Design and development of a Drone Delivery System

Project report submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology in Electronics and Communication Engineering

by

Garvit Goyal - 21dec004

Akul Khandelwal -21uec020

Labhesh Mundhada - 21uec156

Under Guidance of
Dr. Atul Mishra and Dr. Mohit Makkar

The LNM Institute of Information Technology

Department of Electronics and Communication Engineering The LNM Institute of Information Technology, Jaipur

July 2023

The LNM Institute of Information Technology Jaipur, India

CERTIFICATE

This is to certify that the project entitled “Design and development of a drone delivery system” , submitted by Akul Khandelwal (21uec020), Labhesh Mundhada (21uec156) and Garvit Goyal (21dec004) in partial fulfillment of the requirement of degree in Bachelor of Technology (B. Tech), is a bonafide record of work carried out by them at the Department of Electronics and Communication Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2023-2024 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my/our opinion, this report is of standard required for the award of the degree of Bachelor of Technology (B. Tech).

Date 22/07/2023

Adviser: Dr. Atul Mishra

Acknowledgments

We express our heartfelt gratitude to Dr Atul Mishra Sir and Dr. Mohit Makkar Sir for his exceptional mentorship and Udayveer Singh Sir for his unwavering support throughout this project. Our deepest appreciation also extends to the entire robotics lab team, whose dedication and expertise were instrumental in overcoming challenges and achieving success. Their invaluable guidance, tireless assistance, and collaborative spirit paved the way for this project's realization. Without their selfless contributions, this project would have remained a distant dream. We would also like to thank our friend Hardik Agarwal for assisting us in our project. We are truly indebted to everyone for their indelible impact, without which this project would not have been possible

Abstract

The rapid growth of e-commerce and the increasing demand for faster, more efficient delivery methods have paved the way for the emergence of drone technology in the logistics industry. This abstract presents a comprehensive overview of a groundbreaking drone delivery project aimed at revolutionizing last-mile logistics. By leveraging the capabilities of unmanned aerial vehicles (UAVs) and cutting-edge autonomous systems, this project seeks to address the challenges associated with traditional delivery methods, such as congestion, delayed deliveries, and high operational costs.

The primary objective of this project is to develop a robust and scalable drone delivery system that can safely and efficiently transport packages over short distances. Through advanced algorithms, this project aims to optimize route planning and deliver packages efficiently.The project encompasses several key components, including drone design and development and navigation. Furthermore, we have integrated autonomous capabilities, allowing drones to navigate complex environments autonomously.

Additionally, consideration will be given to various aspects, such as package size and weight limitations, delivery hubs, and customer interface, to provide a user-friendly and efficient experience for both businesses and end consumers.

The potential impact of this drone delivery project is vast. By streamlining last-mile logistics, it has the potential to significantly reduce delivery times, lower operational costs, and minimize carbon emissions associated with traditional delivery methods. It could also improve accessibility in remote or underserved areas, providing a lifeline for communities with limited access to essential goods and services.

In conclusion, this abstract outlines a groundbreaking drone delivery project that aims to transform last-mile logistics. This project strives to create an efficient, reliable, and environmentally friendly delivery solution by leveraging advanced drone technology, autonomous systems, and a comprehensive logistical framework. The outcomes of this project have the potential to revolutionize the logistics industry and enhance the

overall customer experience, paving the way for a new era of delivery systems.

Contents

Chapter 1:- Drone Assembly

The various parts used in the drone are as follows:

1) PIXHAWK Flight Controller:

pixhawk is an independent open-hardware project providing readily-available, low-cost, and high-end, autopilot hardware designs to the academic, hobby and industrial communities.

Key benefits of using a Pixhawk series controller include:

- Software support - as PX4 reference hardware these are our best-maintained boards.
- Flexibility in terms of hardware peripherals that can be attached.
- High quality.
- Highly customizable in terms of form factor.
- Widely-used and thus well-tested/stable.
- Automated update of latest firmware via QGroundControl (end-user friendly).

There are many boards available in the market but we found pixhawk best suited for this project.

It Supports 8 RC channels with 4 serial ports. Various user interfaces are available for programing, reviewing logs, even some apps for smartphones & tablets. It detects and configures all its peripherals automatically

The benefits of the Pixhawk system include a Unix/Linux-like programming environment, completely new autopilot functions. Sophisticated scripting of missions and flight behavior, and a custom PX4 driver layer ensuring tight timing across all processes.

2) BLDC Motors:

A motor converts supplied electrical energy into mechanical energy. Various types of motors are in common use. Among these, brushless DC motors (BLDC) feature high efficiency and excellent controllability, and are widely used in many applications. The BLDC motor has power-saving advantages relative to other motor types. The motors have KV ratings which specifies the RPM that a motor will achieve per volt. Here we have used motors of 1000KV.

3) S500 Quadcopter Frame:

This S500 Quadcopter Frame is made from Glass Fiber which makes it tough and durable. They have the arms of ultra-durable Polyamide-Nylon which are the stronger molded arms having a very good thickness so no more arm breakage at the motor mounts on a hard landing. The arms have support ridges on them, which improves stability and provides faster forward flight. The S500 is strong, light, and has a sensible configuration including a PCB(Printed Circuit Board) with which we can directly solder your ESC's to the Quadcopter.

