Automated Drone System

Submitted By-

Vishesh Kumar Mishra

Submitted to-

Ankita Wadhawan

I. Introduction

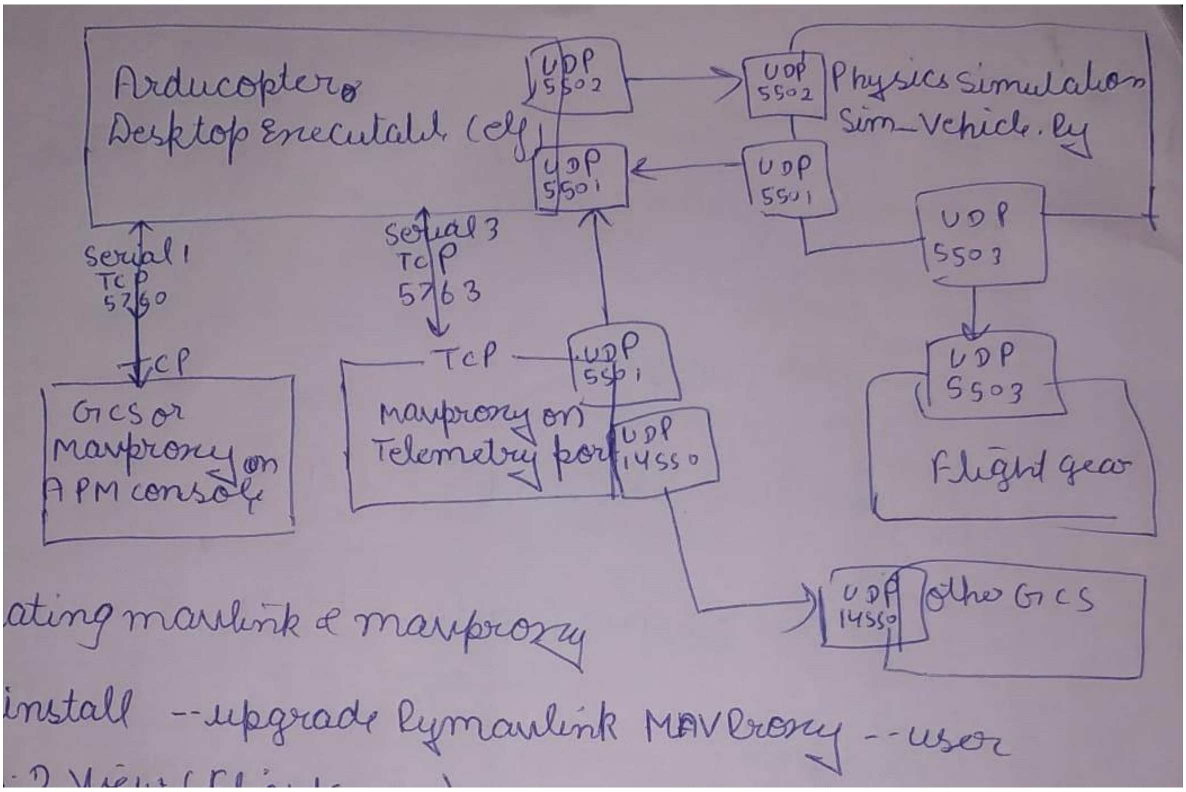
The project is to design an autonomous flying drone, specifically a quadcopter. The drone is fitted with a GPS tracking system and programmed to be able to autonomously fly from one location to another using GPS coordinates. Significant consideration is given to safety and ruggedness due to the possibility of collision with a variety of objects(this part is still under developement). In addition to collisions, the drone is also rugged enough to operate during moderately windy conditions. The goal of the project is to act as a proof of concept for autonomous flying drones without any manual control .

II. Problem Definition

Creating a GPS guided drone holds numerous challenges, chief of which is flight. In order to fly the drone uses four motors at the end of long arms that stick out from a central hub. The hub is where the flight controller and battery are located so that their weight doesn't throw of the drone's delicate balance. Furthermore the motors cannot adjust the pitch of the blades attached to them leaving their rotational speed the main control mechanism. Orientation information for a flying drone is usually discussed using just three key elements pitch, roll and yaw . This methodology of control wouldn't work however if it weren't for the fact that there are two motors spinning counter clockwise and two spinning clockwise . A GPS receiver on the central hub tells the drone where it currently is by communicating with several GPS satellites. This information is used in conjunction with a user selected destination to get a path of travel. To simplify the problem many drones fly at an altitude free of obstacles, as the quadcopter for this project is meant to, so sensors for detecting nearby objects are unnecessary and thus are not included in future it will be inclueded for lw level flying.

III. Project Design

The design of the drone is broken down into two large subcategories of hardware and software.In this project we are mainly focusing on software part.The software part includes linux system running inside respberry pi module which will be connected with flight controller module of our drone which will send the control signal to our drone in order to smooth funtioning of drone.The linux operating system will be running a python script capable in flying the drone in fully autonomus mode including takingoff,travelling on desired location,landing and returning back to home.Here we are running that same python script on virtually created environment which will mimic the real life scenerio of flying a drone .This vertual environment includes repositories from Ardupiolet and configuring it with simulated drone environment(SITL).Which recieves the controlling signal from python script and make the simulated drone do the same.This python script communicates with simulated drone with mavproxy and mavlink protocol on udp port of 5501,5502,5503 and tcp port of 5760 and 5763



