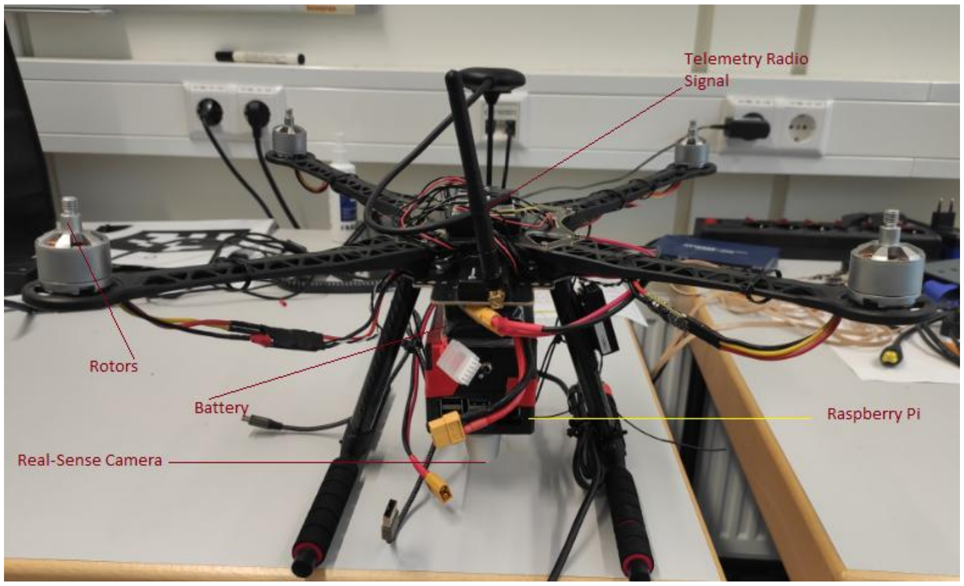


Fig. 4.1 Overall System with Hardware Components

4.1 Hybro S500 UAV Frame

The Hybro S500 V2 ARF frame kit is used for this project. It’s relatively low in cost as compared to other platforms. The dimensions are 383*383*240mm. The frame has available with BLHeli S ESCs, Propeller, motors and battery strips. The Hybro S500 has a payload capacity up to 1.8kg. The ARM adopts high strength of plastics.

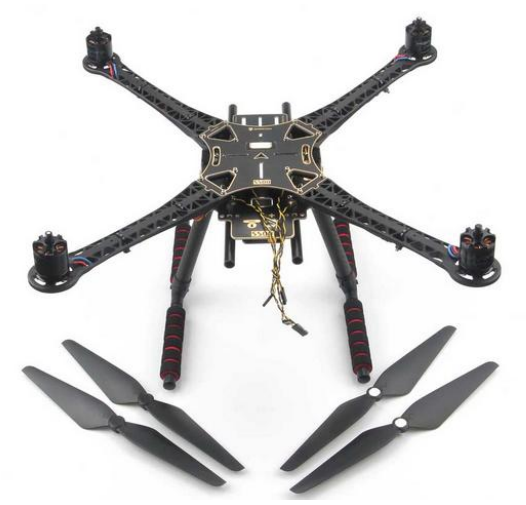


Fig. 4.2 UAV Frame

4.2 Pixhawk Autopilot

Pixhawk 4 is a flight controller unit (FCU) open source autopilot designed and manufactured by Holybro and the PX4 team. There are different types of versions are available in Pixhawk. Pixhawk 4 has been used for this project. It comes preinstalled with latest px4 firmware. Pixhawk FCU has in built in-built accelerometer, gyroscope, barometer and magnetometer.

This FCU has 17 Ports to connect UAV motors, GPS module, on board computer Raspberry Pi and telemetry radio signal etc.

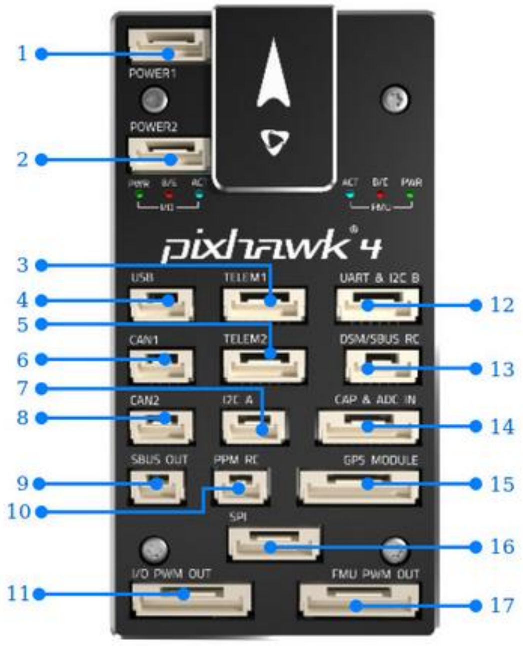


Fig. 4.3 Pixhawk FCU [10]

Fig has following ports:

1. Power module 1
2. Power module 2
3. Telemetry 1 (radio telemetry)
4. USB
5. Telemetry 2 (companion computer)
6. CAN1 bus
7. I2C
8. CAN2
9. S.BUS out for S.BUS servos
10. Radio Control Receiver Input (PPM)
11. Main outputs
12. UART and I2C
13. Radio Control Receiver Input (DSM/SBUS)
14. Input Capture and ADC IN
15. GPS module
16. SPI bus
17. AUX outputs (FMU PMU out)

4.3 4500 mAh Li-PO Power Battery

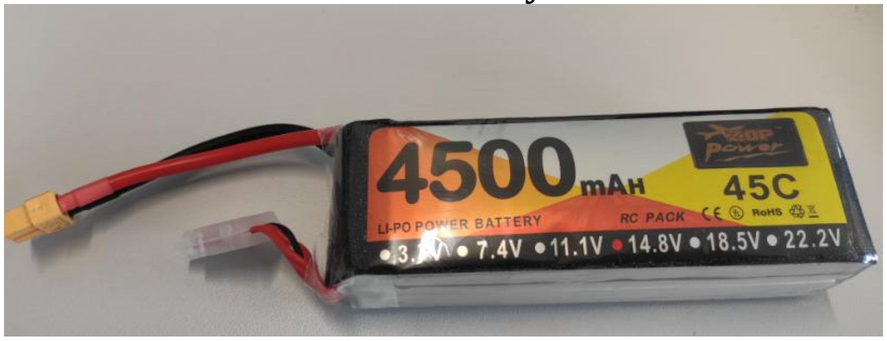


Fig. 4.4 Li-Po Battery

The system uses a 4500 45C mAh Li-Po battery which has a capacity of 4500 mAh and 45C discharging rate. The battery allows for 30-35 minutes flight time without any payload which is sufficient to complete the test.

4.4 HoTT MX-20 Graupner Radio Control Transmitter and receiver

The HoTT MX-20 Graupner 2.4 GHz Radio Control transmitter and receiver (Fig.5) are suitable for the UAV. The output voltage is 3.4-6.0 Volts. The RC dimensions are 190x175x115 mm. it can be used up to range 4000m. It can work in temperature 15-55 degree Celsius. The mx-20 has a total of 24 models on board. The memory can be expanded even more with a micro-SD card. The transmitter contains seven switches / buttons that can be allocated to any function. You can make a lot of modifications and programming with the 12 Mixer and 7 Programmable Flight Phases. The Graupner mx-20 HoTT is the largest transmitter in the mx family, with 12 channels. The up to 4 servos can be controlled simultaneously as a block with a signal repetition time of 10 ms.

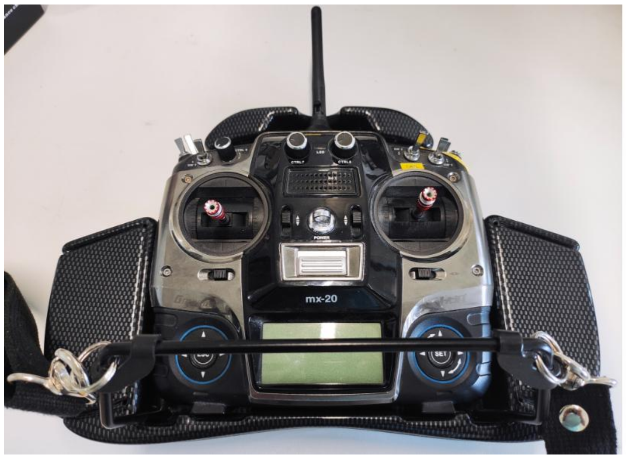


Fig. 4.5 RC Transmitter

4.5 Transceiver Telemetry Radio

Fig. 4.6 Telemetry Radio Cable

A Holybro Telemetry Radio is a compact, light, and low-cost open source radio platform that can generally achieve ranges of over 300 meters right out of the box (the range can be extended to several kilometers with the use of a patch antenna on the ground). The radio is powered by open source software that was created specifically to work with MAVLink packets and interface with the Mission Planner.

4.6 Microprocessor Raspberry Pi 4

Fig. 4.7 Raspberry Pi 4

The system uses the Raspberry Pi 4 computer which has 64-bit quad-core processor, up to 4 GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, USB 3.0, 2 x micro-HDMI ports, 40 Pin GPIO header, Power over Ethernet (PoE) enabled and micro SD card slot for loading operating system and data storage. The micro computer can be operated using different software such as Raspbian, Linux and Windows IoT. The Raspberry Pi 4 is using Ubuntu 18.04 version. The weighs of Raspberry Pi 4 is 45 grams which is suitable to use for an UAV. The reason behind using Raspberry Pi 4 is that it has better CPU i.e. better processing and performance speeds than Raspberry Pi 3B+.

4.7 Raspberry Pi Camera

The Raspberry Pi camera module version 2 as shown in fig 4.8 is 8 megapixel focus camera which supports 1080p, 720p and 360p. The camera dimensions are 25 x 24 x 9 mm and it attached via ribbon cable to the CSI port of Raspberry Pi 4. To capture frames for image processing, the camera can be accessed using the Pi camera Python module or the ROS environment.

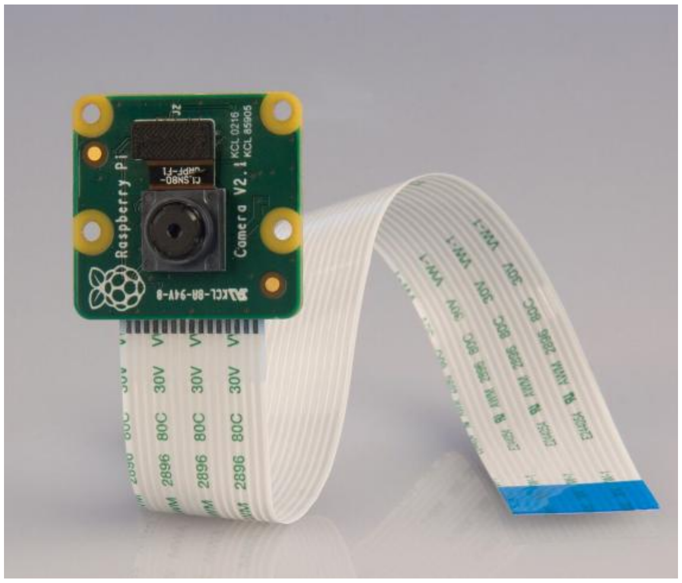


Fig. 4.8 Raspberry Pi Camera

4.8 StromPi Power and Battery Module

The StromPi enables the Raspberry Pi 4 with an optional plug-in battery unit, which is a rechargeable emergency power source to Raspberry pi 4 with a LiFePo4 battery. StromPi 3 allows the voltage between 6-61V and a current with up to 3A. [13]

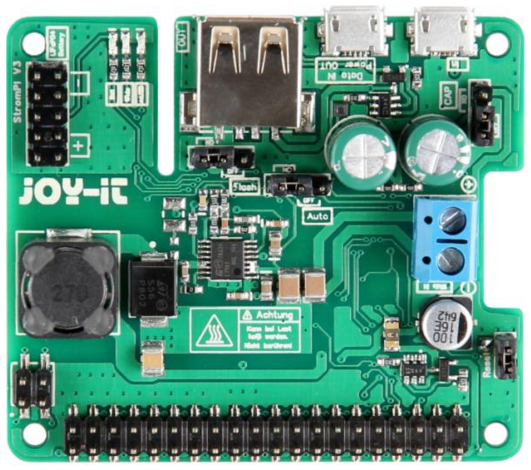


Fig. 4.9 (A) StromPi module [13]

Fig. 4.9 (B) Battery [13]

4.9 Intel Real Sense Depth Camera D435i

Intel RealSense Depth Camera has used to navigate the UAV according to AprilTag fiducial marker. The working of this camera has explained in next chapter. This camera combines the robust depth sensing capabilities of the D435 with the addition of an inertial measurement unit (IMU). By including an IMU in your application, you may improve your depth awareness in any case when the camera moves. It also allows robotics and drones to be more aware of their surroundings. The use of an IMU simplifies registration and calibration for handheld scanning systems, as well as in domains like virtual/augmented reality and drones.

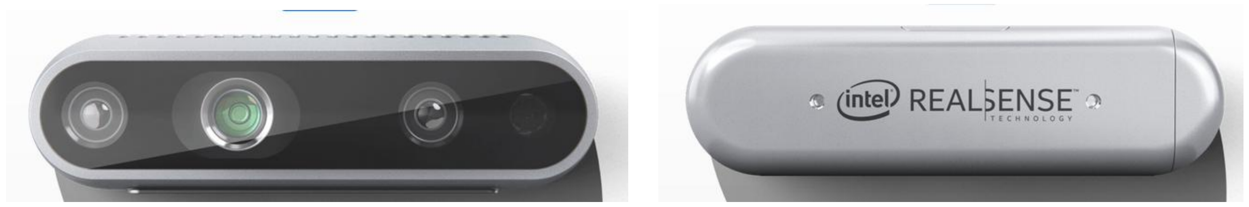


Fig. 4.10 Real-Sense Depth Camera

The inertial measurement unit (IMU) is used to detect 6 degrees of freedom movements and rotations (6DoF). An IMU is a device that detects rotation and movement in three axes, as well as pitch, yaw, and roll, using a combination of sensors and gyroscopes.

4.10 Wi-Fi Connection

A wifi network is used to connect the on board microcomputer i.e. Raspberry Pi 4 with Mission Planner software in order to give commands via SSH connection.

4.11 Mission Planner Software

The Mission Planner software is used on the ground station for this system to control the UAV. For autonomous vehicle, Mission Planner can be used as a configuration tool or as a dynamic control supplement. It’s a open source application which is developed by Michael Oborne for APM autopilot project. It can be used to load the firmware into Pixhawk FCU and setup, configure the UAV. Mission Planner can allow monitoring the UAV status while in working mode with telemetry radio signal that is connected to Mission Planner. Fig 4.11 shows the UAV status on map (green color), alarm status and some small windows with having values of Attitude, Yaw etc.

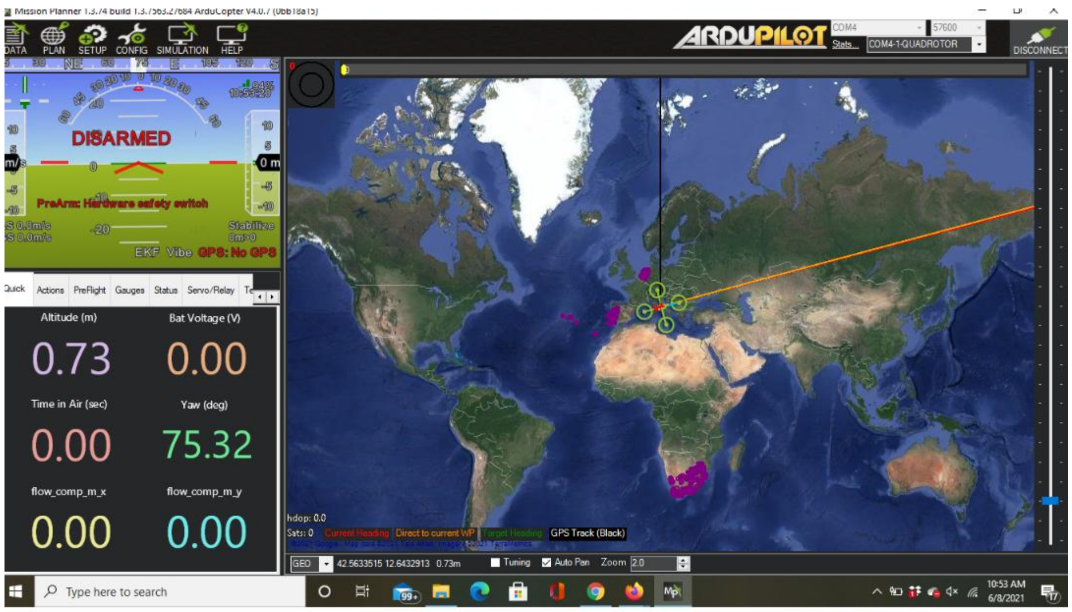


Fig. 4.11 Mission Planner Software

4.12 3D Printing Case for Raspberry Pi 4 microcomputer

Fig shows the 3D printing of an enclosure designed to secure and protect the Raspberry Pi 4, Raspberry Pi Camera, StromPi and Battery Module. The Autodesk Fusion 360 software has used to design case for Raspberry Pi 4 module. The 3D printed part consists of vertical holder to hold the top and bottom part. The Raspberry Pi camera installed by cutting square box and it can hold using four screws as shown in fig.B. Fig (c) shows the enclosure 3D printed case of Raspberry Pi to UAV.