4) M8N GPS and COMPASS Module:

This is a new generation NEO-M8N High Precision GPS Module with Built-in Compass for PIXHAWK and APM FC (Ready to connect to PIXHAWK FC) with onboard Compass, low power consumption, and high precision, the ultimate accuracy is 0.6 meters, actually almost 0.9 meters, greater than the previous generation NEO-7N 1.4-1.6 meters accuracy, support GPS/QZSS L1 C/A, GLONASS L10F, BeiDou B1 protocol, and mode or more.NEO-M8N GPS Module for APM APM2.52 APM Flight Controller with Case and GPS Antenna Mount to protect the GPS preventing electromagnetic interference. A new generation GPS NEO-M8N GPS Module, with low power consumption and high precision, the ultimate accuracy is 0.6 meters, actually almost 0.9 meters, greater than the previous generation NEO-7N 1.4-1.6 meters accuracy. Support GPS/QZSS L1 C/A, GLONASS L10F, BeiDou B1 protocol, and mode or more. The NEO-M8N and ceramic antenna make this a very accurate receiver, fast locks & lots of satellites, and the stand is very sturdy to protect the GPS preventing electromagnetic interference.

5) LIPO Battery:

A lithium polymer battery, or more correctly lithium-ion polymer battery (abbreviated as LiPo, LIP, Li-poly, lithium-poly and others), is a rechargeable battery of lithium-ion technology using a polymer electrolyte instead of a liquid electrolyte. High conductivity semisolid polymers form this electrolyte. These batteries provide higher specific energy than other lithium battery types and are used in applications where weight is a critical feature, such as mobile devices, radio controlled devices and some EV.The voltage of a single LiPo cell depends on its chemistry and varies from about 4.2 V (fully charged) to about 2.7–3.0 V (fully discharged), where the nominal voltage is 3.6 or 3.7 volts (about the middle value of highest and lowest value) for cells based on lithium-metal-oxides .Unlike lithium-ion cylindrical and prismatic cells, which have a rigid metal case, LiPo cells have a flexible, foil-type case, so they are relatively unconstrained. Moderate pressure on the stack of layers that compose the cell results in increased capacity retention, because the contact between the components is maximized and deformation is prevented, which is associated with increase of cell impedance and degradation.

Chapter 2:- Integration with Raspberry Pi

The quadcopter that has been made from the above mentioned components is by far only capable of human controlled and pre planned flights. As per our project requirements we needed a drone to take the data from the website and plan the mission autonomously . This capability is not provided by pixhawk , so we integrated a companion computer called Raspberry Pi with pixhawk. The Raspberry Pi is a full fledged computer that can connect to a network and exchange data. The RPi communicated with pixhawk using serial communication namely UART.

Now after integrating we needed a common protocol for both RPi and pixhawk to communicate. So, we used the mavlink protocol.

1) Raspberry Pi 3:

The Raspberry Pi is a credit card-sized computer. The Raspberry Pi 3 Model B is the third generation Raspberry Pi. It is based on the BCM2837 system-on-chip (SoC), which includes a 1.2 GHz quad-core ARMv8 64bit processor and a powerful VideoCore IV GPU. The Raspberry Pi can run a full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, Debian, Fedora, and Arch Linux, as well as Microsoft Windows 10 IoT Core.

The Raspberry Pi 3 Model B is the first Raspberry Pi to feature onboard wireless and Bluetooth connectivity. The Raspberry Pi was designed by the Raspberry pi foundation to provide an affordable platform for experimentation and education in computer programming. The Raspberry Pi can be used for many of the things that a normal desktop PC does, including word-processing, spreadsheets, high-definition video, games, and programming. USB devices such as keyboards and mice can be connected via the board’s four USB ports.

The RPi has a 40-pin GPIO header where we can connect external sensors and peripherals.the pixhawk was connected to RPi using these GPIO pins.

2) MAVLink Protocol and DroneKit Library:

MAVLink or Micro Air Vehicle Link is a protocol for communicating with small unmanned vehicles. It is designed as a header-only message marshaling library. It is used mostly for communication between a Ground control station and unmanned vehicles , and in the intercommunication of the subsystem of the vehicle. It can be used to transmit the orientation of the vehicle, its GPS location and speed. To make use of the MAVlink protocol we have used a library named dronekit which provides python API to communicate with MAVlink messages.