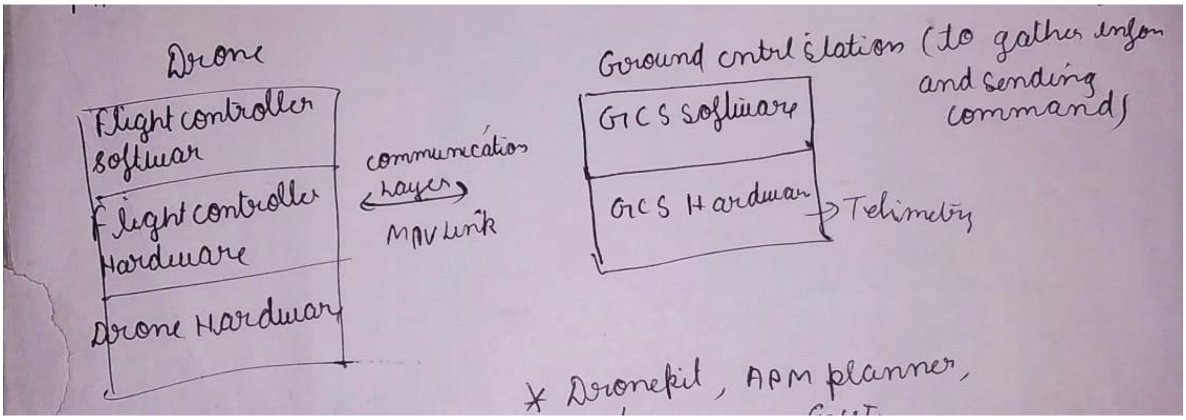
sim_vehicle.py is an python file for launching the SITL in order to simulate the drone.Since this whole thing is running on linux ,we have launch the simulation manually by giving command "sim_vehicle.py --map --console".--map is to launch map in order to see real time position of the drone and --console is to come up with a terminal which shows real time data of drone like height,position,battery percentage,velocity,latitude and longitude.

After launching the simulation we can run our python code to control our drone which will automatically controle the drone by sending the signals for performing an certain action and providing the drone a path to follow in order to reach its destination and return back to home.

This python script communicts on mavlink protocol as stated avove.

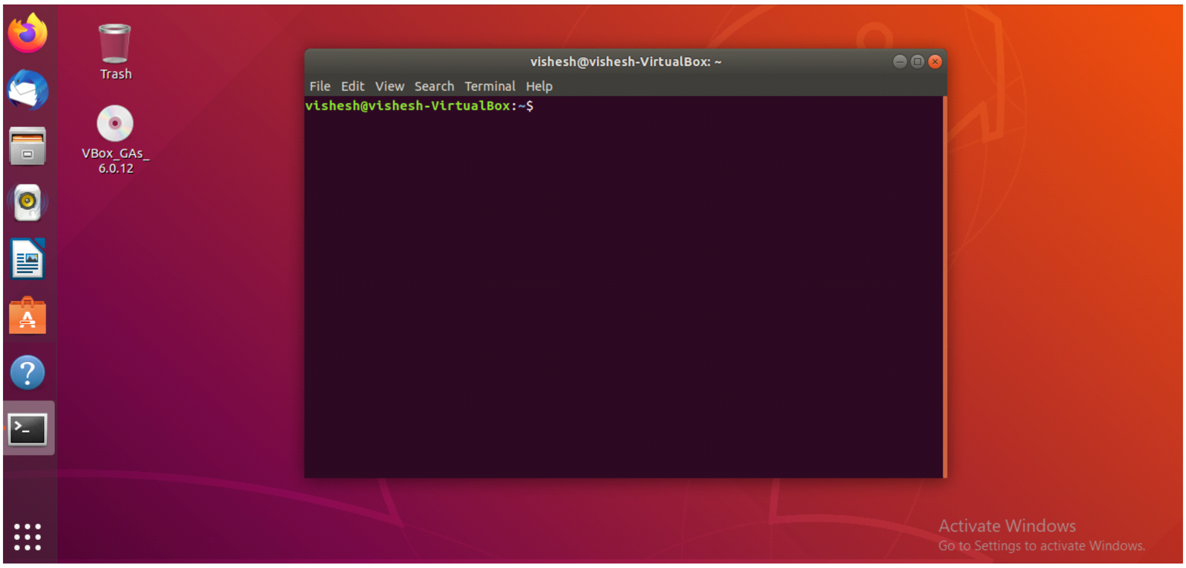
The flight controller is embaded with GPS system which helps in providing real time position of drone.The cordinate of drone is then sended to python script running in background in order to make the path to be followed by drone.

