Indian Institute of Technology (Indian School of Mines), Dhanbad

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT REPORT
SESSION (2019-20)
VIII SEM

Submitted To:

Mr. Tarachand Amgoth

Assistant Professor
IIT (ISM) Dhanbad

Submitted By:

Sambhavesh Haralalka (16JE002482)
Ankit Yadav (16JE001886)
Rahul Verma (16JE002649)
Saumya Singh (16JE001956)

Acknowledgement

We would like to express heartfelt gratitude and regards to our project guide Dr. Tarachand Amgoth, Department of Computer Science and Engineering, IIT (ISM) Dhanbad. We convey our humble thanks to him for his valuable cooperation, support and suggestion throughout the project work which made this project successful. We shall remain indebted throughout my life for his noble help and guidance.

We are thankful to all the faculties of the Department of Computer Science and Engineering, IIT (ISM) Dhanbad for their encouraging words and valuable suggestions towards the project work. Last but not the least we want to acknowledge the contribution of our parents, family members, and friends for their constant and never-ending motivation.

We would like to take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of the project.

Sambhavesh Haralalka (16JE002482)

Ankit Yadav (16JE001886)

Rahul Verma (16JE002649)

Saumya Singh (16JE001956)

Date: April 30, 2020

Place: IIT(ISM) Dhanbad

Abstract

One of the many sources of income in our country is Agriculture. The growth of the crops is based on various factors like temperature, humidity, rain, etc. All of these are natural factors and are not under the control of the farmer. With the significant rise in global warming on earth, we have witnessed many ill-fated forest fires like those of the Amazon rainforest and Australia due to which millions of hectares of green land were burnt to ashes. In addition, the task of sowing the seeds in a field by a farmer is very laborious and demanding which requires time and human effort as input.

So, the main objective of this project is to design a drone that can be used to spray seeds and increase the greenery of the earth using an efficient afforestation method.

In this report, we discuss in brief the algorithm used to cover the land along with a spraying mechanism by an unmanned drone along with its software specifications. This project will help us vegetate the lands that are prone to soil erosion, landslides or deforestation both naturally or due to human negligence by helping us to plant trees there economically and also help us to reduce the plantation time. This project will reduce the workload on farmers as they can use it to plant their seeds along with this it will also help to reduce calamities like erosion and landslides.

Contents

Introduction

"When you want to cover a huge area, manually planting trees is impractical. Certain terrains may not be accessible to people. Drones are helpful in such cases." This statement was made by Professor KPJ Reddy from the Department of Aerodynamics in IISc in an interview.

Revegetation programs characterized the forest habitation approaches and specialized scientific strategies, for example, seed planting. In India, different tree species are being utilized for revegetation. In any case, current progressing vegetation practices are confronting difficulties. The difficulties are identified with the preparation and the operation stage. Revegetation rehearses require a far-reaching evaluation of terrains in the planning stage. The conventional afforestation practices done by direct seeding through hand planting are confronting trouble. This technique requires considerable man force. Besides, the inclusion of workers is limited by access. Thus, planting cannot be done at zones that are hindered by slopes of hills and water bodies considering the security of the laborers. Concerning the productivity, it is evaluated that a drone can disperse at least 10 seeds/minute. Drones have the ability in terms of optical and transportation features. Consequently, through this venture, we present the utilization of an autonomous drone to help with the revegetation of lands. This project means to build up a drone with four rotors called a quadcopter that is equipped for scattering seeds in fields with a significant prospect for afforestation.

Farming in India comprises over 60% of occupation. It fills in as the foundation of our nation's economy. It is essential to improve the profitability and productivity of agribusiness. The process of spraying of seeds is essential. It takes more than a day to cover the entire land the farmer owns. The spraying of seeds will not only save the time of the farmer, but the same drone can also be used to spray pesticides for the farmer, which, if done manually, is a costly procedure for the health of farmers. As per the WHO report, it is recorded that more than 3 million farmers are affected by the pesticides and out of which more than 18000 die annually. This project aims to not only save time or money for people or just barren lands but to save lives as well.

Problem Statement

Given a polygonal field of land on which seeds are to be sprayed, the task is to design an algorithm for an automated drone which when fed with the input parameters, cover the entire field and sprays seeds at a regular interval defined by the user.

Input format-

- Latitude and longitude of each vertex of the polygon field.
- Latitude and longitude of the approximated centre of the filed.
- The regular interval at which the seeds are to be sprayed.
- The altitude at which the drone operates.

On receiving the inputs, the algorithm should calculate the positional coordinates at where the seeds are to be sprayed and generates a path that covers all the seed dispersion points optimally.

Chapter 1.
Installation guide

Walkthrough to set up the PC for successful execution

1.1 Update

Install the package updates on your linux system

```
sudo apt-get update
```

1.2 Python 2

Install python2 if not already installed.

```
sudo apt install python2
```

1.3 DroneKit

DroneKit-Python and the dronekit-sitl simulator can be installed using **pip**.
First of all you will need to install **pip** and **python-dev**:

```
sudo apt-get install python-pip python-dev

pip install dronekit
```

1.4 DroneKit – SITL

The tool is installed (or updated) on all platforms using the command:

```
pip install dronekit-sitl -UI
```

1.5 MAVProxy

```
pip install MAVProxy
```

1.6 APMPlanner2

Download the latest deb file for your machine from:

firmware.ardupilot.org/Tools/APMPlanner

Open a terminal window and go to the location where you downloaded the .deb file from step 2 and type the following command:

```
sudo dpkg -i apm_planner*.deb
```

The installation will likely fail because of missing dependencies. These dependencies can be installed with this command:

```
sudo apt-get -f install
```