The following is the code that has been put into the raspberry pi :

```python
from dronekit import connect,VehicleMode,LocationGlobalRelative
from pymavlink import mavutil
import time
import os
import cv2
import threading
import requests
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)


############## temperory function ###############

def Channel(chNum):

        if vehicle.channels[1] == None :
            try:
                if PH_Channels == None:
                    if vehicle.channels[7] == None:
                        VE.Inform("Channel are None - decorator being deployed on RC_Channels",1,"AP")


                        vehicle.on_message('RC_CHANNELS')
                        def RCIN_listener(name, message):

                            global PH_Channels

                            tempchan = []
                            tempchan.append(0)
                            tempchan.append(message.chan1_raw)
                            tempchan.append(message.chan2_raw)
                            tempchan.append(message.chan3_raw)
                            tempchan.append(message.chan4_raw)
                            tempchan.append(message.chan5_raw)
                            tempchan.append(message.chan6_raw)
                            tempchan.append(message.chan7_raw)
                            tempchan.append(message.chan8_raw)
                            tempchan.append(message.chan9_raw)
                            tempchan.append(message.chan10_raw)
                            tempchan.append(message.chan11_raw)
                            tempchan.append(message.chan12_raw)
                            tempchan.append(message.chan13_raw)
                            tempchan.append(message.chan14_raw)

                            PH_Channels = tempchan

                    channel = None
```

```python
        for id in ids:
            idlist.append(int(id[0]))
        if (True):
            aruco.drawDetectedMarkers(img, bboxs)

        for i in range(0, len((bboxs))):
            if(ids[i][0]==7):
                ans[idlist[i]] = []
                corners[idlist[i]] = None
                list_temp = []
                x1 = int((((bboxs[i])[0])[0])[0])
                y1 = int((((bboxs[i])[0])[0])[1])
                x2 = int((((bboxs[i])[0])[1])[0])
                y2 = int((((bboxs[i])[0])[1])[1])

                for it1, it2 in (bboxs[i])[0]:
                    extemp = []
                    extemp.append(it1)
                    extemp.append(it2)
                    list_temp.append(extemp)

                corners[idlist[i]] = list_temp

                centre = find_centres(img, ((bboxs[i])[0]))
                ans[idlist[i]].append(centre)
                centres.append(centre)
                angle = None
                if (x2 == x1):
                    if (y2 > y1):
                        angle = 90
                    else:
                        angle = -90
                elif (y1 == y2):
                    if (x1 < x2):
                        angle = 0
                    else:
                        angle = -180

                elif (x1 < x2 and y2 < y1):
                    temp = ((y1-y2)/(x2-x1))
                    angle = (1 * int(math.degrees(math.atan(temp))))

                elif (x1 < x2 and y1 < y2):
                    temp = ((y2-y1)/(x2-x1))
                    angle = -1*(int(math.degrees(math.atan(temp))))