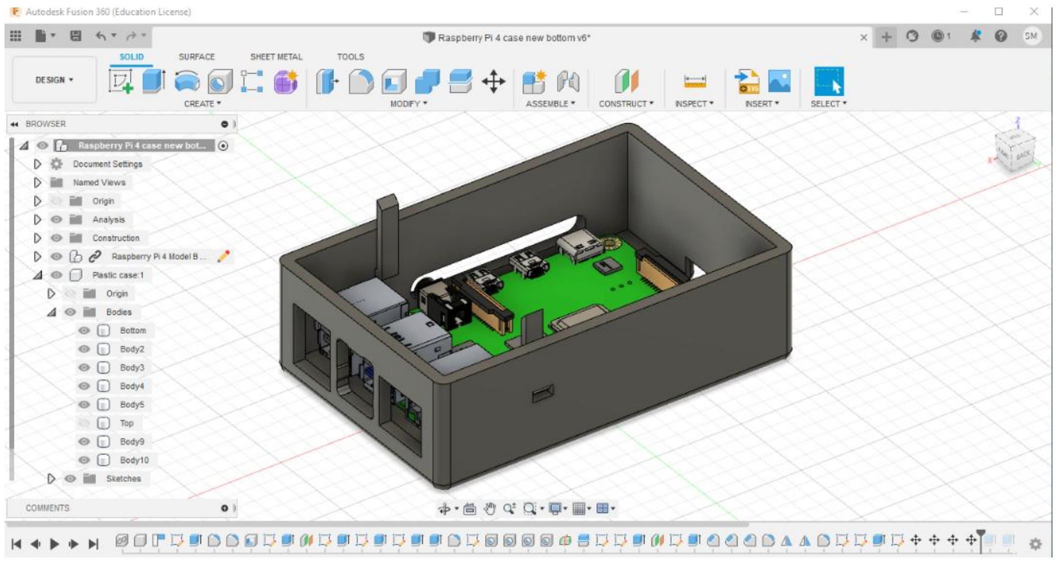


Fig. 4.12 (A) Raspberry Pi Case Design-1

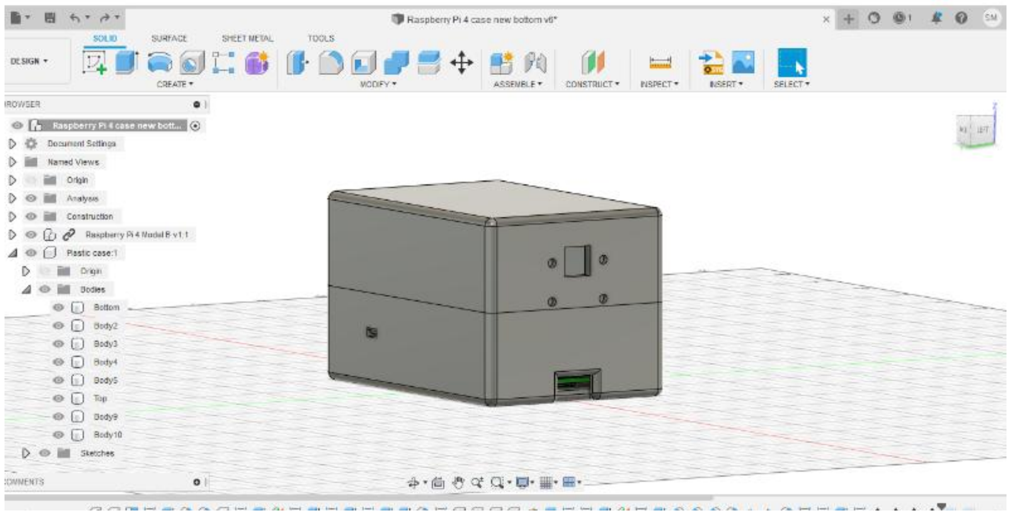


Fig. 4.12 (B) Raspberry Pi Case Design-2

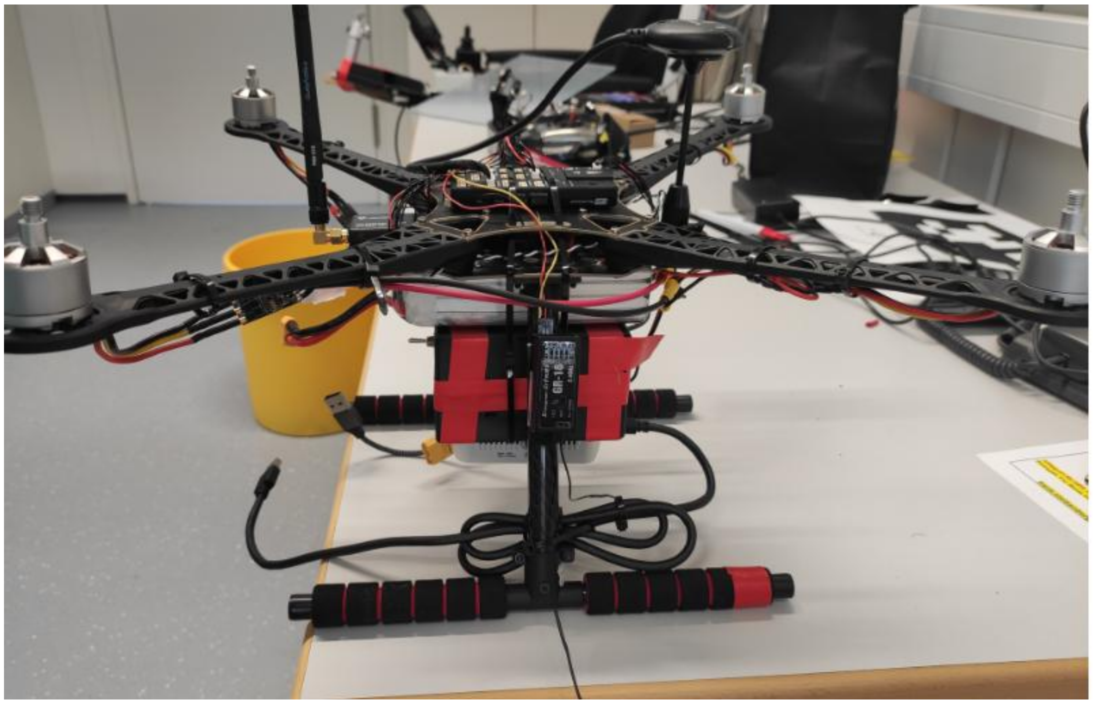


Fig. 4.12 (C) Raspberry Pi Case Mounted On UAV

Above fig shows that 3D printed case is attached to the drone. In that 3D printed case, Raspberry Pi 4, StromPi and Battery module has been placed. And on the bottom section of case, Real-Sense camera is attached.

Chapter 5
Implementation and Observation

As described in Chapter 1, the purpose of this research is to investigate and develop a system of vision based indoor navigation of an UAV.

Chapter 2 discussed the literature review and research objective and also the material and theoretical methods applied in this research.

Chapter 3 presents the computer vision approach to detect the predetermined objects and explained about various methods for detection of objects.

Chapter 4 describes the design of the system architecture for UAV. This chapter details the hardware and software has used for this research.

This chapter extends the observation of flight test for the detection of AprilTag marker and object detection. This chapter also presents the detection algorithm using computer vision approach, method of implementation of every single ROS nodes, steps to do for configure the UAV in Mission Planner software.

5.1 Detecting Customized Objects in Video Streams

The tensorflow API use to detect objects through Raspberry Pi Camera. There are several detection frameworks (refer section 3.2) are available to detect the object. Here, we have used Single Shot Detector (SSD) (see section 3.2.4) CNN framework.

Algorithm to detect the object using computer vision

The workflow of object detection as shown in fig.5.1. First, take some images that want to detect. Label it with appropriate object name. Then .xml file will create for each image. After that, generate the training data for converting “xml” files to “csv” file. After generating data, configure the training data by creating label map. Lastly, the model will train for 4-5 hours. (This work has already done in project module. This is a just a short overview).

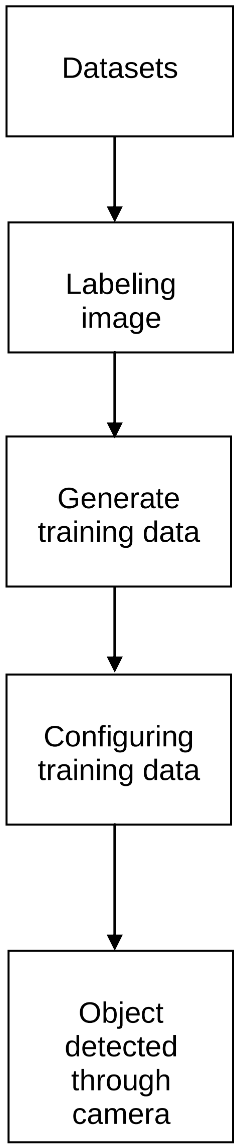


Fig. 5.1 Object Detection Pipeline

Hardware Configuration of UAV in Mission Planner ArduPilot

Before going to do further steps, some mandatory hardware configuration has to be done for the first time setup.

5.2 System Overview of ArduPilot Process

The fig 5.1 shows the basic working structure of ArduPilot firmware which is running on Mission Planner.

The software's main goal is to give the vehicle control, either autonomously or via pilot input via a radio control transmitter or ground control station, or via a companion computer on board the vehicle, all of which are optional, including only loading a fully autonomous mission on the vehicle for execution.

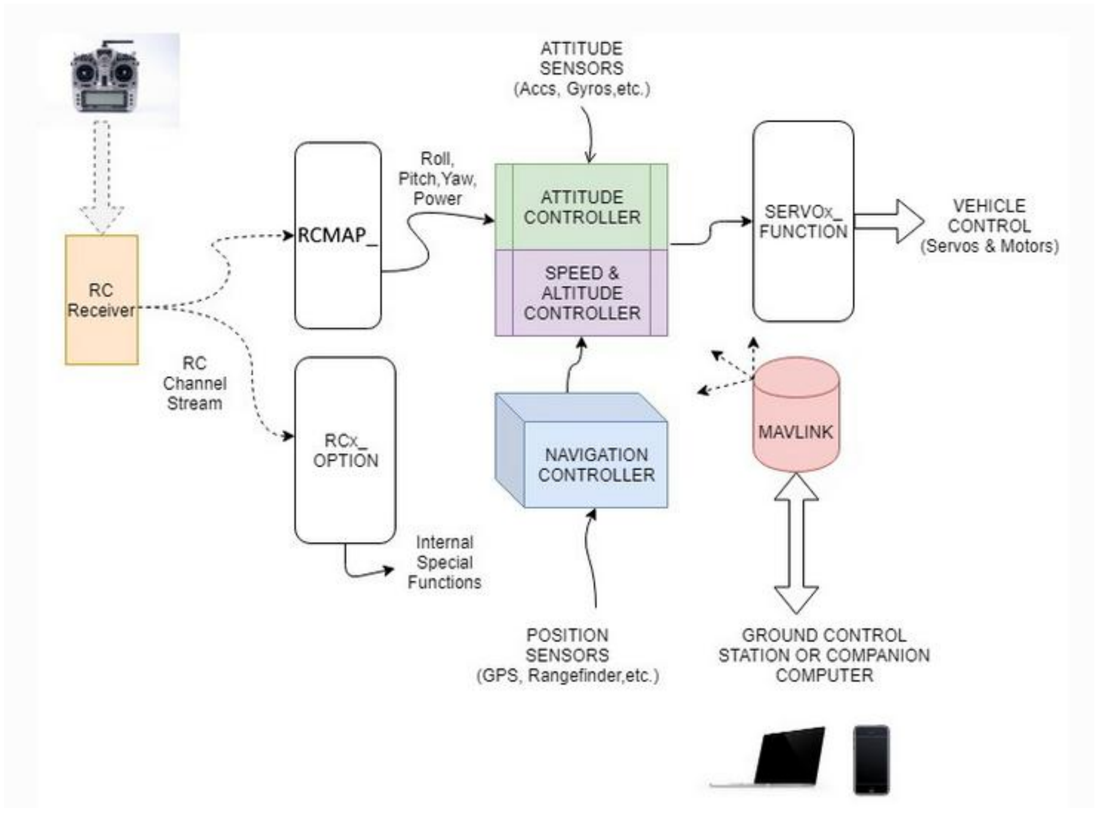

ATTITUDE SENSORS (Accs, Gyros,etc.)
Roll, Pitch,Yaw, Power
RCMAP_
ATTITUDE CONTROLLER
SPEED & ALTITUDE CONTROLLER
SERVOx_ FUNCTION
VEHICLE CONTROL (Servos & Motors)
RC Receiver
RC Channel Stream
RCx_ OPTION
Internal Special Functions
NAVIGATION CONTROLLER
MAVLINK
POSITION SENSORS (GPS, Rangefinder,etc.)
GROUND CONTROL STATION OR COMPANION COMPUTER

Fig 5.2 ArduPilot Process

From above figure, the RC receiver and MAVLink (see section 5.11) communication from telemetry are the control inputs from UAV. The Roll/Pitch/Yaw is getting input from RC receiver to control the attitude, throttle as well as operational modes of the UAV. The vehicles can control through the servos. Motors and relays which are the outputs of this system.

Sensor inputs to the autopilot give information such as attitude, position, power system monitoring, and vehicle speed. At least one or more accelerometers, baros, and gyros are built into every ArduPilot compliant autopilot.

5.3 Frame Selection Configuration

This section explains the selection of appropriate frame that will required for this project. Before going to select frame, first connect the Mission Planner software from computer to Pixhawk AutoPilot.

It can connect using USB cable, Telemetry radios, Bluetooth, IP connections etc. that shows in Fig 5.2 (A) and 5.2 (B).

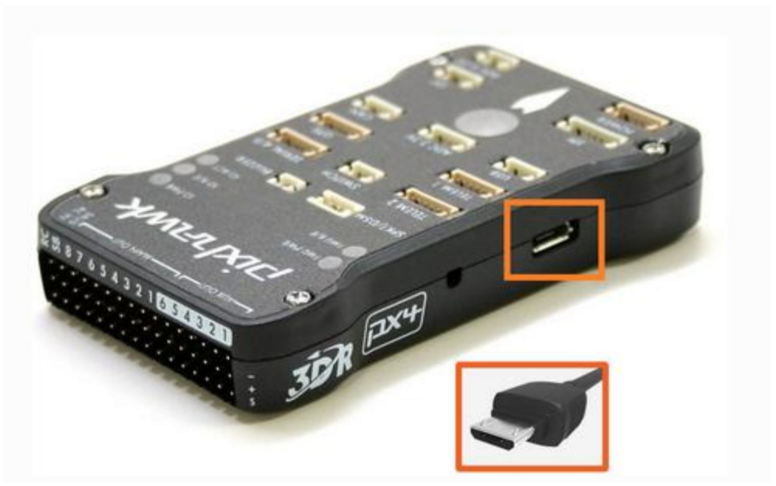


Fig. 5.3 (A) Connection through Pixhawk

Fig. 5.3 (B) Connection through telemetry radio

Now, on Mission Planner software, select the COM port and baud rate. Once USB cable or telemetry radio attached to computer then windows can automatically assign the COM port for autopilot. If the Telemetry Radio is attached to PC then the baud rate should be 57600 and for USB it must be 115200 as shown in fig. 5.3.