Basic System Arcitecture

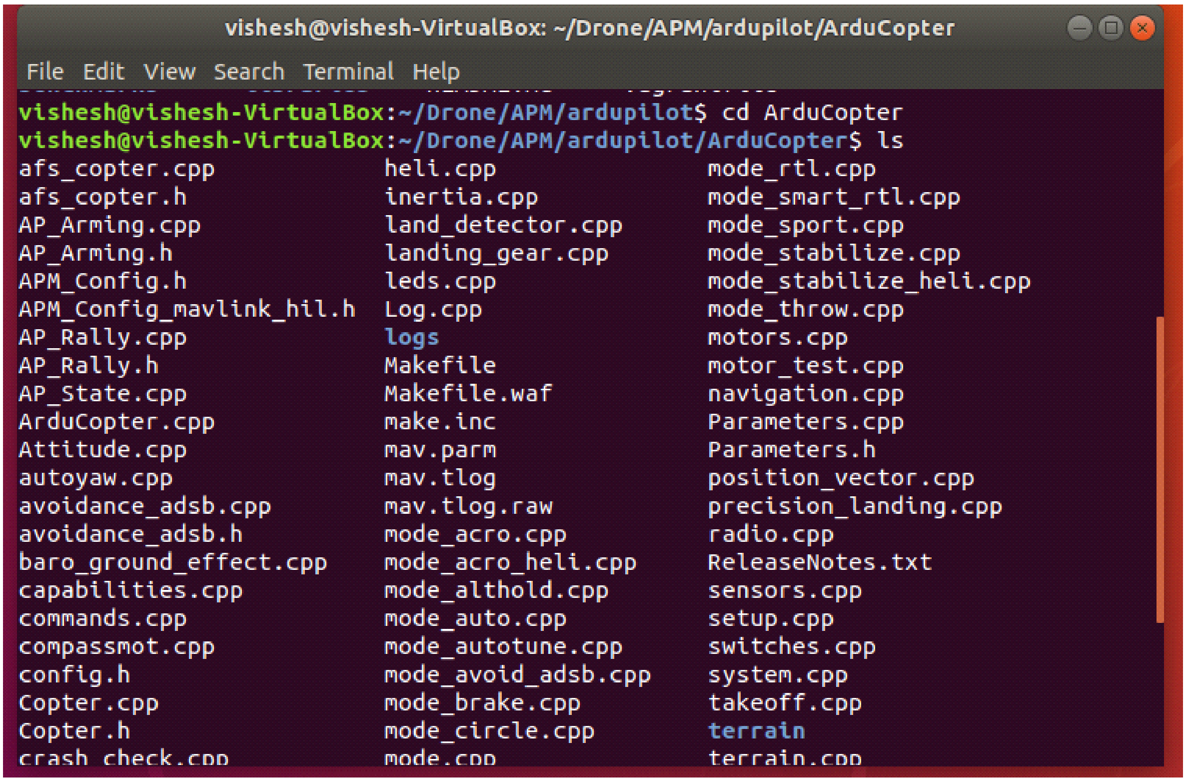


IV. Methodology

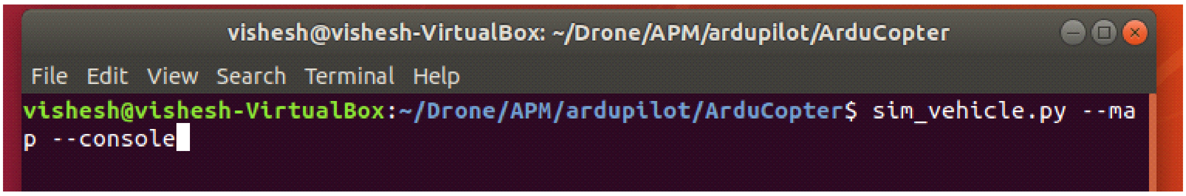
Getting Started



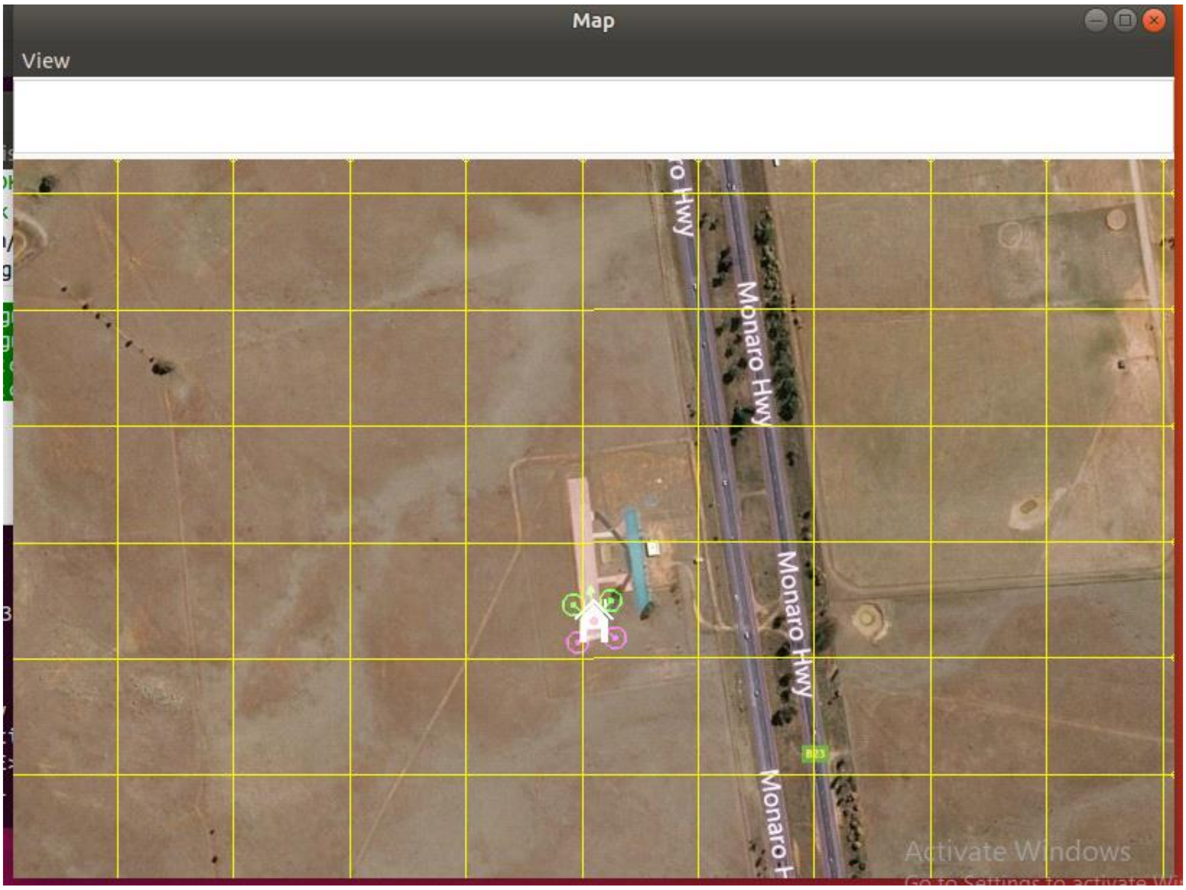
Required Basic Files of SITL



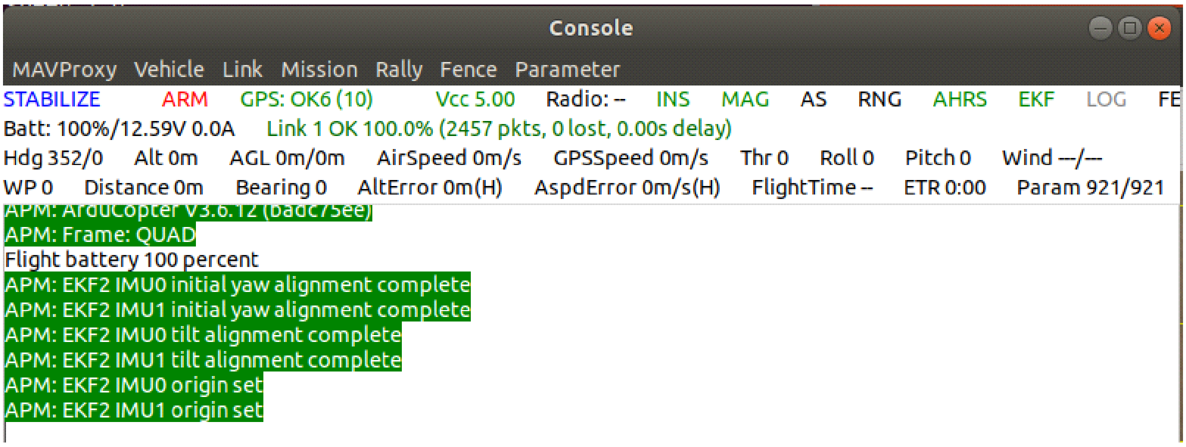
Launching of SITL



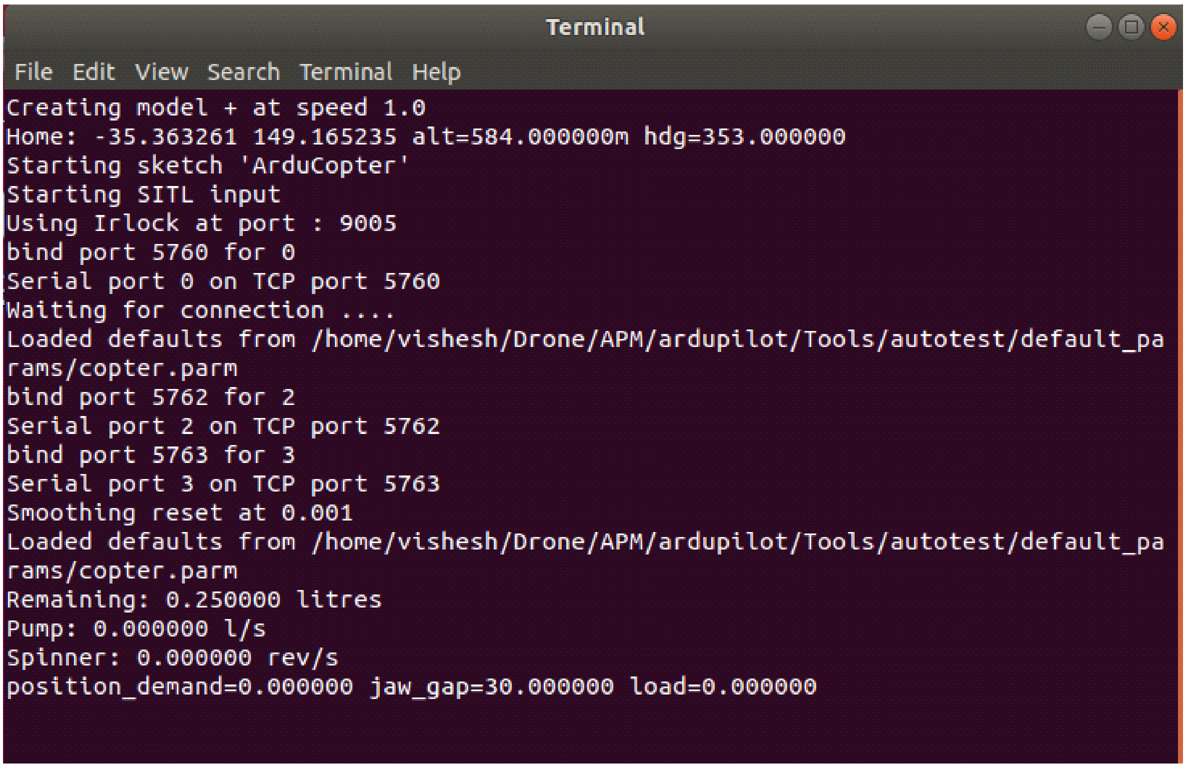
Map



Console

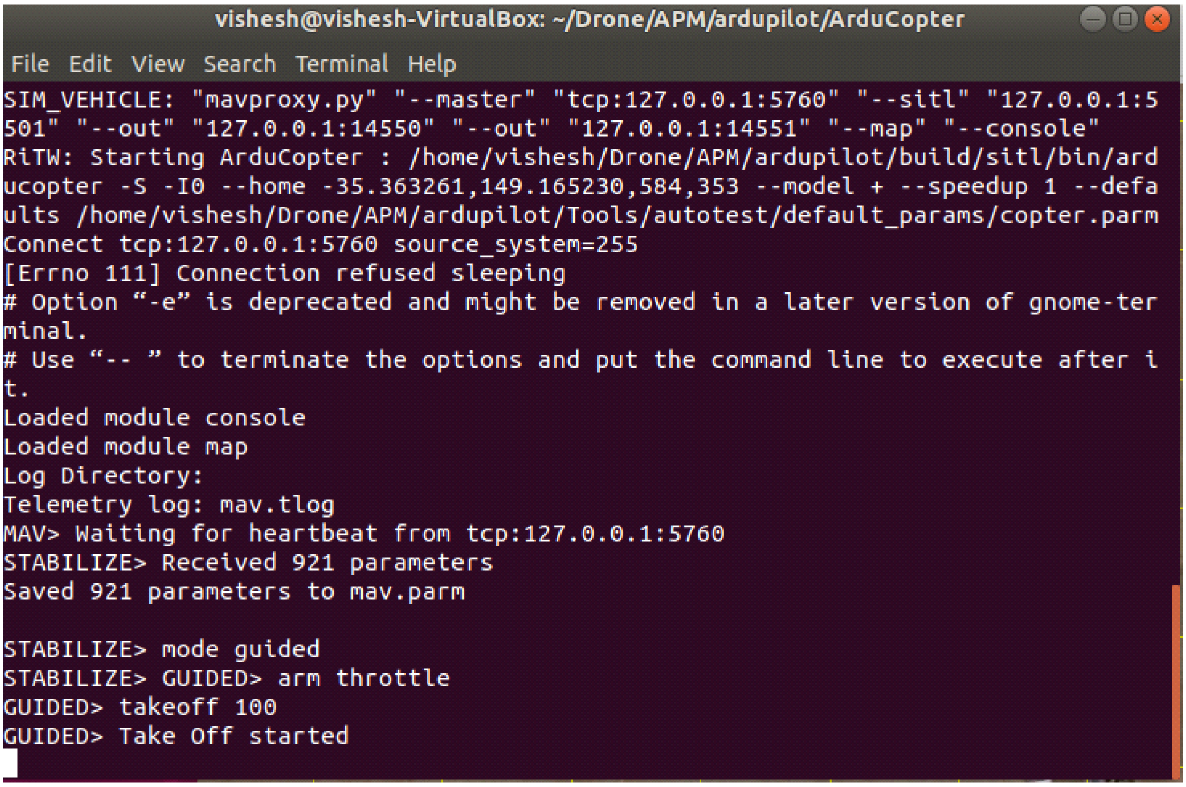


Mavproxy terminal



```
Terminal