```

```python
                else:
                    channel = PH_Channels[chNum]
            except:
                channel = None
        else:
            channel = vehicle.channels[int(chNum)]
        return channel


###############################################

def buffer_clear():
    global frame
    print("buffer clear started")
    cap=cv2.VideoCapture(0)

    while(True):
        ret,fr=cap.read()
        if(ret):
            frame=fr
            cv2.imshow('frame',frame)
            cv2.waitKey(1)


def find_centres(image, arr):
    M = cv2.moments(arr)
    orx = int(M["m10"]/M["m00"])
    ory = int(M["m01"]/M["m00"])
    return [orx, ory]

def get_aruco_centers(image, centres):
    from cv2 import aruco
    import math
    global angle
    img = image
    ans = {}
    corners = {}
    idlist = []
    imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    key = getattr(aruco, f'DICT_{5}X{5}_{250}')
    arucoDict = aruco.Dictionary_get(key)
    arucoParam = aruco.DetectorParameters_create()
    bboxs, ids, _ = aruco.detectMarkers(
        imgray, arucoDict, parameters=arucoParam)
    if ids is None:
        pass
    else:
```

```
                elif (x1 > x2 and y1 < y2):
                    temp = ((y2-y1)/(x1-x2))
                    angle = -(int(180-math.degrees(math.atan(temp))))

                elif (x1 > x2 and y1 > y2):
                    temp = ((y1-y2)/(x1-x2))
                    angle = -(-1 * int(180-(math.degrees(math.atan(temp)))))
    return centres


def landing_align():
    global align_on
    global frame
    global vehicle
    base_roll=1502
    base_pitch=1502
    while(True):
        centres=[]
        centres=get_aruco_centers(frame,centres)

        if(len(centres)>0):
            error_roll=centres[0][0]-320
            error_pitch=centres[0][1]-240

            decr_roll=error_roll*0.01
            if(decr_roll>11 or decr_roll<-11):
                if(decr_roll>11):
                    decr_roll=11
                elif(decr_roll<-11):
                    decr_roll=-11
            new_roll=base_roll+decr_roll

            decr_pitch=error_pitch*0.01
            if(decr_pitch>11 or decr_pitch<-11):
                if(decr_pitch>11):
                    decr_pitch=11
                elif(decr_pitch<-11):
                    decr_pitch=-11
            new_pitch=base_pitch+decr_pitch
```

```python
                vehicle.channels.overrides['1']=int(new_roll)
                vehicle.channels.overrides['2']=int(new_pitch)
            else:
                vehicle.channels.overrides['1']=int(1502)
                vehicle.channels.overrides['2']=int(1502)

            if(align_on==False):
                return


def main_func():

    global frame
    global vehicle
    global align_on
    baseAPI="https://automated-drone-delivery-vha4.vercel.app/api/location/"


    ############################# order_receive_from_website #######################

    while(True):

        r=requests.get("https://automated-drone-delivery-vha4.vercel.app/api/location")
        response_dict=r.json()
        customer_id=None
        loc1=None
        loc2=None
        if(response_dict['message']['success']==True):
            if(len(response_dict['response'])!=0):
                customer_id=response_dict['response']['_id']
                loc1=response_dict['response']['start']
                loc2=response_dict['response']['end']
                print('order received for ' , customer_id)
                print("going from",loc1," to ",loc2)
                break
            else:
                print("no new order")
        time.sleep(3)
    '''
    '''
    vehicle.mode="GUIDED"
    vehicle.armed=True
    while not vehicle.armed:
        print('arming...')
        vehicle.armed=True
        time.sleep(1)
    print("....armed")
```

```
    vehicle.simple_takeoff(3)
    while True:
        if (vehicle.location.global_relative_frame.alt)>=3*0.95:
            break
        time.sleep(1)
    vehicle.mode="GUIDED"
    campuslocations = {"BH2" :  (26.930051,75.920684, 5), "canteen" : (26.930032, 75.920509, 5)}
    (x , y, h) = campuslocations[loc1]
    target1=LocationGlobalRelative(x,y,h)
    vehicle.simple_goto(target1,groundspeed=3)
    time.sleep(15)
    curstate=GPIO.input(1)
    vechicle.mode="LAND"
    time.sleep(10)
    while(vehicle.armed==True):
        time.sleep(2)
    while(GPIO.input(1)==curstate):
        time.sleep(1)
    time.sleep(10)
    ################# landed at pickup loc #########################
    vehicle.mode="GUIDED"
    if(vehicle.armed==False):
        vehicle.armed=True
        while not vehicle.armed:
            print('arming...')
            time.sleep(1)
        print("....armed.....again")
        print(vehicle.mode.name)
        vehicle.simple_takeoff(3)
        while True:
            if (vehicle.location.global_relative_frame.alt)>=3*0.95:
                break
            time.sleep(1)
    (x,y,z)=campuslocations[loc2]
    target2=LocationGlobalRelative(x,y, 5)
    vehicle.simple_goto(target2,groundspeed=3)
    time.sleep(10)
    print("looking for ARUCO")
    while(True):
            centres=[]
            centres=get_aruco_centers(frame,centres)
            if(len(centres)!=0):
                print("LANDING sequence initiated")
                align_on=True
                #align.start()
                time.sleep(8)
                vehicle.mode="LAND"
                print("Landing guidance INTERNAL")
```

```python
                    time.sleep(2)
                align_on=False
                time.sleep(10)
                break
            t2=time.time()
            delta=t2-t1
            if(delta>10):
                break
    curstate=GPIO.input(1)
    vehicle.mode="LAND"
    time.sleep(10)
    while(vehicle.armed==True):
        time.sleep(2)
    while(GPIO.input(1)==curstate):
        time.sleep(1)
    time.sleep(15)
    if(vehicle.armed==False):
        vehicle.armed=True
        while not vehicle.armed:
            print('arming...')

            time.sleep(1)
        print("....armed.....again")
        print(vehicle.mode.name)
        vehicle.simple_takeoff(3)
        while True:
            if (vehicle.location.global_relative_frame.alt)>=3*0.95:
                break
            time.sleep(1)
    target3=LocationGlobalRelative(26.9306055,75.9205639, 5)
    vehicle.simple_goto(target2,groundspeed=3)
    time.sleep(15)
    vehicle.mode="LAND"
    time.sleep(10)
    newAPI=baseAPI+customer_id
    print("Delivered")
    payload={"deliveryStatus":"Delivered"}
    requests.put(newAPI,data=payload)
if __name__=='__main__':
    frame=None
    vehicle=None
    align_on=False
    GPIO.setup(1,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    buff=threading.Thread(target=buffer_clear)
    buff.start()
    time.sleep(2)
    print("UPDATE : Thread started for camera buffer clear.")
    align=threading.Thread(target=landing_align)
```

```
align=threading.Thread(target=landing_align)
vehicle=connect('/dev/ttyAMA0',wait_ready=True, baud=921600)
print("connected...")
time.sleep(1)
while(True):
    try:
        main_func()
    except:
        vehicle.mode="LAND"
    time.sleep(10)
    print("landed")
```

Below is the description of the code:

```
GPIO.setup(1,GPIO.IN,pull_up_down=GPIO.PUD_UP)
buff=threading.Thread(target=buffer_clear)
buff.start()
time.sleep(2)
print("UPDATE : Thread started for camera buffer clear.")
align=threading.Thread(target=landing_align)
vehicle=connect('/dev/ttyAMA0',wait_ready=True, baud=921600)
print("connected...")
time.sleep(1)
while(True):
    try:
        main_func()
    except:
        vehicle.mode="LAND"
    time.sleep(10)
    print("landed")
```

This script sets the GPIO pin as INPUT and starts threads to capture camera frames and start communication with PIXHAWK via serial port.

```python
def main_func():

    global frame
    global vehicle
    global align_on
    baseAPI="https://automated-drone-delivery-vha4.vercel.app/api/location/"


    ############################# order_receive_from_website #######################

    while(True):

        r=requests.get("https://automated-drone-delivery-vha4.vercel.app/api/location")
        response_dict=r.json()
        customer_id=None
        loc1=None
        loc2=None
        if(response_dict['message']['success']==True):
            if(len(response_dict['response'])!=0):
                customer_id=response_dict['response']['_id']
                loc1=response_dict['response']['start']
                loc2=response_dict['response']['end']
                print('order received for ' , customer_id)
                print("going from",loc1," to ",loc2)
                break
            else:
                print("no new order")
        time.sleep(3)
```

The baseAPI variable contains the PI key for fetching and updating the data on the mongoDB database. The RPi makes fetch requests to the database to get the data of any pending delivery.