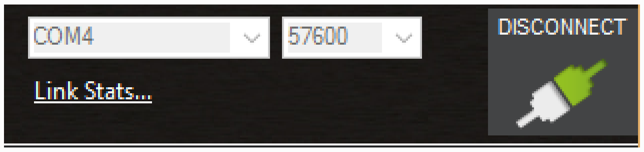
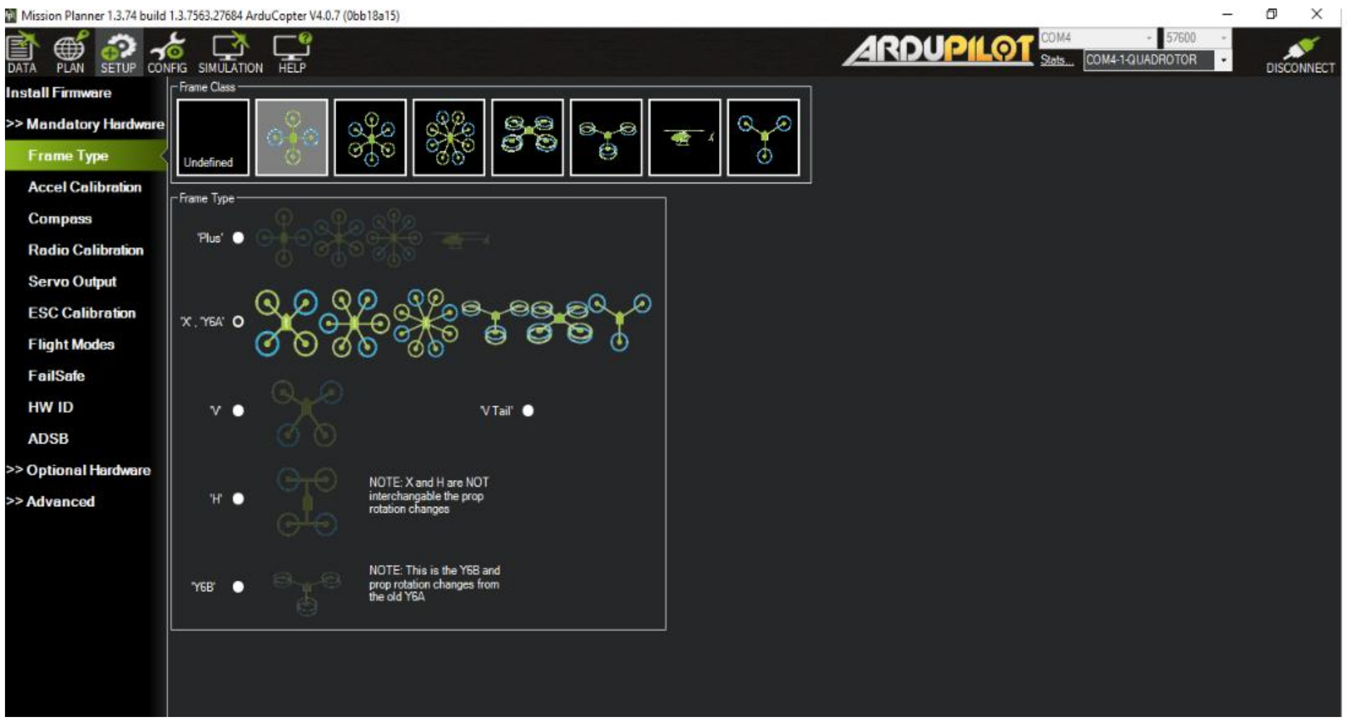


Fig 5.4 COM Port Connection

The next step would be selecting the required frame for UAV.

Steps to select frame:

- First, connect the Pixhawk to Mission planner at baud rate 54600
- Go to Initial Setup -> select Mandatory Hardware -> Frame type
- Select Frame Class Quadcopter as shown in fig 5.4
- In Frame Type, click on ‘X’, ‘Y6A’.


Mission Planner 1.3.74 build 1.3.7563.27684 ArduCopter V4.0.7 (0bb18a15)

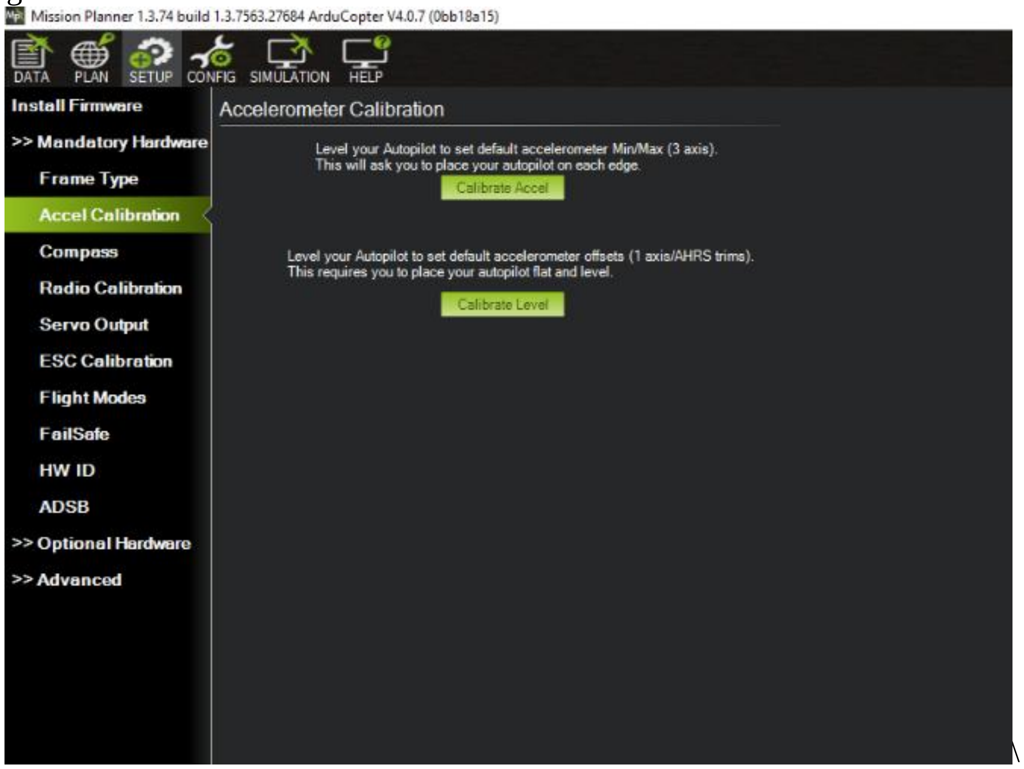
NOTE: X and H are NOT interchangeable the prop rotation changes

NOTE: This is the Y6B and prop rotation changes from the old Y6A

Fig. 5.5 Frame Selection Window

5.4 Accelerometer Calibration

This part presents the accelerometer calibration of the UAV. It is important to calibrate because it determines the position and orientation of the UAV. Keep the UAV at arrow on Pixhawk FCU is pointing forward direction.



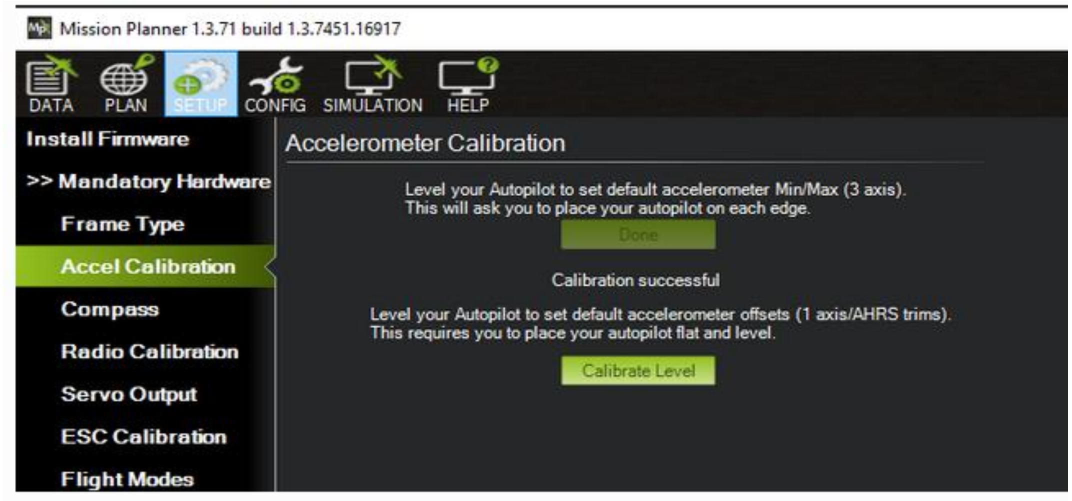
Level your Autopilot to set default accelerometer Min/Max (3 axis).
This will ask you to place your autopilot on each edge.

Level your Autopilot to set default accelerometer offsets (1 axis/AHRS trims).
This requires you to place your autopilot flat and level.

Fig 5.6 Accelerometer Calibration

Steps to do:

- Under Setup option, go to Accel Calibration from the left side menu.
- Click on Calibrate Accel to start the calibration process.
- There are different calibration positions that have to follow such as UAV at level, on right side, on left side, nose down, nose up, and back side.
- After every position it will ask for “Click when done”. Then click on that button and follow the same process for other positions.
- When all the calibration is completed for all the positions, it will display the message “Calibration Successful” as shown in fig. 5.6



Accelerometer Calibration

Level your Autopilot to set default accelerometer Min/Max (3 axis).
This will ask you to place your autopilot on each edge.

Calibration successful

Level your Autopilot to set default accelerometer offsets (1 axis/AHRS trims).
This requires you to place your autopilot flat and level.

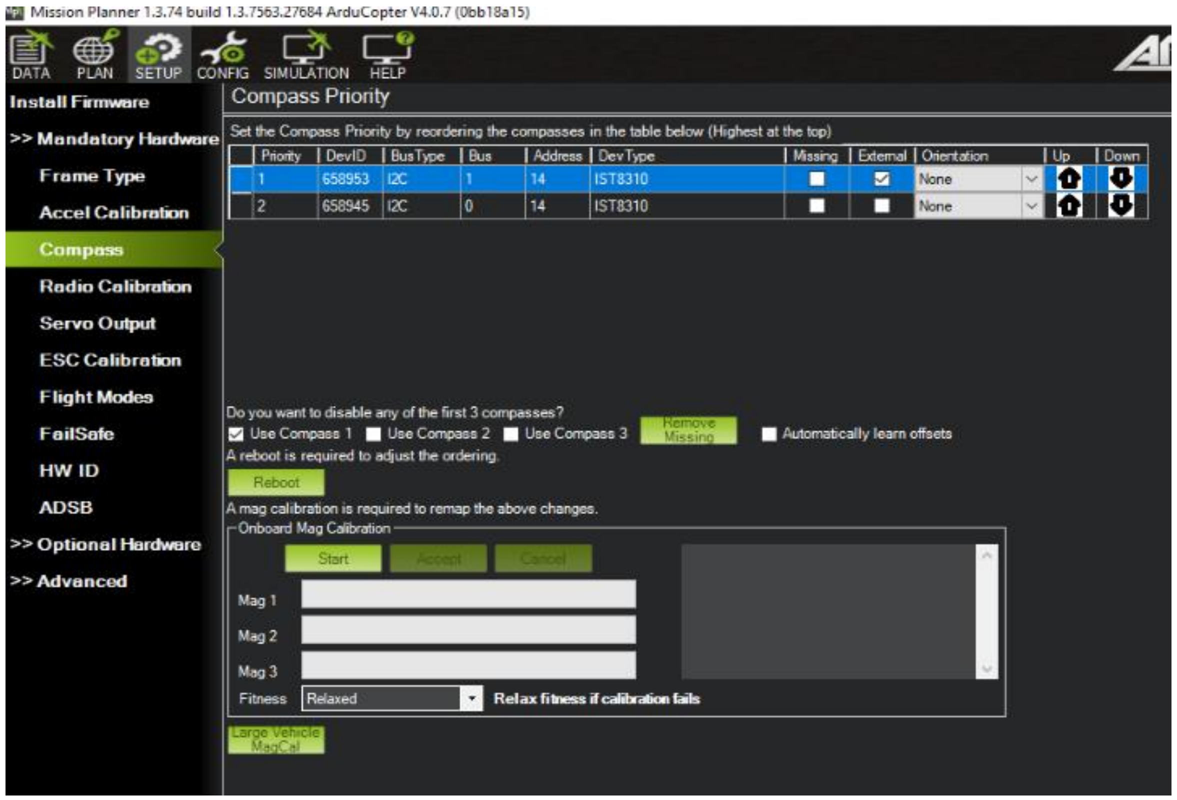
Fig 5.7 Accelerometer Calibration

5.5 Compass Calibration

The next is to calibrate the compass of UAV. For this, at least one compass must have internally or externally in the system.

Compass calibration steps (refer fig 5.7):

- In the Mandatory Hardware, go to option Compass.
- Click on “Onboard Mag Calibration” section’s “Start” button.
- Hold the UAV in the air and rotate in 360 degree angle (back, front, right, top, left and bottom).
- The green bars will extend as the UAV is rotating. The green bar will fill completely when calibration is complete.
- The three rising tone will be emitted after successful calibration.
- One pop-up window will appear with message “Please reboot the autopilot”. Then click on “Reboot” button. After this, the connection between MP and Pixhawk will go. Reconnect it for next RC Calibration.


Mission Planner 1.3.74 build 1.3.7563.27684 ArduCopter V4.0.7 (0bb18a15)

Compass Priority

Set the Compass Priority by reordering the compasses in the table below (Highest at the top)

Priority	DevID	BusType	Bus	Address	DevType	Missing	External	Orientation	Up	Down
1	658953	I2C	1	14	IST8310	☐	✓	None	⬆	⬇
2	658945	I2C	0	14	IST8310	☐	☐	None	⬆	⬇

Do you want to disable any of the first 3 compasses?
[✓ Use Compass 1] ☐ Use Compass 2 ☐ Use Compass 3 Remove Missing ☐ Automatically learn offsets
A reboot is required to adjust the ordering.
Reboot
A mag calibration is required to remap the above changes.
Onboard Mag Calibration
Start Accept Cancel
Mag 1
Mag 2
Mag 3
Fitness: Relaxed **Relax fitness if calibration fails**
Large Vehicle MagCal

Fig 5.8 Compass Calibration

5.6 Radio Control Calibration

The pilot can use an RC transmitter to control the vehicle's movement and direction, as well as turn on and off auxiliary functions (i.e. raising and lowering landing gear, etc).

RC Calibration entails collecting the minimum, maximum, and "trim" values for each RC input channel so that ArduPilot can correctly interpret the input.

RC Transmitter Setup:

- Battery must be disconnected from UAV as accident can happen of the UAV is armed during RC calibration process.
- For connecting autopilot, switch on the button of RC receiver.
- After connecting transmitter and receiver, the green bars should appear in calibration section.
- Check the channel mapping in the transmitter by moving the sticks on RC.
- Identify the transmitter using Mode1 and Mode2 as shown in fig 5.9.

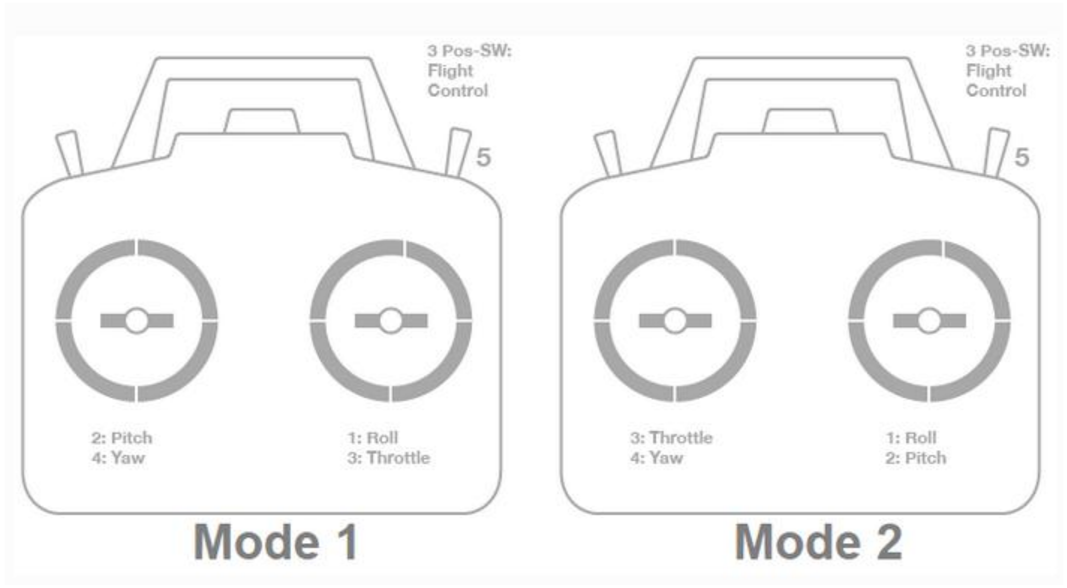


Fig 5.9 Different modes on RC

- Mode 1: left stick controls pitch and yaw, the right stick will control throttle and roll.
- Mode 2: left stick controls throttle and yaw; the right stick will control pitch and roll.
- Channel 1 is for Roll, channel 2 is for Pitch, Channel 3 is for Throttle and Channel 4 is for Yaw.

Calibration Process:

- In mandatory hardware, click on “Radio Calibration”. Then click on “Calibrate Radio” green box.
- Before click on “Calibrate Radio” button, make sure that battery and propellers should not connected to UAV
- Move the transmitter’s control sticks, knobs and switches to their limits. Red lines will appear across the calibration bars to show minimum and maximum values as shown in fig. 5.10.