File  Edit  View  Search  Terminal  Help
Creating model + at speed 1.0
Home: -35.363261 149.165235 alt=584.000000m hdg=353.000000
Starting sketch 'ArduCopter'
Starting SITL input
Using Irlock at port : 9005
bind port 5760 for 0
Serial port 0 on TCP port 5760
Waiting for connection ....
Loaded defaults from /home/vishesh/Drone/APM/ardupilot/Tools/autotest/default_pa
rams/copter.parm
bind port 5762 for 2
Serial port 2 on TCP port 5762
bind port 5763 for 3
Serial port 3 on TCP port 5763
Smoothing reset at 0.001
Loaded defaults from /home/vishesh/Drone/APM/ardupilot/Tools/autotest/default_pa
rams/copter.parm
Remaining: 0.250000 litres
Pump: 0.000000 l/s
Spinner: 0.000000 rev/s
position_demand=0.000000 jaw_gap=30.000000 load=0.000000
```

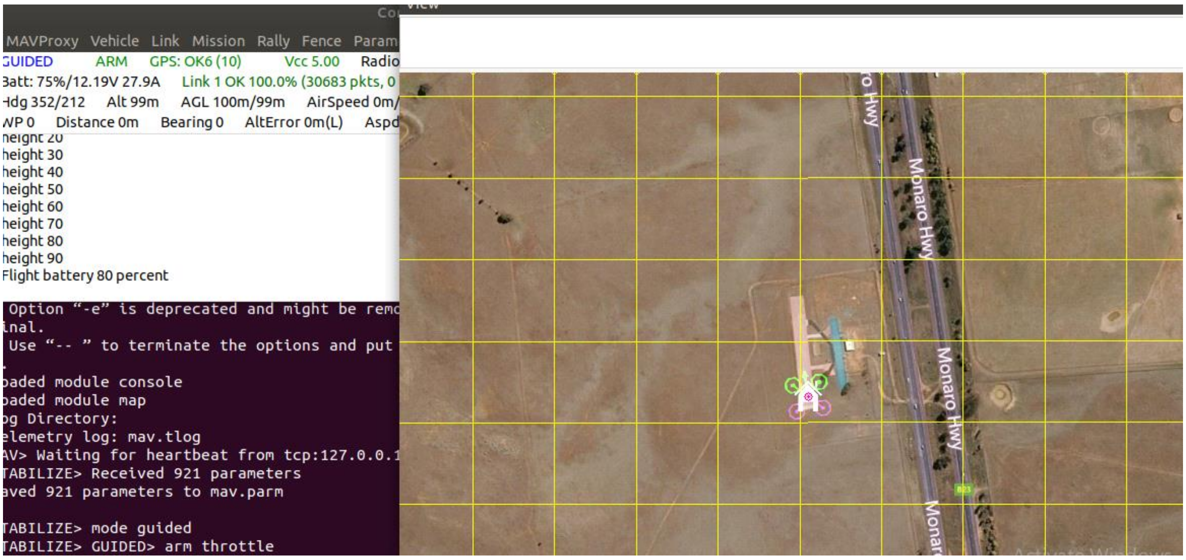
Getting basic commands from python script(GUIDED Mode)