The response from the API contains the username , userID , Pickup location, Delivery location, status of the order whether pending or delivered.

```
vehicle.mode="GUIDED"
vehicle.armed=True
while not vehicle.armed:
    print('arming...')
    vehicle.armed=True
    time.sleep(1)
print("....armed")
print(vehicle.mode.name)

vehicle.simple_takeoff(3)
while True:
    if (vehicle.location.global_relative_frame.alt)>=3*0.95:
        break
    time.sleep(1)
vehicle.mode="GUIDED"
campuslocations = {"BH2" :  (26.930051,75.920684, 5), "canteen" : (26.930032, 75.920509, 5)}
(x , y, h) = campuslocations[loc1]

target1=LocationGlobalRelative(x,y,h)
vehicle.simple_goto(target1,groundspeed=3)
time.sleep(15)
```

This script commands the drone to takeoff to a specified height and go to the pickup location specified by the user.

```
curstate=GPIO.input(1)
vechicle.mode="LAND"
time.sleep(10)
while(vehicle.armed==True):
    time.sleep(2)
while(GPIO.input(1)==curstate):
    time.sleep(1)
time.sleep(10)
```

After reaching the pickup location the drone lands and waits for the input from the user using an external switch to be pressed before taking off again and going to the delivery location.

```python
vehicle.mode="GUIDED"
if(vehicle.armed==False):
    vehicle.armed=True
    while not vehicle.armed:
        print('arming...')

        time.sleep(1)
    print("....armed.....again")
    print(vehicle.mode.name)
    vehicle.simple_takeoff(3)
    while True:
        if (vehicle.location.global_relative_frame.alt)>=3*0.95:
            break
        time.sleep(1)

(x,y,z)=campuslocations[loc2]
target2=LocationGlobalRelative(x,y, 5)
vehicle.simple_goto(target2,groundspeed=3)
time.sleep(10)
```

After getting the input from the user the drone takes off and goto the delivery location.

```python
print("looking for ARUCO")

while(True):
        centres=[]

        centres=get_aruco_centers(frame,centres)

        if(len(centres)!=0):
            print("LANDING sequence initiated")
            align_on=True
            #align.start()
            time.sleep(8)
            vehicle.mode="LAND"
            print("Landing guidance INTERNAL")
            while(vehicle.armed==True):
                time.sleep(2)
            align_on=False
            time.sleep(10)
            break

        t2=time.time()
        delta=t2-t1
        if(delta>10):
            break

curstate=GPIO.input(1)
vehicle.mode="LAND"
time.sleep(10)
while(vehicle.armed==True):
    time.sleep(2)
while(GPIO.input(1)==curstate):
    time.sleep(1)
time.sleep(15)
```

After reaching the delivery location the drone looks for the aruco center and try to align itself with respect to aruco.

```python
if(vehicle.armed==False):
    vehicle.armed=True
    while not vehicle.armed:
        print('arming...')

        time.sleep(1)
    print("....armed.....again")
    print(vehicle.mode.name)
    vehicle.simple_takeoff(3)
    while True:
        if (vehicle.location.global_relative_frame.alt)>=3*0.95:
            break
        time.sleep(1)
target3=LocationGlobalRelative(26.9306055,75.9205639, 5)
vehicle.simple_goto(target2,groundspeed=3)
time.sleep(15)
vehicle.mode="LAND"
time.sleep(10)
```

The drone takes off and goes back to the starting point and lands there. The delivery task has now been completed.

```python
newAPI=baseAPI+customer_id
print("Delivered")
payload={"deliveryStatus":"Delivered"}
requests.put(newAPI,data=payload)
```

On completing the delivery the drone makes an API request to update the status of the order to Delivered.