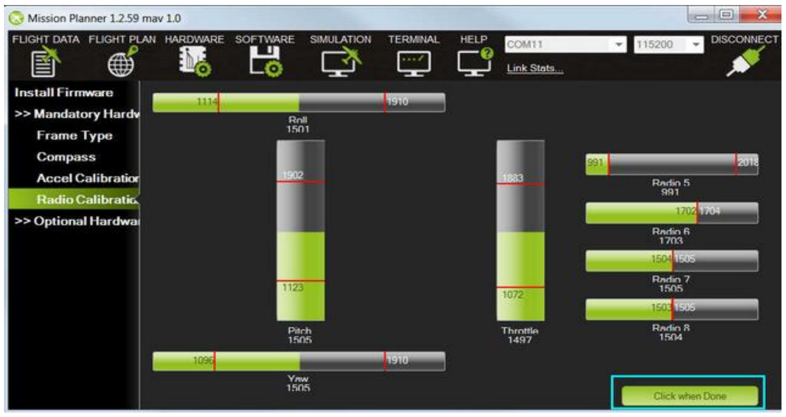


Fig. 5.10 RC Calibration

- When every stick has get minimum and maximum values then click on “Click when Done” button.
- Then press “OK” to ensure that throttle is down to 0 and other sticks are at centered position.
- The values of different channels are between minimum 1100 to maximum 1900.

Now, move the transmitter’s roll, pitch, throttle and yaw sticks and ensure the green bars move in the correct direction (fig. 5.10):

- The green bars for the roll, throttle, and yaw channels should move in the same direction as the transmitter's physical controls.
- The green bar should move in the opposite direction of the transmitter's physical stick when it comes to pitch.
- Reverse the channel in the transmitter if one of the green bars travels in the wrong direction. If reversing the channel in the transmitter is not possible, user can do so in ArduPilot by checking the "Reversed" checkbox [11].

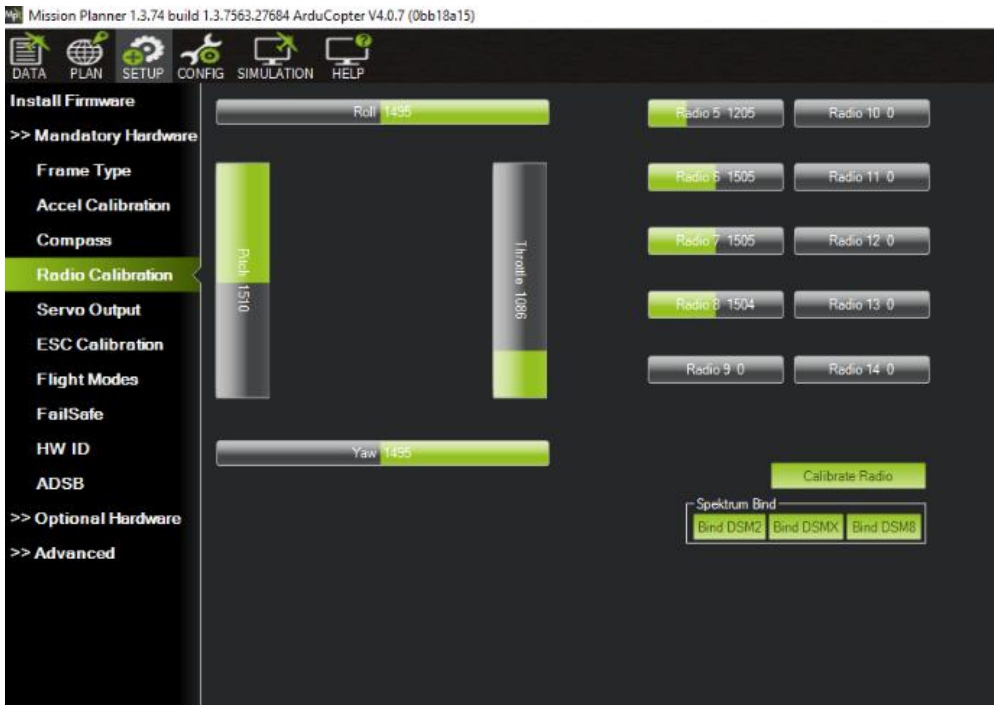


Fig. 5.11 RC Calibration

5.7 Electronic Speed Controller (ESC) Calibration

Electronic speed controllers are in charge of spinning the motors at the autopilot's specified speed. Most ESCs must be calibrated in order to understand the minimum and maximum PMW values sent by the flight controller.

Before doing ESC calibration makes sure that

- LiPo Battery is disconnected
- Propellers are removed from UAV
- FCU is not connected to MP via USB cable

a) Turn on the RC transmitter and keep throttle at maximum as shown below.

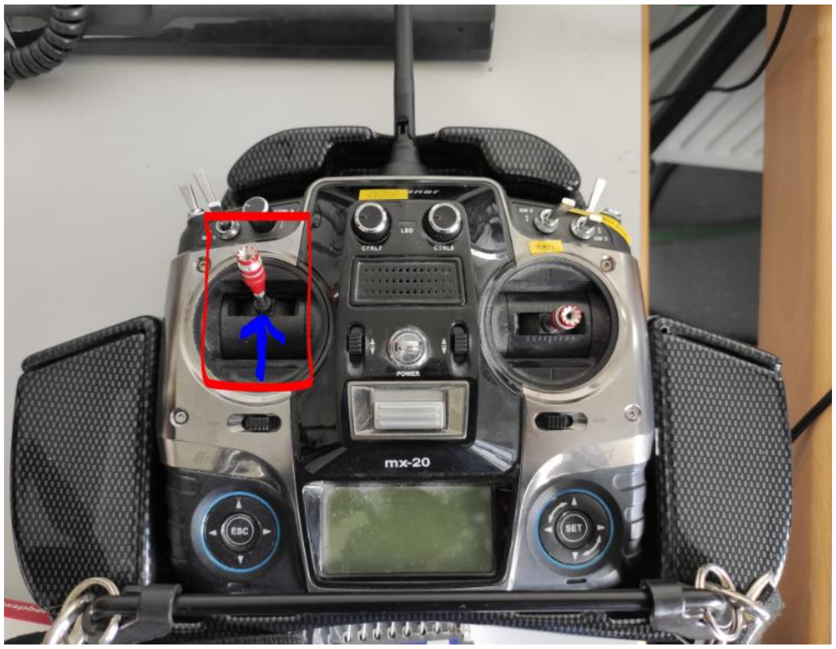


Fig. 5.12 ESC Calibration

b) Connect the Li-Po battery to UAV. The red, blue, and yellow LEDs on the autopilot will light up in a cyclical pattern.

c) Then disconnect the battery and reconnect it. Press the safety switch until it gives single tune. Now autopilot is in ESC calibration mode. Throttle stick is still high.

d) After some seconds, it will give some musical tone. Additional two beeps shows maximum throttle has captured by autopilot.

e) Now, set throttle stick to minimum position as in Fig. 5.12.

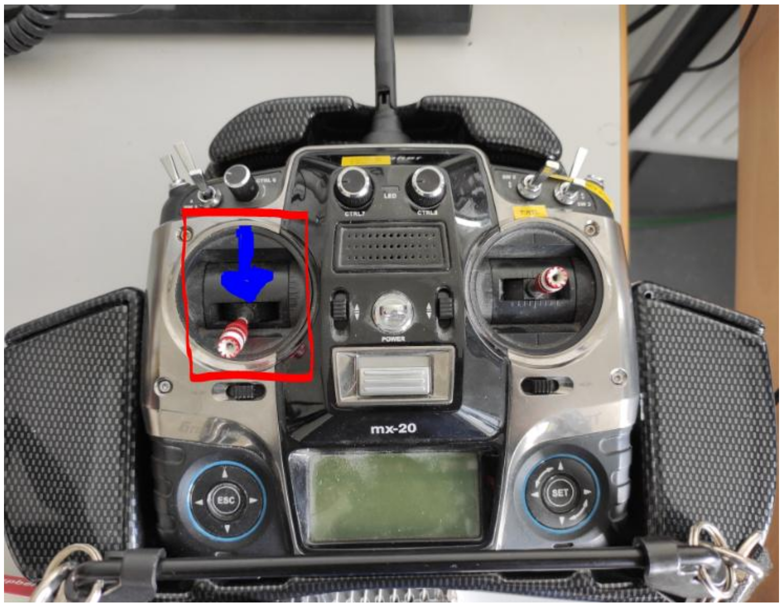


Fig. 5.13 ESC Calibration

f) The ESC will gives long tone which shows the minimum throttle has been captures and ESC calibration is successfully done.
g) Take a trial to rotate the motors. Give some throttle and motors start running and adjust the speed through throttle stick.

5.8 Mission Planner Parameters

After doing this all configurations stuff, connect the Pixhawk FCU to Mission Planner software through telemetry radio channel. Some parameters need to upload.

Go to "CONFIG" menu. Click on "Full Parameter List" and write below parameters.

AHRS_EKF_TYPE = 2
BRD_RTC_TYPES = 2
EK2_ENABLE = 1
EK3_ENABLE = 0
EK2_GPS_TYPE = 3
EK2_POSNE_M_NSE = 0.1
EK2_VELD_M_NSE = 0.1
EK2_VELNE_M_NSE = 0.1
GPS_TYPE = 0
COMPASS_USE = 0
COMPASS_USE2 = 0
COMPASS_USE3 = 0
SERIAL5_BAUD = 921 (the serial port used to connect to Raspberry Pi)

SERIAL5_PROTOCOL = 1
SYSID_MYGCS = 1 (to accept control from mavros)
VISO_TYPE = 0

When all this parameters are uploaded, click on “Write Parameter” on the right side of the screen.

5.9 Robot Operating System (ROS)

ROS is open source software which provides libraries and tools and an easy environment in which to develop robotic applications. It's a set of tools, libraries, and conventions aimed at making building complicated and reliable robot behavior easier on a range of robotic systems. Because of their reliance on enormous collections of open-source software dependencies, the primary ROS client libraries are designed toward a Unix-like environment. Ubuntu Linux is categorized as "Supported" for these client libraries, whereas other variations such as Fedora Linux, macOS, and Microsoft Windows are listed as "experimental" and are supported by the community.

ROS packages consist of several nodes. A node is a process that performs computation. Using streaming topics, the Parameter Server, nodes are connected into a graph and communicate with one another. Suppose, one node control the Raspberry Pi camera, one node control the navigation of UAV and other node control the localization and detection etc. Additional nodes can communicate with each other utilizing topics thanks to the roscore node (process). Any data type can be used as a topic, including Boolean variables, integer variables, float variables, pictures, and custom types. Nodes can be run through the launch files which have specific parameters to launch that particular file. In this system, the Raspberry Pi camera is attached to UAV and Real Sense camera (see section 4.9) is mounted on top section of UAV for navigation. These nodes are developed including a navigation node, camera node, detection node to detect the AprilTag marker (see section 5.12) and objects. The Transform (tf) library in the ROS gives the multiple coordinate frames over time to the user. Here, nodes can use this library to build time-stamped representations of the locations and orientations of the robot's key physical components. These transforms are given names and grouped into a tree, from which they can be combined to create new transformations. This can be used to determine the positions of components on the robot as well as objects in space. In RVIZ, users can visualize the tf of Real Sense camera and AprilTag marker.

5.10 MAVROS

MAVROS is a ROS package that enables MAVLink extendable communication between computers running ROS for any ground station. It's officially bridge between ROS and ArduPilot by translating ROS topics into MAVLink messages.

Features of MAVROS:

- Communication with autopilot via serial port, UDP or TCP (e.g. PX4 Pro or ArduPilot)
- Plug-in system for ROS-MAVLink translation
- Parameter manipulation tool
- Waypoint manipulation tool
- PX4Flow support (by mavros_extras)
- OFFBOARD mode support
- Geographic coordinates conversions.

5.11 Communication of Raspberry Pi 4 to ArduPilot with MAVLink

MAVLink is a messaging protocol for the drone communication and between onboard drone components. Messages are defined within XML files. With the use of MAVLink, offboard and onboard communications are both possible between GCS and drones, and between drone autopilot.

Before connecting the MAVLink to Raspberry Pi, some settings in Raspberry Pi configuration has to be done.

If the first time Raspberry Pi is connecting to MAVLink then configure the serial UART port from the Raspi configuration utility by giving command “raspi-config”. In that, select “Interfacing Options” -> “Serial”.

Then it will ask two statements:

1. “Would you like a login shell to be accessible over serial?”, click on **no**.
2. “Would you like the serial port hardware to be enabled?”, click on **yes.**

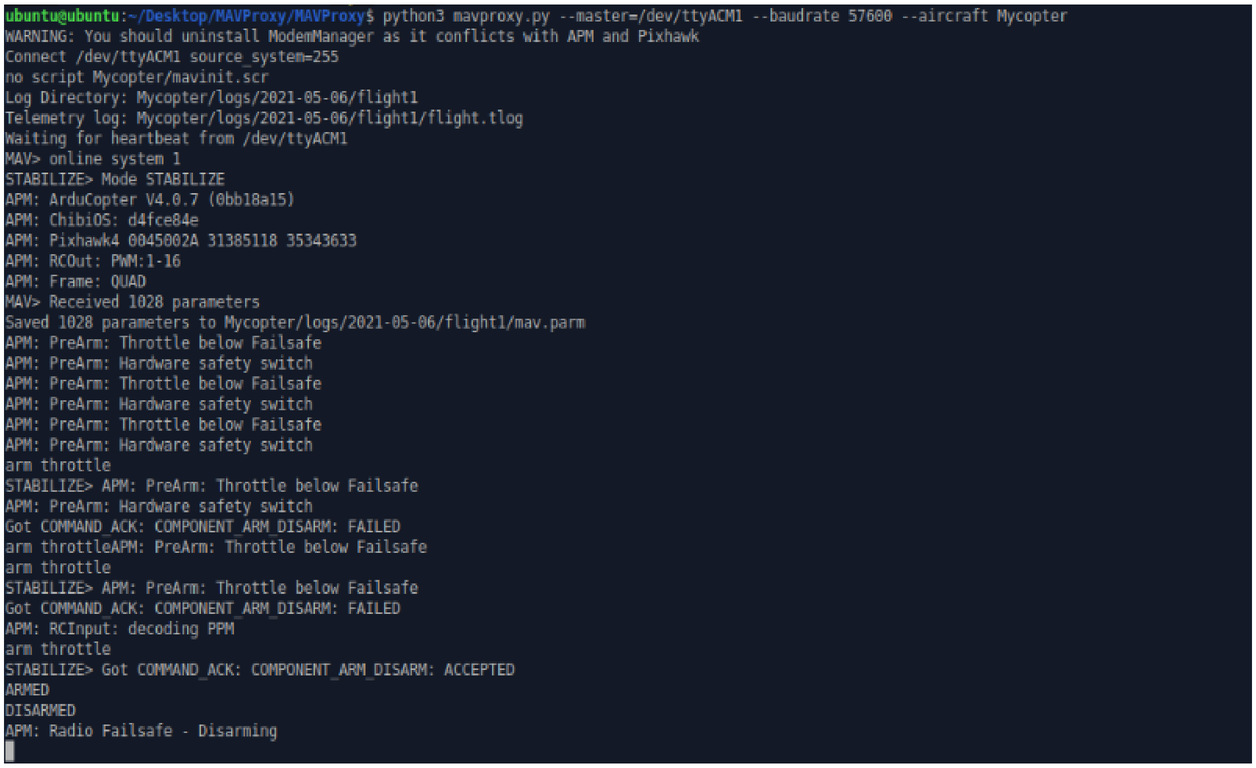
After this, reboot the Raspberry Pi to save these settings.
To connect the MAVLink to flight controller, MAVProxy can be used to send commands to the flight controller from the Pi. MAVProxy is portable and it can run on POSIX OS with python, pyserial etc.

To ensure that the Raspberry Pi 4 and flight controller can interact with one another, make sure they are both powered up. Give the following command:

- python3 mavproxy.py --master=/dev/serial0 --baudrate 57600 --aircraft MyCopter

MAVProxy commands are written using Python script which allows MAVProxy to be interfaced with other library software such as OpenCV.