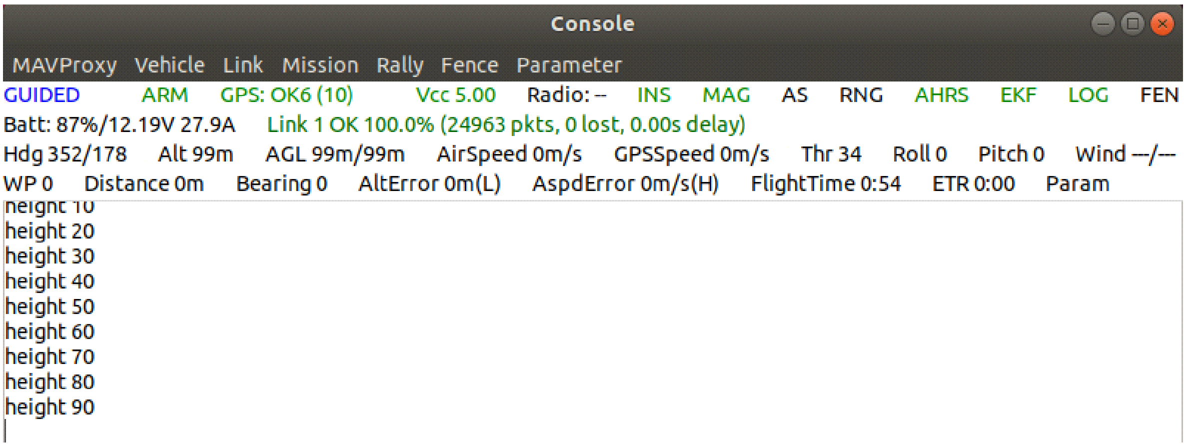
```
vishesh@vishesh-VirtualBox: ~/Drone/APM/ardupilot/ArduCopter
File  Edit  View  Search  Terminal  Help
SIM_VEHICLE: "mavproxy.py" "--master" "tcp:127.0.0.1:5760" "--sitl" "127.0.0.1:5
501" "--out" "127.0.0.1:14550" "--out" "127.0.0.1:14551" "--map" "--console"
RiTW: Starting ArduCopter : /home/vishesh/Drone/APM/ardupilot/build/sitl/bin/ard
ucopter -S -I0 --home -35.363261,149.165230,584,353 --model + --speedup 1 --defa
ults /home/vishesh/Drone/APM/ardupilot/Tools/autotest/default_params/copter.parm
Connect tcp:127.0.0.1:5760 source_system=255
[Errno 111] Connection refused sleeping
# Option "-e" is deprecated and might be removed in a later version of gnome-ter
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
Loaded module console
Loaded module map
Log Directory:
Telemetry log: mav.tlog
MAV> Waiting for heartbeat from tcp:127.0.0.1:5760
STABILIZE> Received 921 parameters
Saved 921 parameters to mav.parm

STABILIZE> mode guided
STABILIZE> GUIDED> arm throttle
GUIDED> takeoff 100
GUIDED> Take Off started
```

Initial position after taking off

Height given by script is 100 meters for testing purpose.We can see its altitude as 99 m in console.