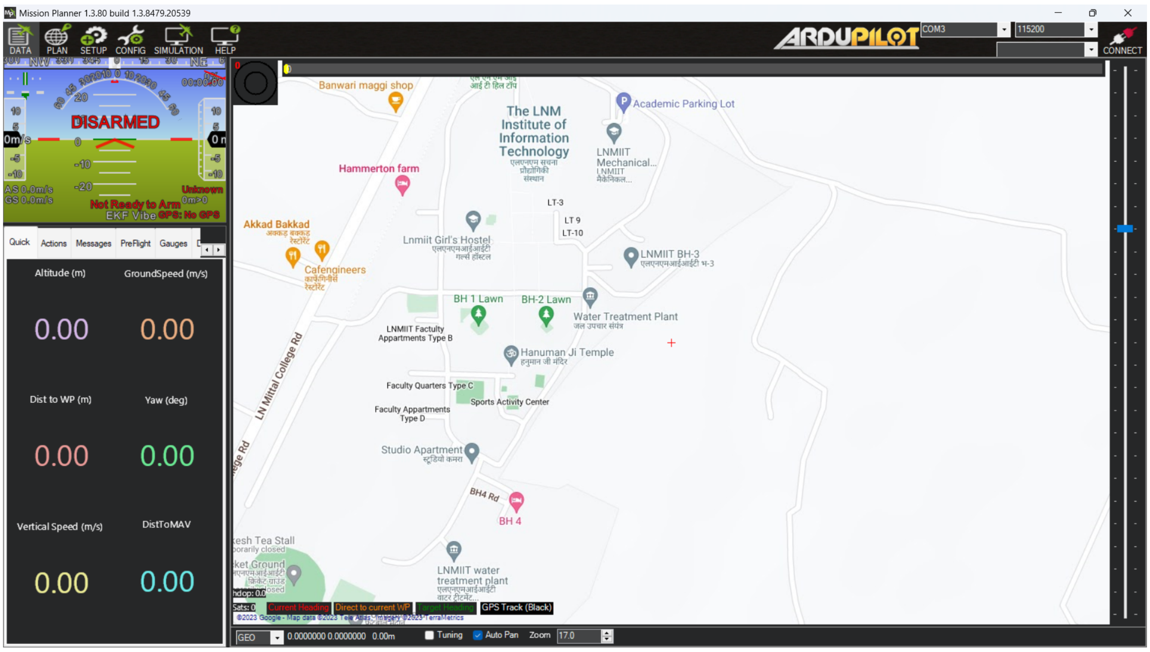
Chapter 3:- Simulation via Gazebo and Ardupilot

Simulation with Gazebo on drones is a vital tool for UAV and robotics research. Gazebo's open-source 3D simulator provides realistic environments, enabling drone modeling and testing under various conditions. Its versatility allows for complex virtual landscapes and the integration of sensors, such as cameras and LiDAR, to fine-tune algorithms. Cost-effectiveness is achieved by reducing physical drone testing, while safety is improved during development. Gazebo's wide adoption in the robotics community fosters collaboration and knowledge-sharing, driving advancements in drone technology. In conclusion, Gazebo's role in enhancing drone capabilities and innovation is crucial, making it an indispensable asset in the field.

We installed the Gazebo environment on Ubuntu and simulated our drone in the environment. When we run our dronekit code, we can see the simulation in the Gazebo of the code we have written in our script.

We tried Gazebo on Ubuntu 16.0 with the Ardupilot plugin, but we were unsuccessful in doing that with Ardupilot which is essential for testing our codes. So, in this project, we can use Gazebo but not with Ardupilot.

Using Gazebo or any other simulation tool helped us in visualizing our code without testing it on a physical drone. This is useful, especially in large projects where the risk is even greater for mishappenings or injury.



Chapter 4:- Additional Features

In addition to integrating the drone with RaspberryPi, we enhanced its capabilities by incorporating a camera, which provided real-time visuals for surveillance and navigation purposes. Furthermore, we added a payload box to expand its capacity for efficient transportation of goods.

A. Aruco Marker Detection

Aruco markers are specially designed square or rectangular patterns with unique binary codes. These markers are commonly used in computer vision applications for tasks like camera calibration, object tracking, and augmented reality. Their distinct patterns enable accurate and efficient detection, allowing systems to identify and track their positions and orientations in a real-world environment. Aruco markers have widespread applications in robotics, drone navigation, and interactive experiences where precise localization and tracking are essential.

OpenCV, a popular computer vision library, is employed to detect Aruco markers by analyzing video frames. It utilizes marker-specific algorithms to identify and extract the markers' positions, orientations, and unique IDs, enabling various applications such as augmented reality and object tracking.

Let us understand its detection coded function by function:

```
def buffer_clear():
    global frame
    print("buffer clear started")
    cap=cv2.VideoCapture(0)

    while(True):
        ret,fr=cap.read()
        if(ret):
            frame=fr
            cv2.imshow('frame',frame)
            cv2.waitKey(1)
```

We define a function that captures frames from rpi camera.

```
def find_centres(image, arr):
    M = cv2.moments(arr)
    orx = int(M["m10"]/M["m00"])
    ory = int(M["m01"]/M["m00"])
    return [orx, ory]
```

We define a function that returns the coordinates of the center of the frame.

```python
def get_aruco_centers(image, centres):
    from cv2 import aruco
    import math
    global angle
    img = image
    ans = {}
    corners = {}
    idlist = []
    imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    key = getattr(aruco, f'DICT_{5}X{5}_{250}')
    arucoDict = aruco.Dictionary_get(key)
    arucoParam = aruco.DetectorParameters_create()
    bboxs, ids, _ = aruco.detectMarkers(
        imgray, arucoDict, parameters=arucoParam)
    if ids is None:
        pass
    else:
        for id in ids:
            idlist.append(int(id[0]))
        if (True):
            aruco.drawDetectedMarkers(img, bboxs)
        for i in range(0, len((bboxs))):
            if(ids[i][0]==7):
                ans[idlist[i]] = []
                corners[idlist[i]] = None
                list_temp = []
                x1 = int((((bboxs[i])[0])[0])[0])
                y1 = int((((bboxs[i])[0])[0])[1])
                x2 = int((((bboxs[i])[0])[1])[0])
                y2 = int((((bboxs[i])[0])[1])[1])
                for it1, it2 in (bboxs[i])[0]:
                    extemp = []
                    extemp.append(it1)
                    extemp.append(it2)
                    list_temp.append(extemp)
                corners[idlist[i]] = list_temp
                centre = find_centres(img, ((bboxs[i])[0]))
                ans[idlist[i]].append(centre)
```