After running the command, Raspberry Pi 4 is communicated to Pixhawk FCU through telemetry radio cable with baudrate 57600 as discussed in section 5.2.2. Make sure that band rate matches with baudrate is in Mission Planner GCS. When the communication of Raspberry Pi and Pixhawk is successfully established, it shows that FCU is in STABILIZE mode as it is set in RC transmitter (refer fig. 5.13). After that, FCU will receive parameters and showing the warning or error messages such as Throttle below Failsafe, Hardware safety switch etc. If user arms the UAV from Mission Planner software then it can give the message as “ARMED”.



```
ubuntu@ubuntu:~/Desktop/MAVProxy/MAVProxy$ python3 mavproxy.py --master=/dev/ttyACM1 --baudrate 57600 --aircraft Mycopter
WARNING: You should uninstall ModemManager as it conflicts with APM and Pixhawk
Connect /dev/ttyACM1 source_system=255
no script Mycopter/mavinit.scr
Log Directory: Mycopter/logs/2021-05-06/flight1
Telemetry log: Mycopter/logs/2021-05-06/flight1/flight.tlog
Waiting for heartbeat from /dev/ttyACM1
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: ArduCopter V4.0.7 (0bb18a15)
APM: ChibiOS: d4fce84e
APM: Pixhawk4 0045002A 31385118 35343633
APM: RCOut: PWM:1-16
APM: Frame: QUAD
MAV> Received 1028 parameters
Saved 1028 parameters to Mycopter/logs/2021-05-06/flight1/mav.parm
APM: PreArm: Throttle below Failsafe
APM: PreArm: Hardware safety switch
APM: PreArm: Throttle below Failsafe
APM: PreArm: Hardware safety switch
APM: PreArm: Throttle below Failsafe
APM: PreArm: Hardware safety switch
arm throttle
STABILIZE> APM: PreArm: Throttle below Failsafe
APM: PreArm: Hardware safety switch
Got COMMAND_ACK: COMPONENT_ARM_DISARM: FAILED
arm throttleAPM: PreArm: Throttle below Failsafe
arm throttle
STABILIZE> APM: PreArm: Throttle below Failsafe
Got COMMAND_ACK: COMPONENT_ARM_DISARM: FAILED
APM: RCInput: decoding PPM
arm throttle
STABILIZE> Got COMMAND_ACK: COMPONENT_ARM_DISARM: ACCEPTED
ARMED
DISARMED
APM: Radio Failsafe - Disarming
```

Fig. 5.14 MAVLink Communication to Raspberry Pi

5.12 AprilTag Marker Detection

AprilTag [14] (refer https://github.com/AprilRobotics/apriltag_ros) is a visual fiducial system which is used in camera calibration, augmented reality and robotics. It computes the 3D position, orientation and identifies the marker relative to camera. These tags are similar to QR codes. The marker is basically consisting of black square with white foreground that can create a particular pattern. Every marker has its own pattern that defines its identity (see fig. 5.14). In below figure, Tag3611 is the class of AprilTag marker identity.

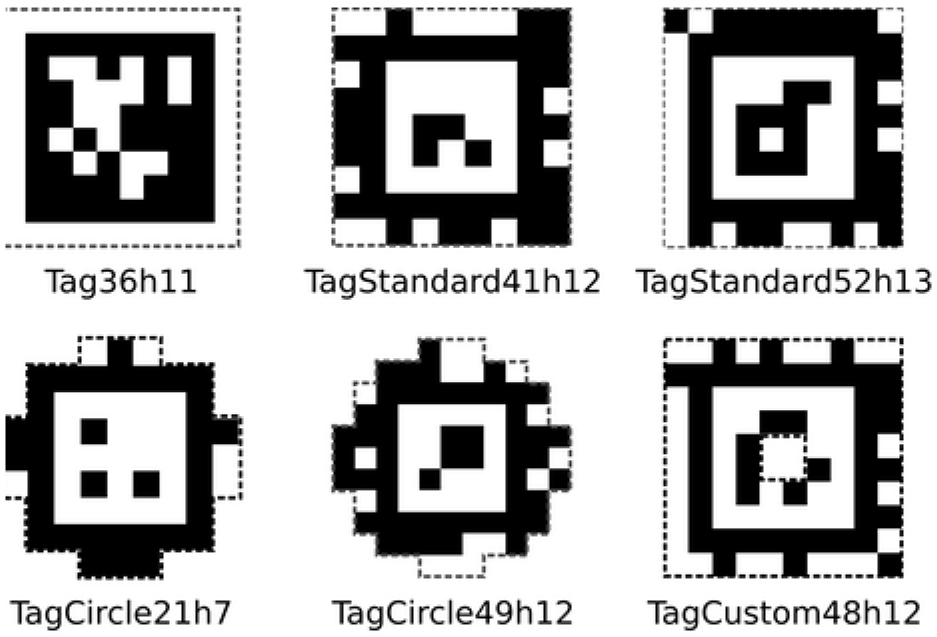
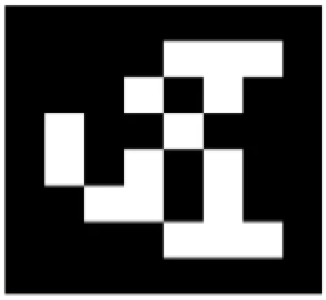
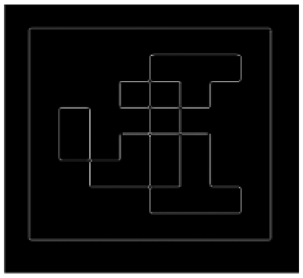


Fig. 5.15 Different AprilTag Markers

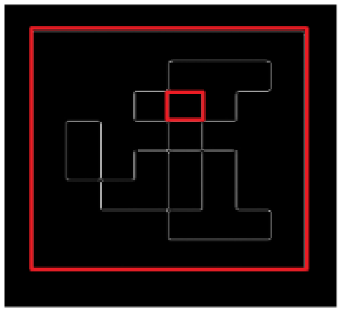
Let’s take an example to understand this detection algorithm. Suppose input image of AprilTag maker is of class 36H10.



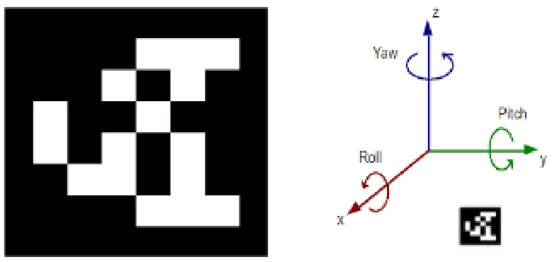
Then it calculates the magnitude and direction of a gradient at each pixel in an image with the AprilTag.



Following that, using a graph-based technique, these estimated gradients are sorted into components termed clusters based on common gradient properties. A line is fitted to every component using a weighted least square technique, so that the direction of the gradients affects the direction of the component.



At last, the all shapes are detected after getting the lines. Using homography and intrinsic estimation over an extracted tag, a 6-DOF pose of the tag in the camera frame of reference is also returned.



5.13 Vision Inertial Capture Systems for Position Estimation

Visual Inertial Odometry (VIO) system allows UAV to navigate when global position is absent or unstable in indoor environment. Pose data from either type of system can be utilized to update the local location estimate (relative to the local origin) of a PX4-based autopilot, as well as optionally fused into the vehicle attitude estimation. Furthermore, if the external pose system also supplies linear velocity measurements, they can be employed to improve the state estimate.

“vision_to_mavros” node will listen the output of AprilTag pose in camera frame and publish the body pose to a topic that can remapped to topic “/mavros/vision_pose/pose [12].

vision_to_mavros [12] is a transformation of tf pose data to NED frame for vision based MAVROS topics using Real-Sense camera. The structure of the system is shown in fig. 5.15.

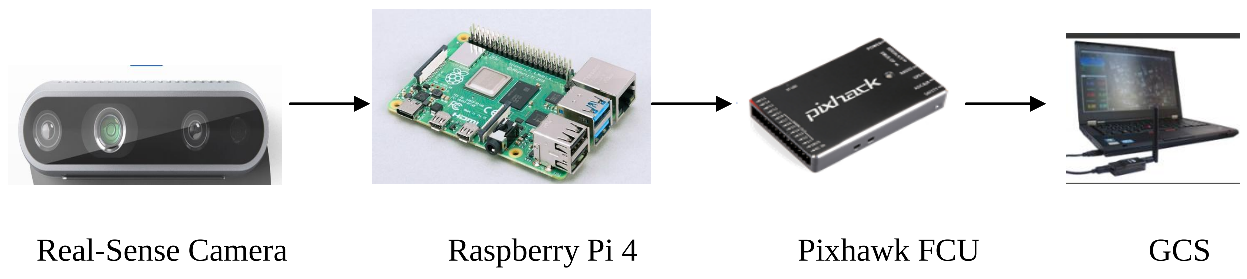


Fig. 5.16 Hardware Connection

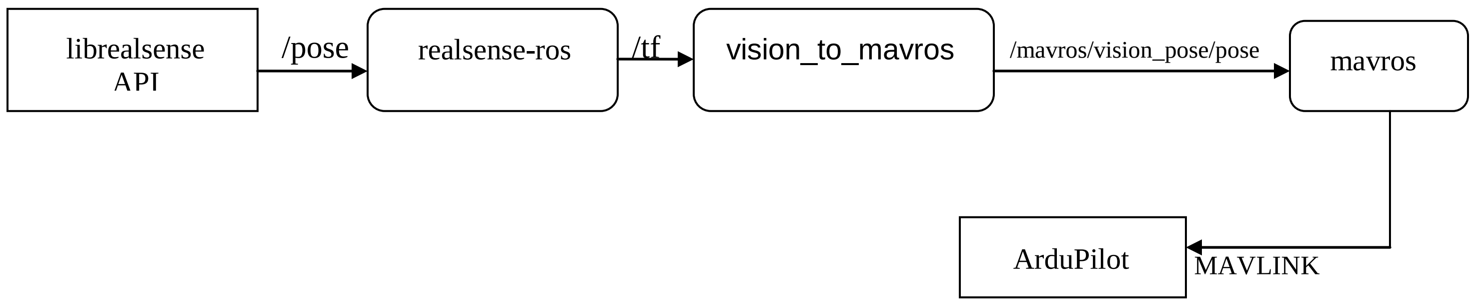


Fig. 5.17 Dataflow

There are several topics that can be used to communicate external navigation data to ArduPilot using MAVROS such as “VISION_POSITION_ESTIMATE” which accepts Euler angles, Roll, Pitch and Yaw.
The position data in /tf will be converted and sent to MAVROS via the subject /mavros/vision

pose/pose. The ENU - NED frames transition will be handled by Mavros and sent to ArduPilot over MAVLink.

5.13.1 Getting Pose Data Into ROS

PX4 employs FRD for both the local body frame and the reference frame (X Forward, Y Right, and Z Down). When using the magnetometer's heading, the PX4 reference frame's x axis will be aligned with north, hence the name NED (X North, Y East, Z Down). Depending on the source of the reference frame, the user must apply a specific transformation to the pose estimation before sending the MAVLink Vision. This is required in order to align the source and target frames of the posture estimation in accordance with the PX4 protocol.

Working of ENU and NED frames:

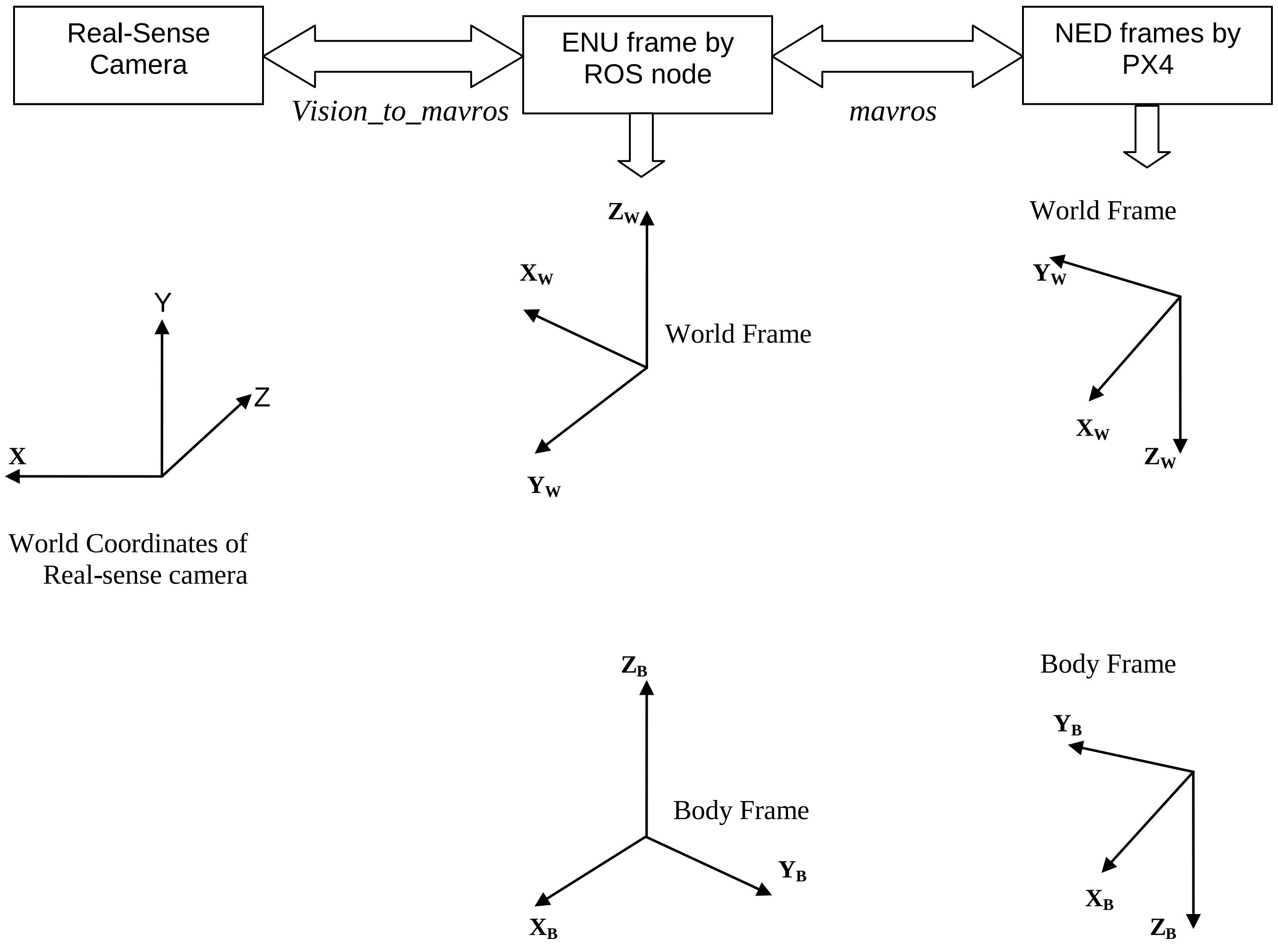


Fig. 5.18 ENU and NED frames Used by ROS and FCU

Position feedback must be supported in ENU frame as ROS uses ENU frames i.e. FLU (X = Forward, Y = Left, Z = Up) and PX4 FCU uses NED i.e. FRD (X = Forward, Y = Right, Z = Down) frame. As mentioned in 5.7.1, there are two frames called source_frame_id and target_frame_id.

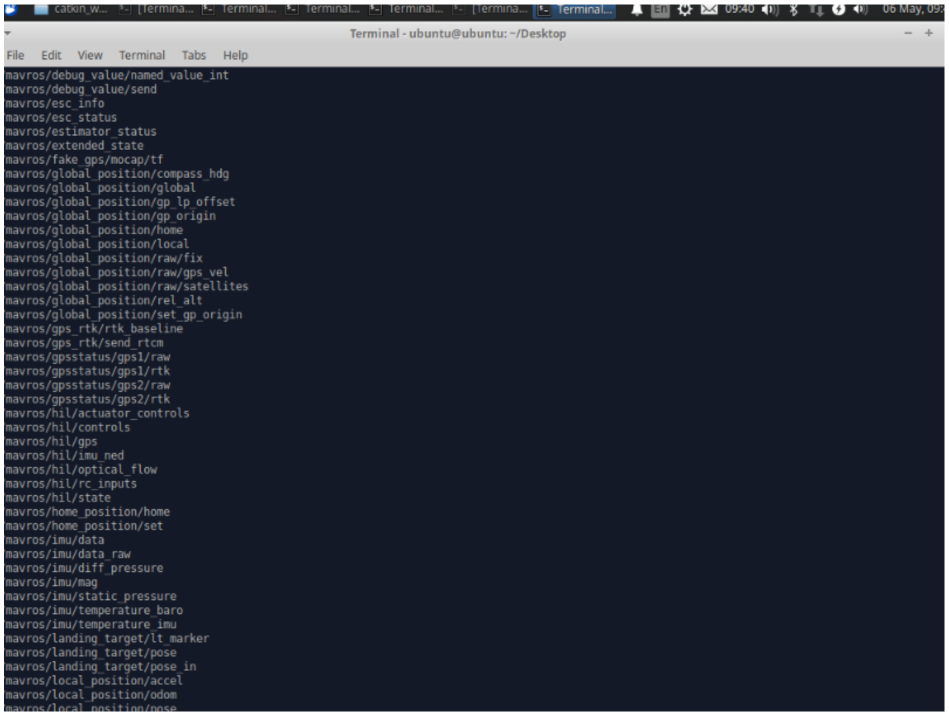
target_frame_id would be the world {W} frame and transformation of source_frame_id to body {B} frame is needed to get pose estimation. Source frame id will be aligned with that default B by rotating around its own x, y, and z axes by angles defined by roll cam, pitch cam, and yaw cam, in that order.