Getting height data on console

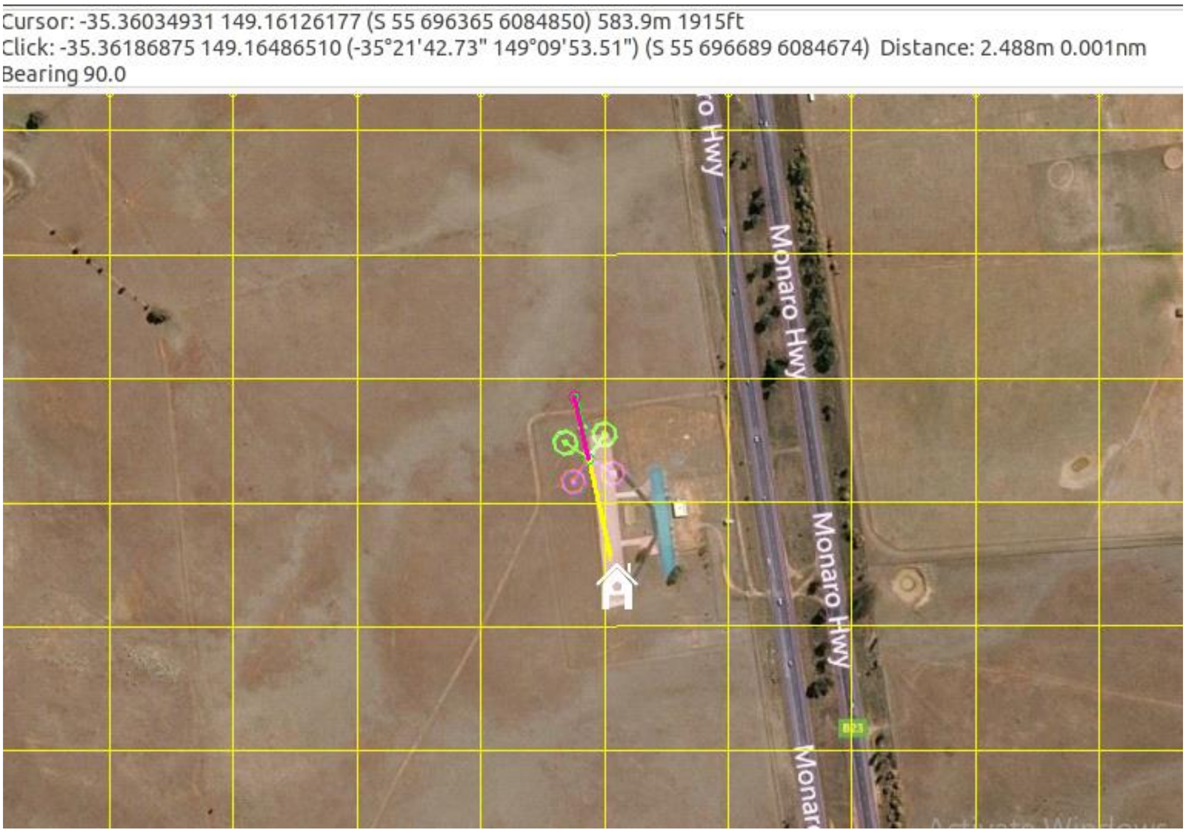


Sending Latitude and Longitude of location

Script sends the location of destination after configuring it through GPS and it will make an path for that

```
vishesh@vishesh-VirtualBox: ~/Drone/APM/ardupilot/ArduCopter
File  Edit  View  Search  Terminal  Help
501" "--out" "127.0.0.1:14550" "--out" "127.0.0.1:14551" "--map" "--console"
RiTW: Starting ArduCopter : /home/vishesh/Drone/APM/ardupilot/build/sitl/bin/ard
ucopter -S -I0 --home -35.363261,149.165230,584,353 --model + --speedup 1 --defa
ults /home/vishesh/Drone/APM/ardupilot/Tools/autotest/default_params/copter.parm
Connect tcp:127.0.0.1:5760 source_system=255
[Errno 111] Connection refused sleeping
# Option "-e" is deprecated and might be removed in a later version of gnome-ter
minal.
# Use "-- " to terminate the options and put the command line to execute after i
t.
Loaded module console
Loaded module map
Log Directory:
Telemetry log: mav.tlog
MAV> Waiting for heartbeat from tcp:127.0.0.1:5760
STABILIZE> Received 921 parameters
Saved 921 parameters to mav.parm

STABILIZE> mode guided
STABILIZE> GUIDED> arm throttle
GUIDED> takeoff 100
GUIDED> Take Off started
Guided (-35.36186875467945, 149.16486510017734) 100.0
```

In its way to destination



On the desired position

Returning to home after performing its task

V.Code-Snippet

The following codes are combined through automated scripts which are not included in this report.

(I)Performence test

```
from __future__ import print_function

from dronekit import connect

from pymavlink import mavutil

import time

import sys

from datetime import datetime

import argparse

parser = argparse.ArgumentParser(description='Generates max, min and current interval
between message sent and ack recieved. Will start and connect to SITL if no connection
string specified.')

parser.add_argument('--connect',


               help="vehicle connection target string. If not specified, SITL automatically started
and used.")

args = parser.parse_args()

connection_string=args.connect

if not connection_string:


    import dronekit_sitl


    sitl = dronekit_sitl.start_default()


    connection_string = sitl.connection_string()


print('Connecting to vehicle on: %s' % connection_string)

vehicle = connect(connection_string, wait_ready=True)
```

```
def cur_usec():

rom __future__ import print_function

from dronekit import connect

from pymavlink import mavutil

import time

import sys

from datetime import datetime

#Set up option parsing to get connection string

import argparse

parser = argparse.ArgumentParser(description='Generates max, min and current interval
between message sent and ack recieved. Will start and connect to SITL if no connection
string specified.')

parser.add_argument('--connect',


                    help="vehicle connection target string. If not specified, SITL automatically started
and used.")

args = parser.parse_args()

connection_string=args.connect

#Start SITL if no connection string specified

if not connection_string:

    import dronekit_sitl

    sitl = dronekit_sitl.start_default()

    connection_string = sitl.connection_string()

# Connect to the Vehicle

print('Connecting to vehicle on: %s' % connection_string)

vehicle = connect(connection_string, wait_ready=True)

#global vehicle

def cur_usec():
```

```
    """Return current time in usecs"""

    # t = time.time()

    dt = datetime.now()
    t = dt.minute * 60 + dt.second + dt.microsecond / (1e6)
    return t
class MeasureTime(object):
    def __init__(self):
        self.prevtime = cur_usec()
        self.previnterval = 0
        self.numcount = 0
        self.reset()
    def reset(self):
        self.maxinterval = 0
        self.mininterval = 10000
    def log(self):
        #print "Interval", self.previnterval
        #print "MaxInterval", self.maxinterval
        #print "MinInterval", self.mininterval
        sys.stdout.write('MaxInterval: %s\tMinInterval: %s\tInterval: %s\r' %
(self.maxinterval,self.mininterval, self.previnterval) )
        sys.stdout.flush()
    def update(self):
        now = cur_usec()
        self.numcount = self.numcount + 1
        self.previnterval = now - self.prevtime
        self.prevtime = now
        if self.numcount>1: #ignore first value where self.prevtime not reliable.
```

```
            self.maxinterval = max(self.previnterval, self.maxinterval)
            self.mininterval = min(self.mininterval, self.previnterval)
            self.log()
acktime = MeasureTime()
#Create COMMAND_ACK message listener.
@vehicle.on_message('COMMAND_ACK')
def listener(self, name, message):
    acktime.update()
    send_testpackets()
def send_testpackets():
    #Send message using `command_long_encode` (returns an ACK)
    msg = vehicle.message_factory.command_long_encode(
                                        1, 1,    # target system, target component

                                        #mavutil.mavlink.MAV_CMD_DO_SET_RELAY, #command
                                        mavutil.mavlink.MAV_CMD_DO_SET_ROI, #command

                                        0, #confirmation

                                        0, 0, 0, 0, #params 1-4

                                        0,

                                        0,

                                        0

                                        )
```

```
    vehicle.send_mavlink(msg)
#Start logging by sending a test packe
send_testpackets()
print("Logging for 30 seconds")
for x in range(1,30):
    time.sleep(1)
    # Close vehicle object before exiting script
vehicle.close()
if not args.connect:
    # Shut down simulator if it was started.
    sitl.stop()   # t = time.time()
    dt = datetime.now()
    t = dt.minute * 60 + dt.second + dt.microsecond / (1e6)
    return t
class MeasureTime(object):
    def __init__(self):
        self.prevtime = cur_usec()
        self.previnterval = 0
        self.numcount = 0
        self.reset()
    def reset(self):
        self.maxinterval = 0
        self.mininterval = 10000
    def log(self):
        #print "Interval", self.previnterval
```

```
        #print "MaxInterval", self.maxinterval

        #print "MinInterval", self.mininterval

        sys.stdout.write('MaxInterval: %s\tMinInterval: %s\tInterval: %s\r' %
(self.maxinterval,self.mininterval, self.previnterval) )

        sys.stdout.flush()

    def update(self):

        now = cur_usec()

        self.numcount = self.numcount + 1

        self.previnterval = now - self.prevtime

        self.prevtime = now

        if self.numcount>1: #ignore first value where self.prevtime not reliable.


            self.maxinterval = max(self.previnterval, self.maxinterval)


            self.mininterval = min(self.mininterval, self.previnterval)


            self.log()






acktime = MeasureTime()

#Create COMMAND_ACK message listener.

@vehicle.on_message('COMMAND_ACK')

def listener(self, name, message):

    acktime.update()

    send_testpackets()

def send_testpackets():
```

```
    #Send message using `command_long_encode` (returns an ACK)
    msg = vehicle.message_factory.command_long_encode(
                                            1, 1,    # target system, target component

                                            #mavutil.mavlink.MAV_CMD_DO_SET_RELAY, #command
                                            mavutil.mavlink.MAV_CMD_DO_SET_ROI, #command
                                            0, #confirmation
                                            0, 0, 0, 0, #params 1-4
                                            0,
                                            0,
                                            0
                                            )


    vehicle.send_mavlink(msg)
#Start logging by sending a test packet
send_testpackets()
print("Logging for 30 seconds")
for x in range(1,30):
    time.sleep(1)
# Close vehicle object before exiting script
vehicle.close()
if not args.connect:
    # Shut down simulator if it was started.
sitl.stop()
```