Upon detecting the Aruco marker, the code determines its center coordinates. Subsequently, we utilize this information to manipulate the drone's movement by overriding the "roll" and "pitch" channels. This enables precise control and maneuvering during the landing process.

```python
def landing_align():
    global align_on
    global frame
    global vehicle
    base_roll=1502
    base_pitch=1502
    while(True):
        centres=[]
        centres=get_aruco_centers(frame,centres)

        if(len(centres)>0):
            error_roll=centres[0][0]-320
            error_pitch=centres[0][1]-240

            decr_roll=error_roll*0.01
            if(decr_roll>11 or decr_roll<-11):
                if(decr_roll>11):
                    decr_roll=11
                elif(decr_roll<-11):
                    decr_roll=-11
            new_roll=base_roll+decr_roll

            decr_pitch=error_pitch*0.01
            if(decr_pitch>11 or decr_pitch<-11):
                if(decr_pitch>11):
                    decr_pitch=11
                elif(decr_pitch<-11):
                    decr_pitch=-11
            new_pitch=base_pitch+decr_pitch
            vehicle.channels.overrides['1']=int(new_roll)
            vehicle.channels.overrides['2']=int(new_pitch)
        else:
            vehicle.channels.overrides['1']=int(1502)
            vehicle.channels.overrides['2']=int(1502)

        if(align_on==False):
            return
```

The implemented code governs the drone's landing procedure, wherein it employs Aruco marker detection. When an Aruco marker is detected, the code calculates its center and verifies the non-empty status of the "center" list. Upon successful detection, the code adjusts the "roll" or "pitch" channel with an empirically derived factor obtained from manual flight experiments. Additionally, the code establishes predefined limits for the respective channels to ensure safe and controlled drone behavior during landing operations.

B. Payload Box

Due to hardware constraints, we opted to utilize a cardboard box as a payload box in our drone delivery system instead of employing servo motors. The cardboard box provides a practical and lightweight alternative for lifting payloads, with the capacity to support weights of up to 500 grams

To implement this solution, we carefully selected a sturdy and appropriately sized cardboard box that could securely hold the desired payload. The box was designed to fit within the drone's payload area while maintaining balance and stability during flight.

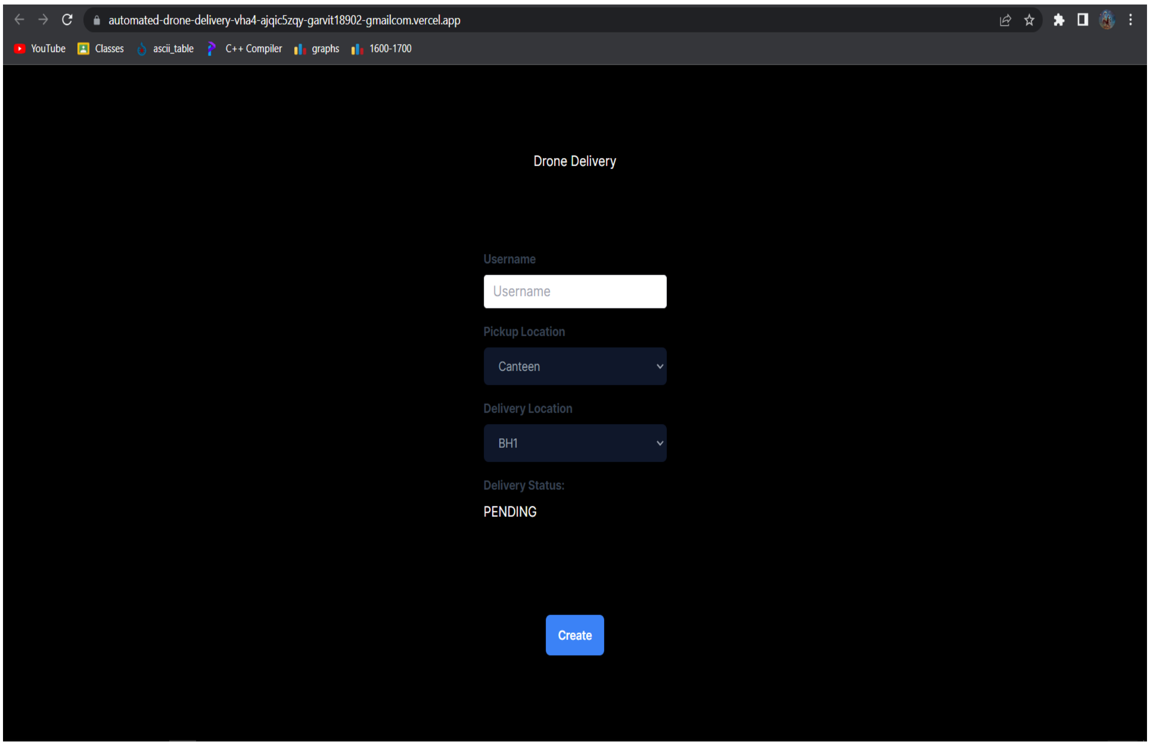
In our testing phase, we specifically examined the box's capability by placing a packet of biscuits weighing approximately 300 grams inside. This allowed us to verify that the cardboard box could successfully handle the intended payload weight without compromising the drone's stability or flight performance.