In this system, AprilTag marker is fitted at selling and Real-Sense camera is facing upward direction so position of Z axis will be change. And these changes have done vision_to_mavros.cpp file.

In the vision_to_mavros package, the "tf_to_mavros.launch" file has source_frame_id and target_frame_id which is marker and camera respectively.

This package basically contains 3 main functions:

- **/tf:** It is the transformation information that allows the user to track various coordinate frames across time. This type of data is available out of the box in many localization packages. Tf basically use for to convert the ROS topics between real-sense ros and mavros i.e. /mavros/vision_pose/pose topic. Fig. 5.17 shows topic list.



```
mavros/debug_value/named_value_int
mavros/debug_value/send
mavros/esc_info
mavros/esc_status
mavros/estimator_status
mavros/extended_state
mavros/fake_gps/mocap/tf
mavros/global_position/compass_hdg
mavros/global_position/global
mavros/global_position/gp_lp_offset
mavros/global_position/gp_origin
mavros/global_position/home
mavros/global_position/local
mavros/global_position/raw/fix
mavros/global_position/raw/gps_vel
mavros/global_position/raw/satellites
mavros/global_position/rel_alt
mavros/global_position/set_gp_origin
mavros/gps_rtk/rtk_baseline
mavros/gps_rtk/send_rtcm
mavros/gpsstatus/gps1/raw
mavros/gpsstatus/gps1/rtk
mavros/gpsstatus/gps2/raw
mavros/gpsstatus/gps2/rtk
mavros/hil/actuator_controls
mavros/hil/controls
mavros/hil/gps
mavros/hil/imu_ned
mavros/hil/optical_flow
mavros/hil/rc_inputs
mavros/hil/state
mavros/home_position/home
mavros/home_position/set
mavros/imu/data
mavros/imu/data_raw
mavros/imu/diff_pressure
mavros/imu/mag
mavros/imu/static_pressure
mavros/imu/temperature_baro
mavros/imu/temperature_imu
mavros/landing_target/lt_marker
mavros/landing_target/pose
mavros/landing_target/pose_in
mavros/local_position/accel
mavros/local_position/odom
mavros/local_position/pose
```

Fig. 5.19 MAVROS topic list

Topics can also read through “rqt_graph”. rqt_graph provides a GUI plug-in for visualizing the ROS computation graph(refer fig. 5.18). The topics are present in rectangle and nodes are in circle shape.

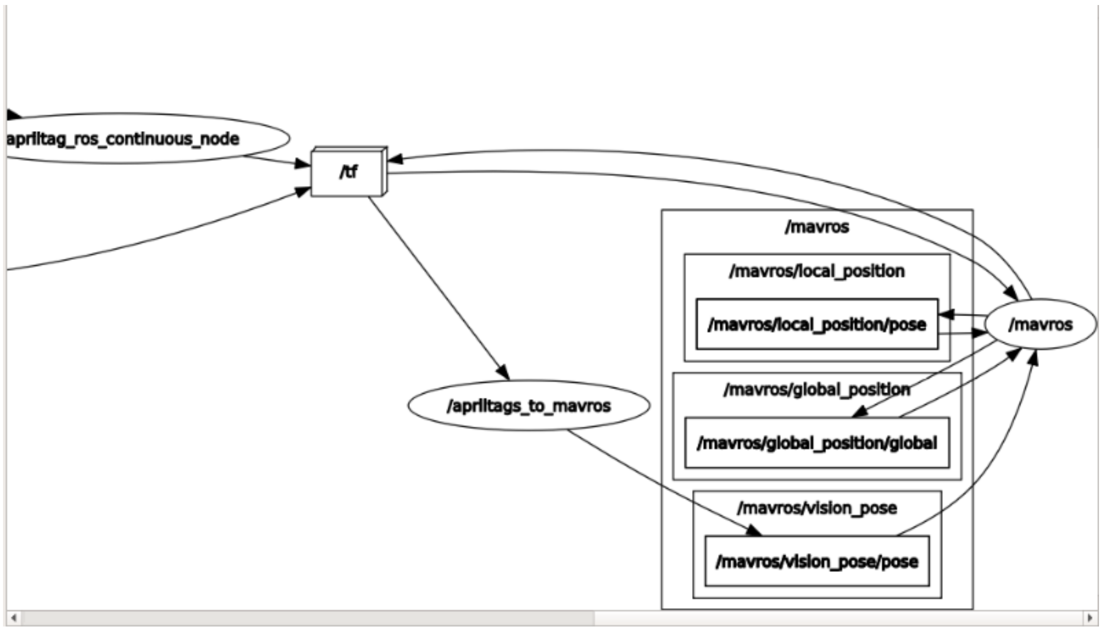


Fig. 5.20 rqt_graph

- **Output rate:** This parameter gives the output rate at which the pose data will be published. FCU can handle 30 Hz data rates. This data rate can change in tf_to_mavros.launch file [12].

- **Frame Alignment:** The position feedback from real-sense ROS is in ENU convention. Here, marker is placed at above the camera. So, the position of the Z axis will change i.e. it will pointed at downward direction.

5.14 Working Protocol of the System

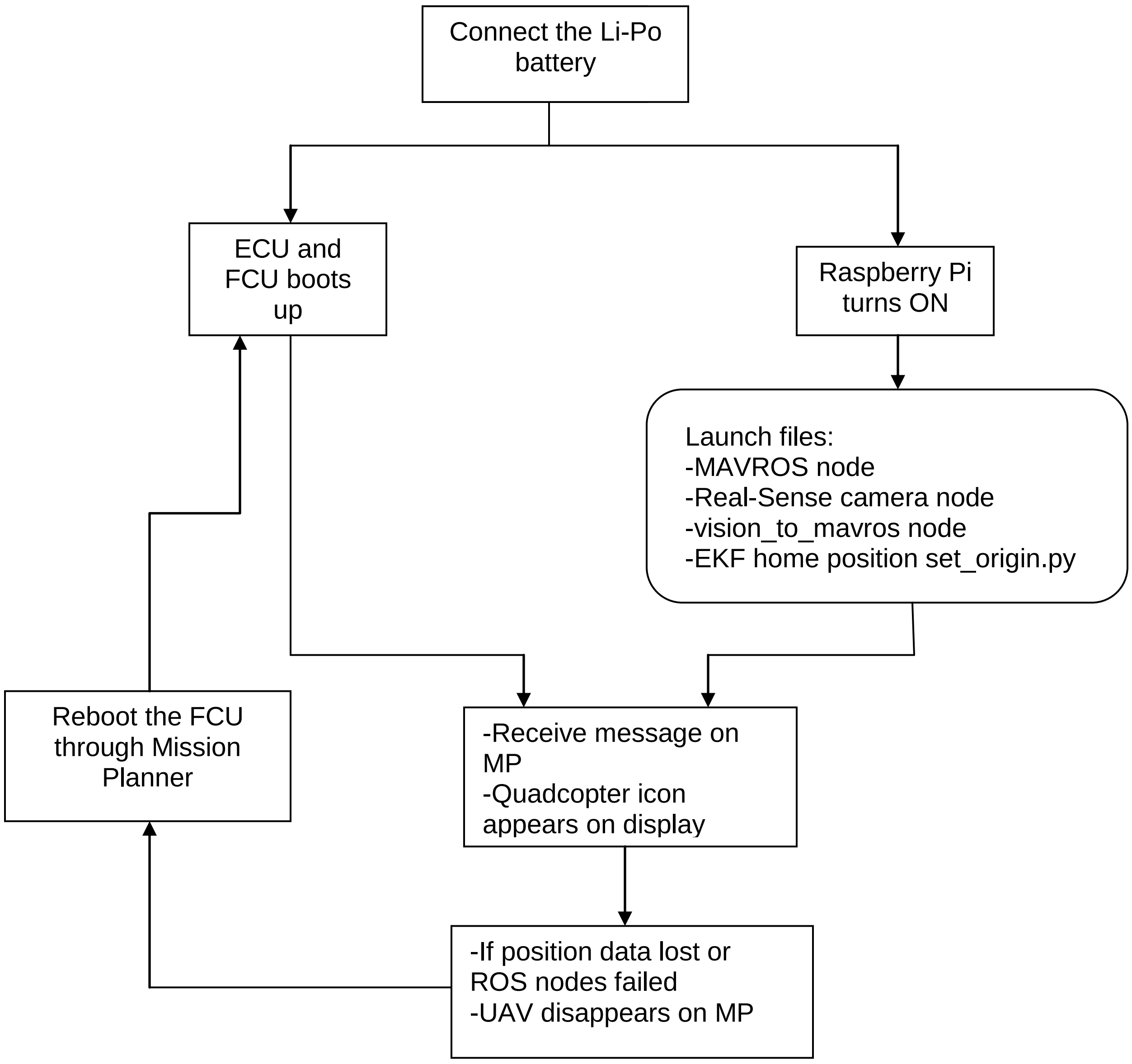

Connect the Li-Po battery
ECU and FCU boots up
Raspberry Pi turns ON
Launch files:
-MAVROS node
-Real-Sense camera node
-vision_to_mavros node
-EKF home position set_origin.py
Reboot the FCU through Mission Planner
-Receive message on MP
-Quadcopter icon appears on display
-If position data lost or ROS nodes failed
-UAV disappears on MP

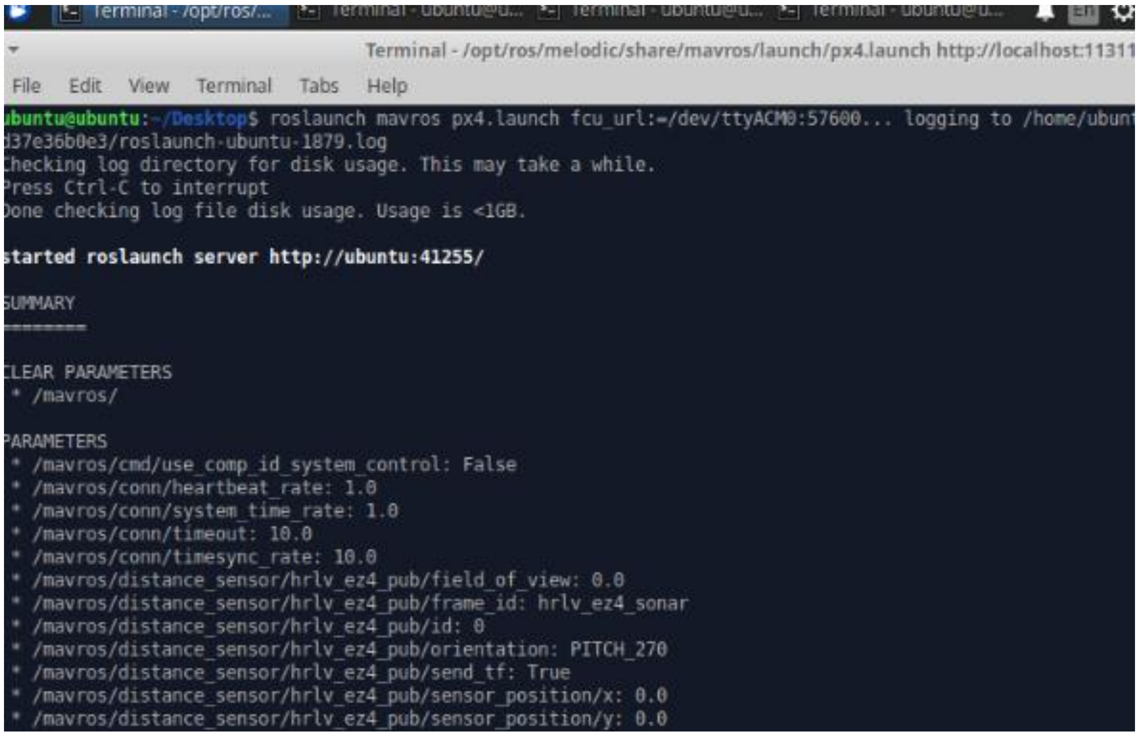
Fig. 5.21 Workflow of the system

Launching all the ROS nodes:

To test the system, three main ROS nodes must be run on three separate terminals of Raspberry Pi 4.

1. **MAVROS Node:**

Give the command “roslaunch px4.launch fcu_url:<ttyACM0:57600” on terminal 1 as shown in fig. 5.20


Terminal - /opt/ros/...
Terminal - ubuntu@u...
Terminal - ubuntu@u...
Terminal - ubuntu@u...
En
Terminal - /opt/ros/melodic/share/mavros/launch/px4.launch http://localhost:11311
File Edit View Terminal Tabs Help

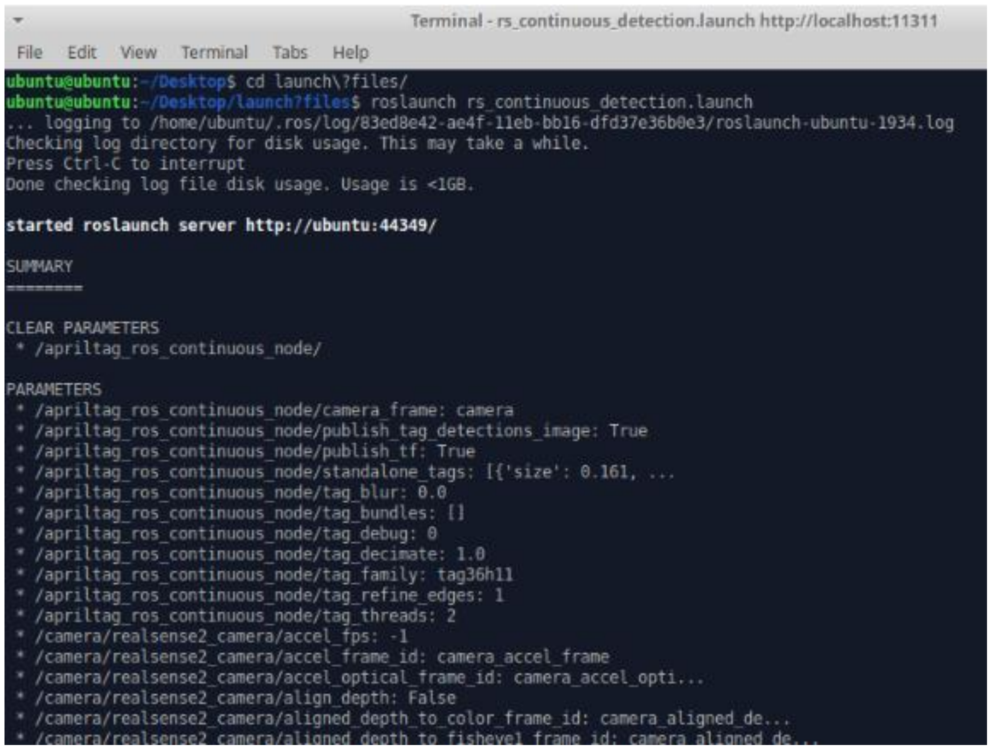
```
buntu@ubuntu:~/Desktop$ roslaunch mavros px4.launch fcu_url:=/dev/ttyACM0:57600... logging to /home/ubun
d37e36b0e3/roslaunch-ubuntu-1879.log
hecking log directory for disk usage. This may take a while.
ress Ctrl-C to interrupt
one checking log file disk usage. Usage is <1GB.

tarted roslaunch server http://ubuntu:41255/

UMMARY
=======

LEAR PARAMETERS
 * /mavros/

ARAMETERS
 * /mavros/cmd/use_comp_id_system_control: False
 * /mavros/conn/heartbeat_rate: 1.0
 * /mavros/conn/system_time_rate: 1.0
 * /mavros/conn/timeout: 10.0
 * /mavros/conn/timesync_rate: 10.0
 * /mavros/distance_sensor/hrlv_ez4_pub/field_of_view: 0.0
 * /mavros/distance_sensor/hrlv_ez4_pub/frame_id: hrlv_ez4_sonar
 * /mavros/distance_sensor/hrlv_ez4_pub/id: 0
 * /mavros/distance_sensor/hrlv_ez4_pub/orientation: PITCH_270
 * /mavros/distance_sensor/hrlv_ez4_pub/send_tf: True
 * /mavros/distance_sensor/hrlv_ez4_pub/sensor_position/x: 0.0
 * /mavros/distance_sensor/hrlv_ez4_pub/sensor_position/y: 0.0
```

Fig. 5.22 MAVROS launch file

Verify that MAVROS is running properly by giving command “/mavros/status” which shows the FCU is connected or not.

2. **Real-Sense Camera Node:**

On terminal 2 of Raspberry Pi: “roslaunch rs_continous_detection.launch”.
The topic /tf will published at 200 Hz. (refer fig. 5.20). Make sure that camera is located below the marker.