(II) Sending and Bringing back

```
from __future__ import print_function
```

```
import time

from dronekit import connect, VehicleMode, LocationGlobalRelative



# Set up option parsing to get connection string

import argparse

parser = argparse.ArgumentParser(description='Commands vehicle using
vehicle.simple_goto.')

parser.add_argument('--connect',

                    help="Vehicle connection target string. If not specified, SITL automatically
started and used.")

args = parser.parse_args()


connection_string = args.connect

sitl = None
```

```
# Start SITL if no connection string specified

if not connection_string:

    import dronekit_sitl

    sitl = dronekit_sitl.start_default()

    connection_string = sitl.connection_string()




# Connect to the Vehicle

print('Connecting to vehicle on: %s' % connection_string)

vehicle = connect(connection_string, wait_ready=True)
```

```
def arm_and_takeoff(aTargetAltitude):

    """

    Arms vehicle and fly to aTargetAltitude.

    """



    print("Basic pre-arm checks")

    # Don't try to arm until autopilot is ready

    while not vehicle.is_armable:

        print(" Waiting for vehicle to initialise...")

        time.sleep(1)



    print("Arming motors")

    # Copter should arm in GUIDED mode

    vehicle.mode = VehicleMode("GUIDED")
```

```
vehicle.armed = True


# Confirm vehicle armed before attempting to take off
while not vehicle.armed:
    print(" Waiting for arming...")
    time.sleep(1)


print("Taking off!")
vehicle.simple_takeoff(aTargetAltitude)  # Take off to target altitude


# Wait until the vehicle reaches a safe height before processing the goto
#  (otherwise the command after Vehicle.simple_takeoff will execute
#   immediately).
while True:
```

```
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)

        # Break and return from function just below target altitude.

        if vehicle.location.global_relative_frame.alt >= aTargetAltitude * 0.95:

            print("Reached target altitude")

            break

        time.sleep(1)




arm_and_takeoff(10)



print("Set default/target airspeed to 3")

vehicle.airspeed = 3



print("Going towards first point for 30 seconds ...")
```

```
point1 = LocationGlobalRelative(-35.361354, 149.165218, 20)

vehicle.simple_goto(point1)



# sleep so we can see the change in map

time.sleep(30)



print("Going towards second point for 30 seconds (groundspeed set to 10 m/s) ...")

point2 = LocationGlobalRelative(-35.363244, 149.168801, 20)

vehicle.simple_goto(point2, groundspeed=10)



# sleep so we can see the change in map

time.sleep(30)



print("Returning to Launch")
```

```
vehicle.mode = VehicleMode("RTL")

# Close vehicle object before exiting script

print("Close vehicle object")

vehicle.close()

# Shut down simulator if it was started.

if sitl:

    sitl.stop()
```

VI.Result

A simulated quadcopter that is capable of being given a GPS coordinate using a PC and attempts to fly to that coordinate by running through the constructed program. This simulated is fully aumated it can fly upto many kelometer of distance without being manually controlled through radio controllers.

VII.Conclusion

After embading this whole system into a hardware it can be a something different.

It can be used in medical services,disaster management and as a monitaring system.

After integrating it with cloud system this drone can travel more than 50 to 60 km without facinc any problem.