Using a cardboard box as the payload container offers several advantages. First, it is a cost-effective solution compared to servo motors, which may require additional hardware and power sources. Additionally, the cardboard box is lightweight, minimizing the impact on the drone's maneuverability and battery life.

However, it is important to note that using a cardboard box as a payload box may have limitations in terms of durability and weather resistance. Depending on the specific application and environmental conditions, additional measures such as waterproofing or reinforcing the box may be necessary.

Overall, by employing a cardboard box as a payload box, we were able to overcome hardware constraints and successfully transport a payload of up to 600 grams, as demonstrated through our testing with a packet of biscuits. This solution provides a practical and economical alternative for the drone delivery system, enabling efficient transportation of goods while maintaining flight stability and performance.

C. Website Integration



To interact with the user and to receive further instructions , we have developed a full stack

application using NextJS to create API and UI to feed pickup and Delivery location for the drone.

We have used the MongoDB database for storing geolocation and reading it back in python using raspberry pi.

Our whole area of operation is mapped into different locations and for each location we have taken its GPS coordinates. The user just has to select the location ad=nd the RPi will automatically retrieve its GPS coordinates from the pre mapped geolocations.

Conclusions and Future Work

In conclusion, our project focused on the development of a drone delivery system, encompassing various stages from building a drone to integrating it with Raspberry Pi and incorporating additional features such as a camera and payload box. Throughout this journey, we aimed to explore the capabilities of drones and harness their potential for automating delivery processes.

Initially, we embarked on constructing a drone, understanding its components, and familiarizing ourselves with the necessary technical aspects. This stage allowed us to gain practical knowledge of drone assembly, enabling us to comprehend the intricate mechanics involved. With a fully functional drone at our disposal, we proceeded to integrate it with Raspberry Pi, a powerful and versatile microprocessor. This integration provided us with enhanced control and automation capabilities, paving the way for the next phase of our project.

To augment the drone's functionality, we integrated a camera, enabling us to capture real-time footage and enhance the situational awareness of the system. This addition proved invaluable in terms of surveillance, navigation, and even remote monitoring because of hardware constraints. Moreover, we incorporated a payload box, expanding the drone's capacity to transport goods and revolutionize the delivery process. This innovation opened up possibilities for faster, more efficient, and safer delivery operations.

Throughout the project, we encountered various challenges, ranging from technical hurdles to logistical considerations. However, through collaboration, perseverance, and problem-solving, we overcome these obstacles, ultimately achieving a successful drone delivery system.

Looking ahead, the implications of our project are promising.The drone’s own weight is 1.3kg and it can lift payload up to 500 grams of weight. Its flight time is around 10 minutes with a battery of 2400 mAh. The drone delivery system we have developed can revolutionize various industries, from e-commerce to emergency services, offering swift, reliable, and cost-effective solutions. Additionally, our project has provided us with invaluable insights into drone technology, automation, and the integration of different components.

In conclusion, our project not only enabled us to build and automate a drone but also equipped us with the knowledge and skills to explore the vast possibilities of drone delivery systems in the future. With continued advancements in technology and regulatory frameworks, we are confident that drones will play an increasingly integral role in shaping the future of logistics and transportation.

Pictures of Drone :-


Gemei

Video Link :-

Prototype for Automated Drone Delivery Project LNMIIT

Bibliography:

1. DroneKit-Python Documentation: https://dronekit-python.readthedocs.io/en/latest/
2. openCv tutorials : https://www.youtube.com/watch?v=kdLM6AOd2vc&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K
3. pixhawk documentation: https://ardupilot.org/copter/docs/common-pixhawk-overview.html
4. quadcopter making tutorial:
https://www.youtube.com/watch?v=OWaXIK9sHeE
https://www.youtube.com/watch?v=k0GBPQ5MfEA
https://www.youtube.com/watch?v=QAe51IMY-kc
https://www.youtube.com/watch?v=kB9YyG2V-nA
5. RPi documentation : https://www.raspberrypi.com/documentation/
6. ArduPilot documentation: https://ardupilot.org/ardupilot/
7. Aruco Marker detection: https://www.youtube.com/watch?v=UlM2bpqo_o0
8. Drone Motor : https://www.engineersgarage.com/how-to-select-drone-motors/
9. https://www.youtube.com/watch?v=CUEbTx1Fh1w