Terminal - rs_continuous_detection.launch http://localhost:11311
File Edit View Terminal Tabs Help

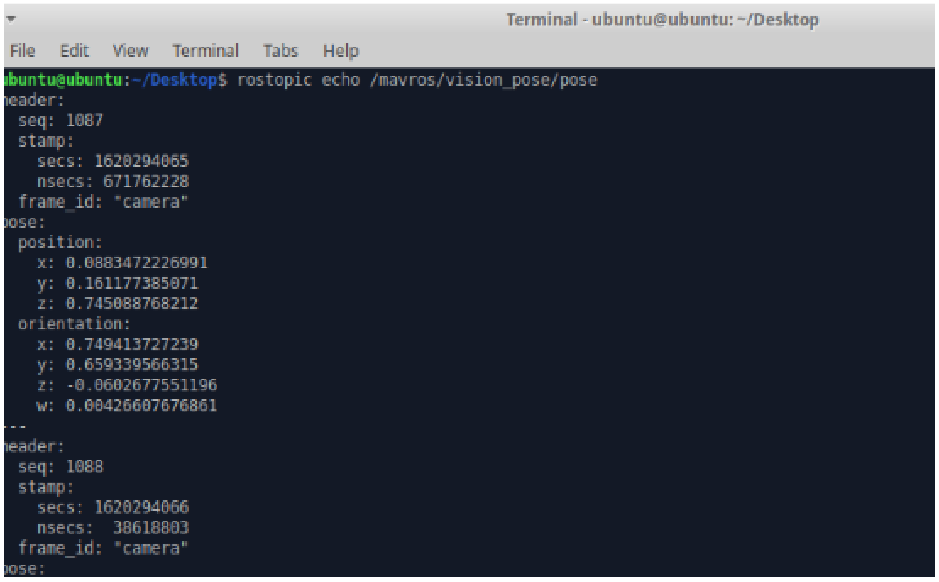
```
ubuntu@ubuntu:~/Desktop$ cd launch\?files/
ubuntu@ubuntu:~/Desktop/launch?files$ roslaunch rs_continuous_detection.launch
... logging to /home/ubuntu/.ros/log/83ed8e42-ae4f-11eb-bb16-dfd37e36b0e3/roslaunch-ubuntu-1934.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:44349/

SUMMARY
========

CLEAR PARAMETERS
 * /apriltag_ros_continuous_node/

PARAMETERS
 * /apriltag_ros_continuous_node/camera_frame: camera
 * /apriltag_ros_continuous_node/publish_tag_detections_image: True
 * /apriltag_ros_continuous_node/publish_tf: True
 * /apriltag_ros_continuous_node/standalone_tags: [{'size': 0.161, ...
 * /apriltag_ros_continuous_node/tag_blur: 0.0
 * /apriltag_ros_continuous_node/tag_bundles: []
 * /apriltag_ros_continuous_node/tag_debug: 0
 * /apriltag_ros_continuous_node/tag_decimate: 1.0
 * /apriltag_ros_continuous_node/tag_family: tag36h11
 * /apriltag_ros_continuous_node/tag_refine_edges: 1
 * /apriltag_ros_continuous_node/tag_threads: 2
 * /camera/realsense2_camera/accel_fps: -1
 * /camera/realsense2_camera/accel_frame_id: camera_accel_frame
 * /camera/realsense2_camera/accel_optical_frame_id: camera_accel_opti...
 * /camera/realsense2_camera/align_depth: False
 * /camera/realsense2_camera/aligned_depth_to_color_frame_id: camera_aligned_de...
 * /camera/realsense2_camera/aligned_depth_to_fisheye1_frame_id: camera_aligned_de...
```

Fig. 5.23 Real-Sense camera launch file

3. AprilTag node and vision_to_mavros node:

On the 3rd terminal: “roslaunch vision_to_mavros tf_to_mavros.launch”.

The command “rostopic echo /mavros/vision_pose/pose will show the pose data from Real-Sense camera. (See fig. 5.21)



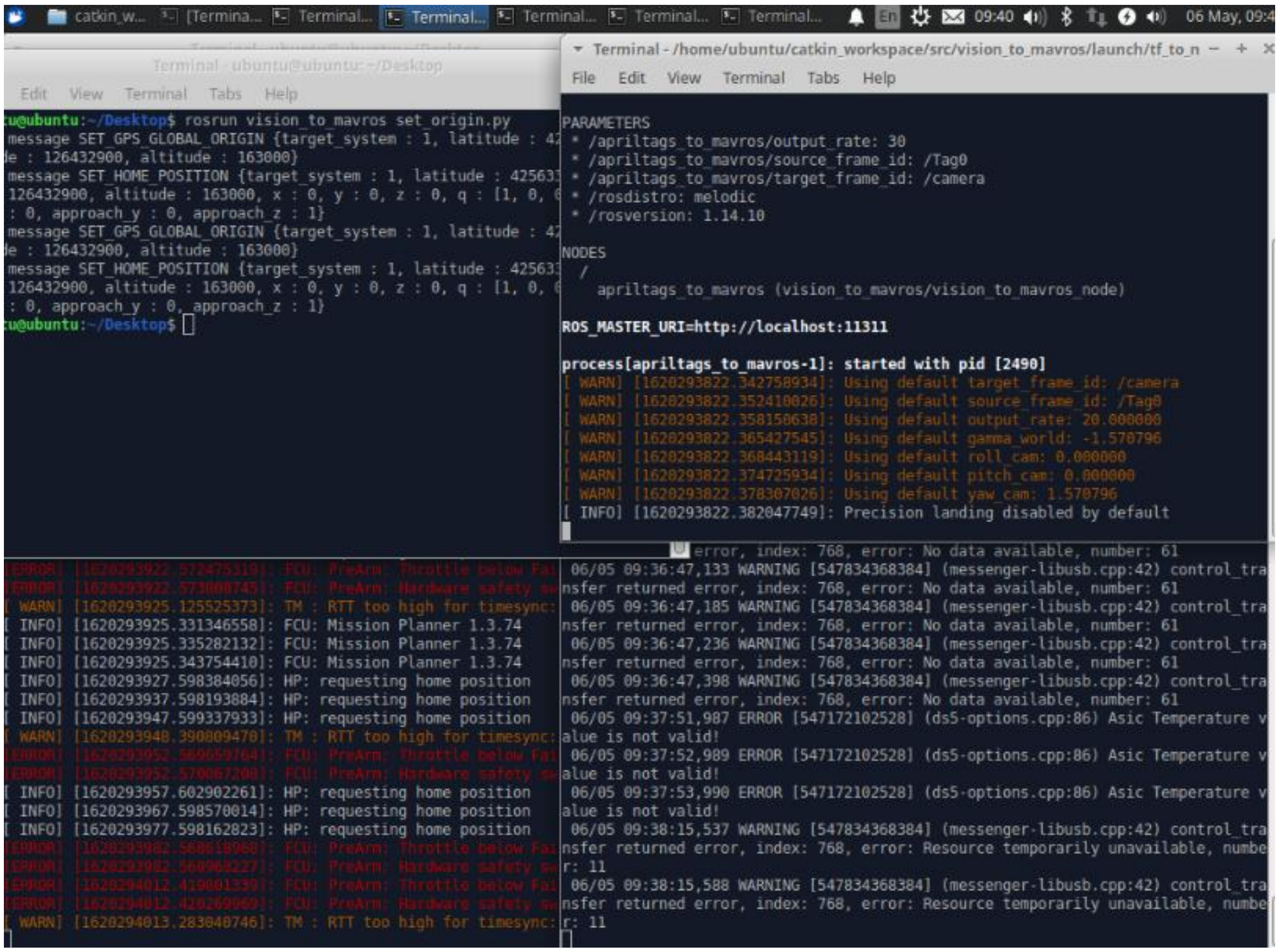
Terminal - ubuntu@ubuntu: ~/Desktop
File Edit View Terminal Tabs Help

```
buntu@ubuntu:~/Desktop$ rostopic echo /mavros/vision_pose/pose
eader:
  seq: 1087
  stamp:
    secs: 1620294065
    nsecs: 671762228
  frame_id: "camera"
ose:
  position:
    x: 0.0883472226991
    y: 0.161177385071
    z: 0.745088768212
  orientation:
    x: 0.749413727239
    y: 0.659339566315
    z: -0.0602677551196
    w: 0.00426607676861
---
eader:
  seq: 1088
  stamp:
    secs: 1620294066
    nsecs:  38618803
  frame_id: "camera"
ose:
```

Fig. 5.24 vision_to_pose data

5.15 Ground test and Flight test:

Now, first connect the Li-Po battery to the system and then run the 3 nodes terminals looks like:



```
Terminal - ubuntu@ubuntu: ~/Desktop
Edit  View  Terminal  Tabs  Help
u@ubuntu:~/Desktop$ rosrun vision_to_mavros set_origin.py
 message SET_GPS_GLOBAL_ORIGIN {target_system : 1, latitude : 42
le : 126432900, altitude : 163000}
 message SET_HOME_POSITION {target_system : 1, latitude : 425633
 126432900, altitude : 163000, x : 0, y : 0, z : 0, q : [1, 0, 0
 : 0, approach_y : 0, approach_z : 1}
 message SET_GPS_GLOBAL_ORIGIN {target_system : 1, latitude : 42
le : 126432900, altitude : 163000}
 message SET_HOME_POSITION {target_system : 1, latitude : 425633
 126432900, altitude : 163000, x : 0, y : 0, z : 0, q : [1, 0, 0
 : 0, approach_y : 0, approach_z : 1}
u@ubuntu:~/Desktop$
```

```
Terminal - /home/ubuntu/catkin_workspace/src/vision_to_mavros/launch/tf_to_n
File  Edit  View  Terminal  Tabs  Help

PARAMETERS
 * /apriltags_to_mavros/output_rate: 30
 * /apriltags_to_mavros/source_frame_id: /Tag0
 * /apriltags_to_mavros/target_frame_id: /camera
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES
  /
    apriltags_to_mavros (vision_to_mavros/vision_to_mavros_node)

ROS_MASTER_URI=http://localhost:11311

process[apriltags_to_mavros-1]: started with pid [2490]
[ WARN] [1620293822.342758934]: Using default target_frame_id: /camera
[ WARN] [1620293822.352410026]: Using default source_frame_id: /Tag0
[ WARN] [1620293822.358150638]: Using default output_rate: 20.000000
[ WARN] [1620293822.365427545]: Using default gamma_world: -1.570796
[ WARN] [1620293822.368443119]: Using default roll_cam: 0.000000
[ WARN] [1620293822.374725934]: Using default pitch_cam: 0.000000
[ WARN] [1620293822.378307026]: Using default yaw_cam: 1.570796
[ INFO] [1620293822.382047749]: Precision landing disabled by default
```

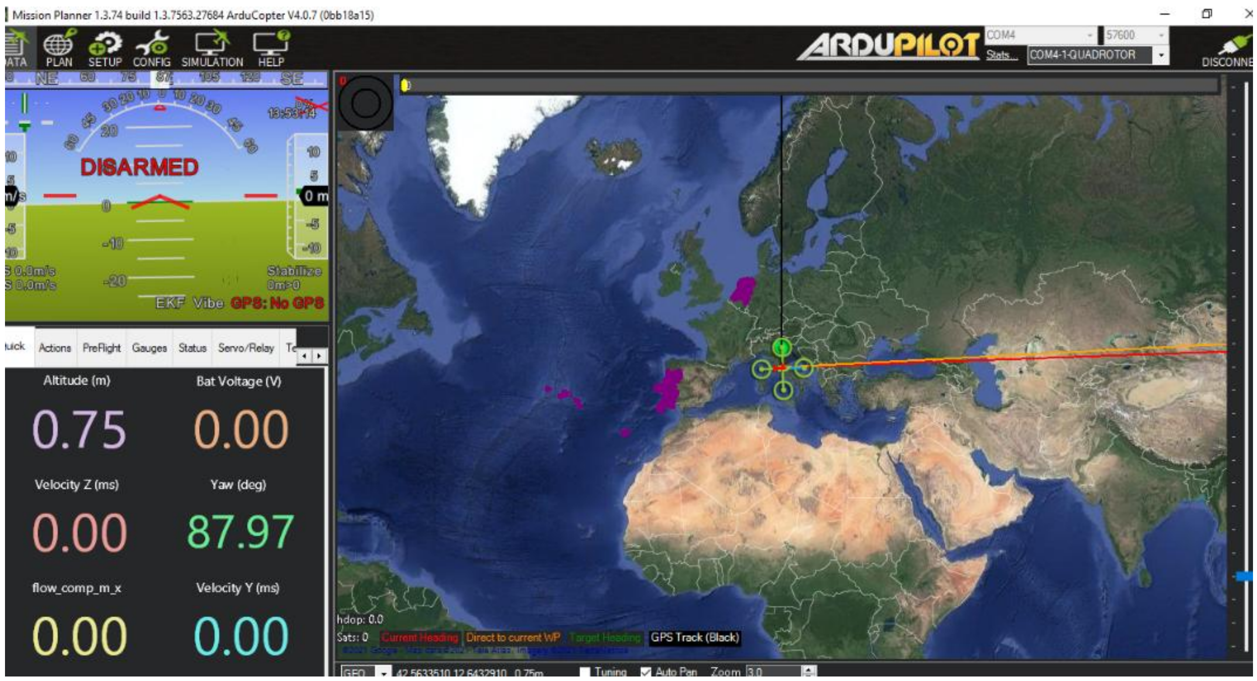
```
[ERROR] [1620293922.572475319]: FCU: PreArm: Throttle below Fai
[ERROR] [1620293922.573000745]: FCU: PreArm: Hardware safety sw
[ WARN] [1620293925.125525373]: TM : RTT too high for timesync:
[ INFO] [1620293925.331346558]: FCU: Mission Planner 1.3.74
[ INFO] [1620293925.335282132]: FCU: Mission Planner 1.3.74
[ INFO] [1620293925.343754410]: FCU: Mission Planner 1.3.74
[ INFO] [1620293927.598384056]: HP: requesting home position
[ INFO] [1620293937.598193884]: HP: requesting home position
[ INFO] [1620293947.599337933]: HP: requesting home position
[ WARN] [1620293948.390809470]: TM : RTT too high for timesync:
[ERROR] [1620293952.569659764]: FCU: PreArm: Throttle below Fai
[ERROR] [1620293952.570067208]: FCU: PreArm: Hardware safety sw
[ INFO] [1620293957.602902261]: HP: requesting home position
[ INFO] [1620293967.598570014]: HP: requesting home position
[ INFO] [1620293977.598162823]: HP: requesting home position
[ERROR] [1620293982.568618968]: FCU: PreArm: Throttle below Fai
[ERROR] [1620293982.568968227]: FCU: PreArm: Hardware safety sw
[ERROR] [1620294012.419001339]: FCU: PreArm: Throttle below Fai
[ERROR] [1620294012.420269969]: FCU: PreArm: Hardware safety sw
[ WARN] [1620294013.283040746]: TM : RTT too high for timesync:
```

```
error, index: 768, error: No data available, number: 61
 06/05 09:36:47,133 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: No data available, number: 61
 06/05 09:36:47,185 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: No data available, number: 61
 06/05 09:36:47,236 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: No data available, number: 61
 06/05 09:36:47,398 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: No data available, number: 61
 06/05 09:37:51,987 ERROR [547172102528] (ds5-options.cpp:86) Asic Temperature v
alue is not valid!
 06/05 09:37:52,989 ERROR [547172102528] (ds5-options.cpp:86) Asic Temperature v
alue is not valid!
 06/05 09:37:53,990 ERROR [547172102528] (ds5-options.cpp:86) Asic Temperature v
alue is not valid!
 06/05 09:38:15,537 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: Resource temporarily unavailable, numbe
r: 11
 06/05 09:38:15,588 WARNING [547834368384] (messenger-libusb.cpp:42) control_tra
nsfer returned error, index: 768, error: Resource temporarily unavailable, numbe
r: 11
```

Fig. 5.25 All ROS nodes are running at the same time

Next, show the UAV icon on MP software by enabling the EKF i.e. set EKF home. For this, send the SET_GPS_GLOBAL_ORIGIN and SET_HOME_POSITION MAVLink messages.

Run the script "vision_to_mavros set_origin.py on another terminal. Now, the UAV icon appear on the map as shown in fig. 5.23



Altitude (m) 0.75
Bat Voltage (V) 0.00
Velocity Z (ms) 0.00
Yaw (deg) 87.97
flow_comp_m_x 0.00
Velocity Y (ms) 0.00

Fig. 26 UAV location on MP

After that, ready for the takeoff. Switch on the RC transmitter and go to “Action” menu on Mission Planner. Click on “Arm/Disarm” button to ARM the UAV.

Always do the takeoff in stabilize mode and check the UAV is stable or not.

Chapter 6
Experimental Result

The test cases and experiments that tried in the laboratory are explained in this chapter. This chapter presents the issues that occurred while flying the UAV, some extra configurations and do the changes to get the position using marker and evaluation of results.

6.1 UAV Frame change from Quadcopter to Hexacopter

As explained in section 4.1, Hybro S500 UAV Frame has been used when starting this project. But the Electronic Speed Controller (ESC) of one rotor was become faulty and it cannot control the motor. Because of one faulty ESC, UAV cannot do the further operation. Though, by ordering the new ESC could not solve the issue. For this purpose, we moved to other UAV i.e. Hexacopter as shown below.

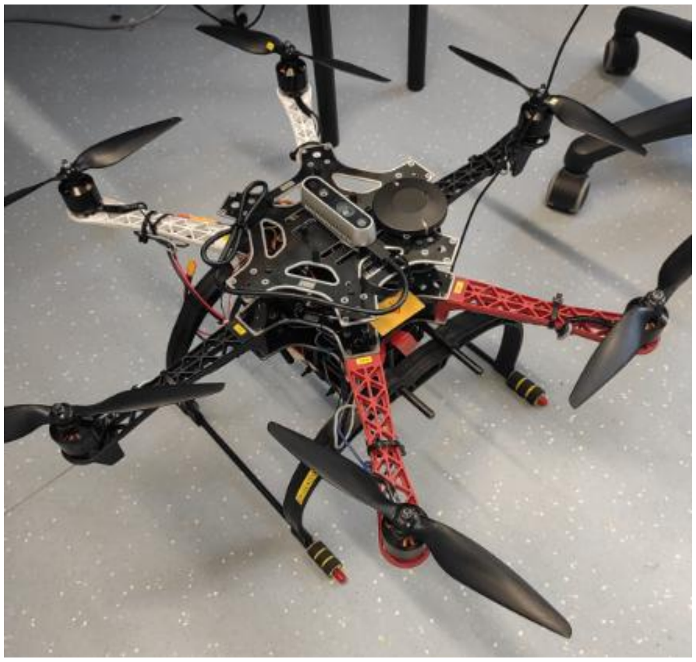


Fig. 6.1 Hexacopter

There are some changes have done to installing Raspberry Pi case and Real-Sense Camera. From image we can see that Real-Sense camera has been mounted on top part of the UAV. This change can give easy solution to detect the AprilTag marker which is explained in section 6.3.

6.2 Replace the Li-Po Battery

As the UAV frame is change from Quadcopter to Hexacopter, the rotors are increased. So, the system need more power supply to run the rotors, Raspberry Pi, StromPi battery. The 4500 mAh

Li-Po battery (see section 4.3) is not sufficient for Hexacopter. We have installed new battery i.e. 6600 mAh power supply. (Refer Fig. 6. 2)

Fig. 6.2 Li-Po 6600 mAh Battery

6.3 Marker Position

The Real-Sense camera is pointing upward and marker is on ceiling. The orientation has been change as the marker position is change from floor to ceiling. The position of Z-axis is change as marker pointing towards the camera as shown in Fig. 6.3.

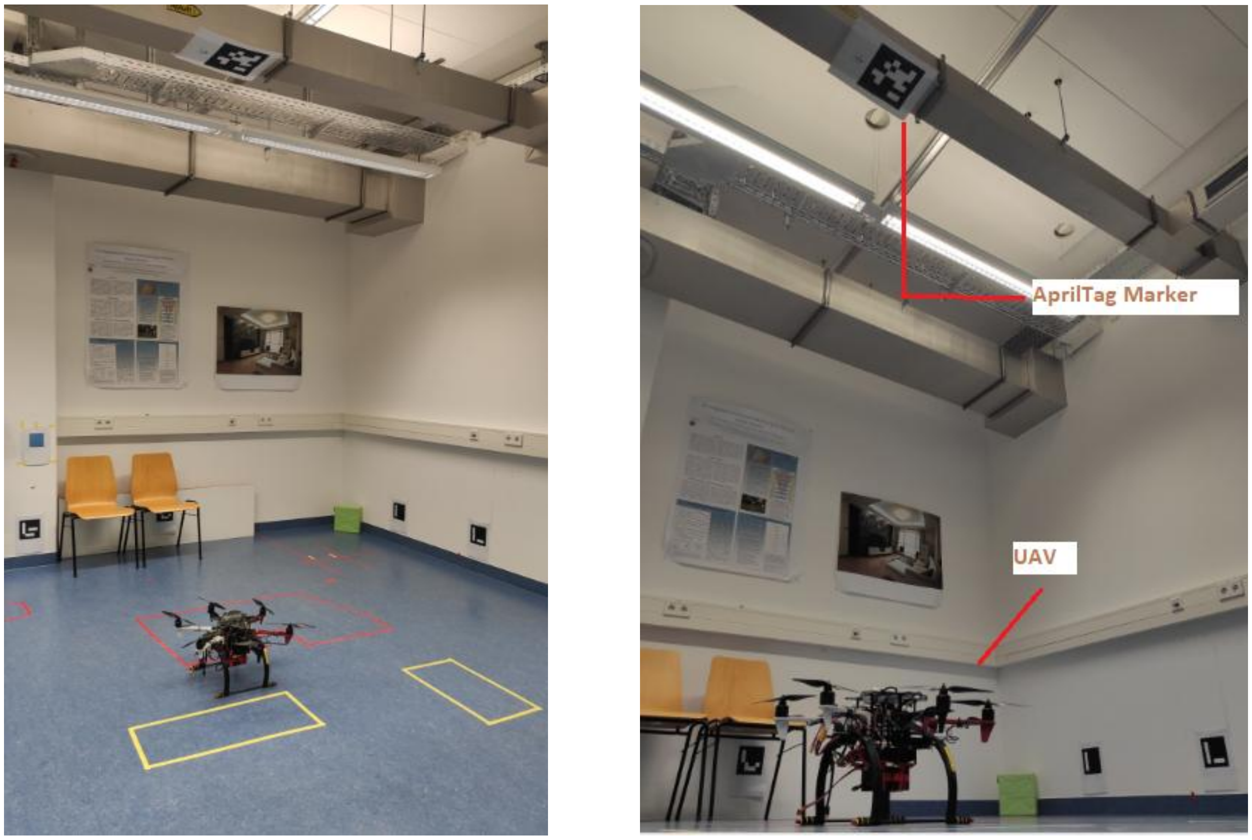


Fig. 6.3 UAV and Marker Position

6.4 Object Detection of an UAV Using Raspberry Pi Camera

Keep the UAV in front of the object that has to be detect (see Fig. 6.4). Start the UAV as mentioned in section 5.14 and 5.15. If the UAV can fly stable then it can detect object easily. In Fig. 6.5 (A) and Fig. 6.5 (B) can see that object (i.e. red square) can detect through Raspberry Pi Camera.

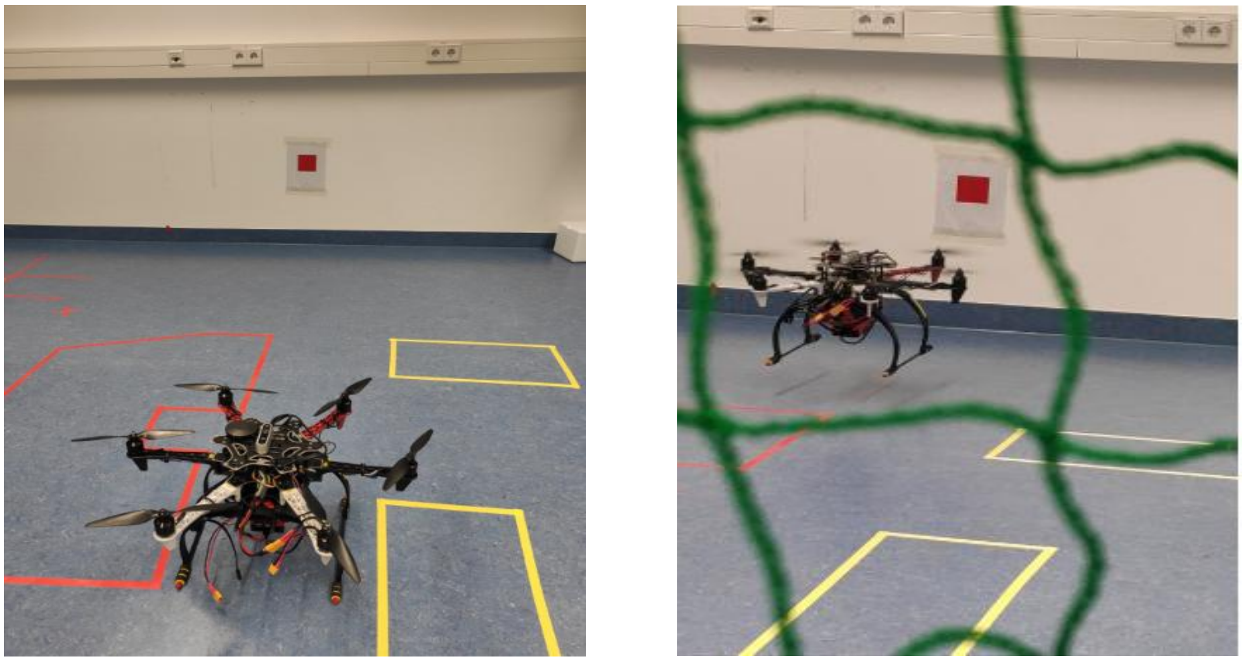


Fig. 6.4 Keep UAV Infront of Object

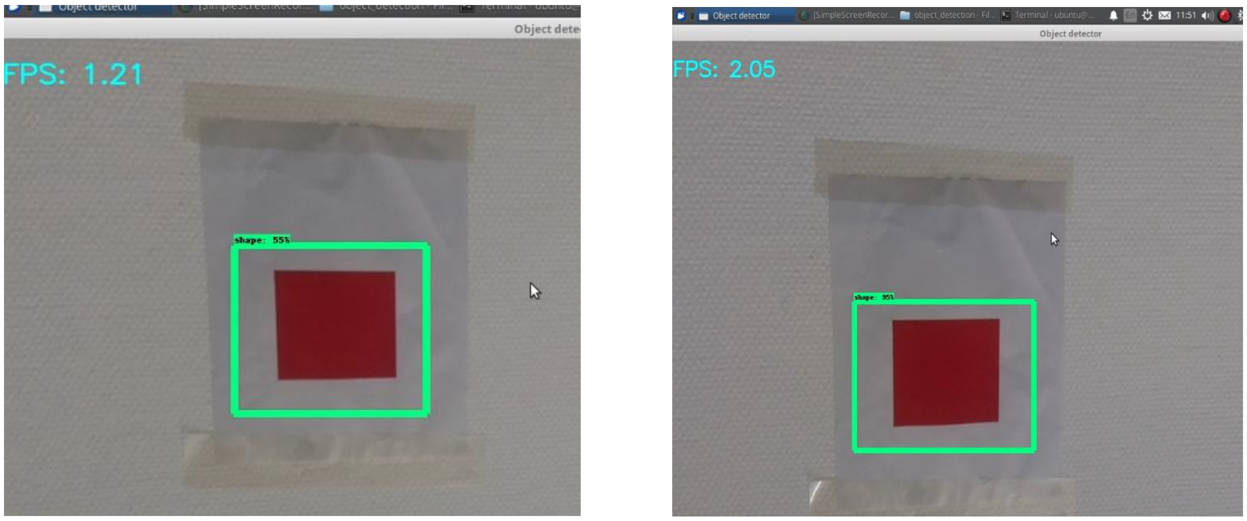


Fig. 6.5 Detected Object

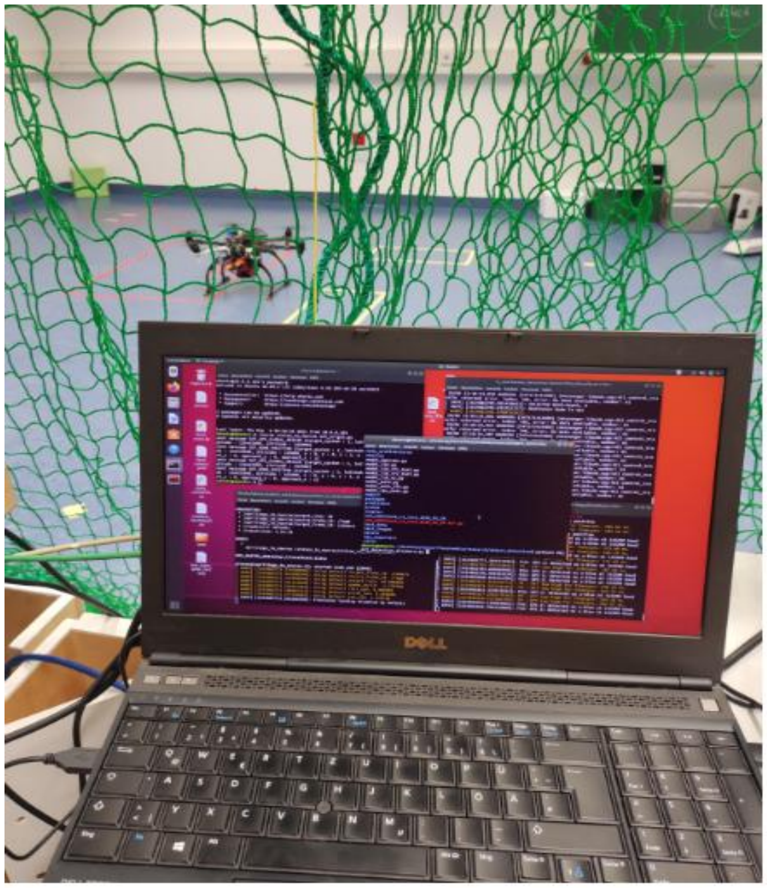


Fig. 6.6 Control UAV from Computer

Fig. 6.6 shows that all ROS nodes (MAVROS, Real-Sense, and AprilTag) are running on computer and we can control the UAV and Raspberry Pi using ground control stations.

Chapter 7
Conclusion and Future Scope

The navigation of an UAV in GPS-denied indoor environment is a challenging issue in warehouses. Despite the fact that there are different approaches to completing the task of investigating an unknown location, the idea of this thesis was to enable an appropriate localization method of an UAV using computer vision approach. The thesis was able to develop the system that can navigate and get position from AprilTag fiducial marker. Because GPS signals aren't used in pose estimation or flight command generation, the UAV can land even if the signal isn't there.

As future work, the UAV can navigate fully autonomously in non-GPS indoor environment. In the further work, the flight could be more stable so it can hold the position of UAV for some period and detect the objects easily.

LIST OF FIGURES

References

1) W. Kwon, J. H. Park, M. Lee, J. Her, S. -H. Kim and J. -W. Seo, "Robust Autonomous Navigation of Unmanned Aerial Vehicles (UAVs) for Warehouses’ Inventory Application," in IEEE Robotics and Automation Letters, vol. 5, no. 1, pp. 243-249, Jan. 2020, doi: 10.1109/LRA.2019.2955003.

2) Polvara, R.; Sharma, S.; Wan, J.; Manning, A.; Sutton, R. Vision-Based Autonomous Landing of a Quadrotoron the Perturbed Deck of an Unmanned Surface Vehicle.Drones2018,2, doi:10.3390/drones2020015

3) P.ˇSkrinjar, P.ˇSkorput, and M. Furdi´c, “Applications of unmanned aerial vehicles in logistic processes,”inNew Technologies, Development and Application,I. Karabegovi´c, Ed.Cham: Springer InternationalPublishing, 2019, pp. 359–366

4) E. H. Ch. Harik et. al., Towards an Autonomous Warehouse Inventory Scheme. 2016 IEEE Symposium Series on Computational Intelligence, 2016

5) Kalinov, I.; Safronov, E.; Agishev, R.; Kurenkov, M.; Tsetserukou, D. High-precision UAV localization systemfor landing on a mobile collaborative robot based on an IR marker pattern recognition.

6) A. Rohan, M. Rabah and S. -H. Kim, "Convolutional Neural Network-Based Real-Time Object Detection and Tracking for Parrot AR Drone 2," in IEEE Access, vol. 7, pp. 69575-69584, 2019, doi: 10.1109/ACCESS.2019.2919332.

7) Liu, L., Ouyang, W., Wang, X. et al. Deep Learning for Generic Object Detection: A Survey. Int J Comput Vis 128, 261–318 (2020). https://doi.org/10.1007/s11263-019-01247-4

8) https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/

9) https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/

10) https://docs.px4.io/master/en/flight_controller/pixhawk4.html

11) https://ardupilot.org/planner/docs/configuring-hardware.html

12) https://github.com/thien94/vision_to_mavros

13) https://joy-it.net/en/products/rb-strompi3

14) https://github.com/AprilRobotics/apriltag_ros