Then retry the APMPlanner2 installation again:

```
sudo dpkg -i apm_planner*.deb
```

1.7 Running the code

Method 1:

Directly running the python scripts:

```
python filename.py
```

Method 2:

Run the APM Planner2 and minimize it.
Open 3 terminals and execute each of the following in each terminal:

a) First run the DroneKit simulator with desired vehicle and its attributes.

```
dronekit-sitl copter --home=10.0,20.0,0,180
```

Above command runs the start the simulation with copter at home location (latitude = 10.0 and longitude = 20.0 in above command) and 0 and 180 are copter parameters like model etc. It starts the SITL connection at: tcp:127.0.0.1:5760

b) Use MavProxy to replicate the connection to other links like:

```
mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out 127.0.0.1:14551
```

APM Planner should automatically connect to one of the out links from above command.

c) Now run the required python drone script drone_AUTO.py or drone_GUIDED.py as follows:

```
python drone_GUIDED.py --connect udp:127.0.0.1:14551
```

or

```
python drone_AUTO.py --connect udp:127.0.0.1:14551
```

Monitor the drone in APM Planner and terminal outputs.
For detailed terminal outputs view the Master.log generated by the code.

Chapter 2. Software Description

This Chapter focuses on the various Software's used in the project and their API's

2.1 About DroneKit

DroneKit-Python allows developers to design applications that work on an onboard companion computer and communicate with the ArduPilot flight controller using a low-latency link. The onboard applications can greatly increase the power of the autopilot, which will increase intelligence of vehicle behaviour, and functions that are computationally intensive (e.g. computer vision, path planning, or 3D modelling). DroneKit-Python is used for ground station applications, interacting with vehicles on a higher latency RF-link.

The API interacts with vehicles over MAV Link. It gives automatic access to an associated vehicle's telemetry, state and parameter data, and empowers both strategic and direct authority over vehicle movement and activities.

DroneKit-Python is an open source and community-driven project.

2.1.1 API Features

The API provides classes and methods to:

- It can connect to one or more vehicles from script
- It can get and set vehicle state and parameter data.
- Get asynchronous notification about state changes.
- Guide a drone/vehicle to given position (GUIDED mode).
- Send discretionary custom messages to control UAV movement and other equipment (GUIDED mode).
- Create and manage waypoint missions (AUTO mode).
- Override RC channel settings

2.1.2 Launching scripts

DroneKit-Python 2.0 apps are executed through a simple Python command prompt.

```
python some_python_script.py
```

2.2 Setting up a Simulated Vehicle (SITL)

The SITL (Software In The Loop) simulator allows us to create and test DroneKit-Python applications without a real time vehicle.

SITL can run on Linux , Mac and Windows, or within a virtual machine. It can be installed on the same computer as DroneKit, or on another computer on the same network..

2.2.1 DroneKit-SITL

DroneKit-SITL is the simplest as well as fastest way to run SITL on Windows, Linux (x86 architecture only), as well as Mac OS X. It is installed with the help of Python's *pip* tool on all mentioned platforms. It works by downloading and running already built vehicle binaries that are appropriate for the host operating system.

2.2.2 Running SITL

```
dronekit-sitl copter
```

2.3 MAVProxy

A MAVLink protocol proxy and ground station. MAVProxy is oriented towards command line operation, and is suitable for embedding in small autonomous vehicles or for using on ground control stations. It also features a number of graphical tools such as a slip map for satellite mapping view of the vehicle's location, and status console and several useful vehicle control modules.

2.3.1 Features

- MAVProxy is a command line-based application. There are modules remembered for MAVProxy to give a fundamental GUI.
- Can be simulated on a number of different devices.
- It's versatile; it should run on any POSIX OS with python, pyserial, and select() work calls, which implies Linux, OS X, Windows, and others.
- The light weight configuration implies it can run on little notebook easily.
- It underpins loadable modules, and has modules to help console/s, moving maps, joysticks, reception apparatus trackers and so forth.
- Tab-completion of orders.

2.4 APM Planner

APM Planner 2.0 represents an open-source ground station application for MAV link-based autopilots which also include APM and PX4/Pixhawk that can be run on Windows, Mac OSX, and Linux.

2.4.1 Features

- Arrange and align your ArduPilot or PX4 autopilot for self-governing vehicle control.
- Plan a target to achieve with GPS waypoints and control events.
- Associate a 3DR Radio to see live information and initiate orders in flight

Chapter 3. Documentation of drone_AUTO.py

Following chapters explain each line of the python script

drone_Auto.py

Importing header files:

In [ ]:
```
from __future__ import print_function
from dronekit import connect, VehicleMode, LocationGlobalRelative, Comma
nd
from pymavlink import mavutil
import os
import json
import urllib
import math
import time
import argparse
import logging , logging.handlers
```

Logging configuration:

In [ ]:
```
logging.basicConfig(filename = "Master.log" , level = logging.DEBUG , fo
rmat = "%(levelname)s: %(filename)s: %(funcName)s: %(lineno)d:
        %(message)s")
logVAR = logging.getLogger(__name__)
logVAR.setLevel(logging.DEBUG)
logFileHand = logging.FileHandler("drone_seed_AUTO.log")
logFileHand.setLevel(logging.DEBUG)
logFile_streamHandler = logging.StreamHandler()
logFile_streamHandler.setLevel(logging.ERROR)
logForVAR = logging.Formatter("%(levelname)s: %(filename)s: %(funcName)
s: %(lineno)d:                  %(message)s")
logFileHand.setFormatter(logForVAR)
logFile_streamHandler.setFormatter(logForVAR)
logVAR.addHandler(logFileHand)
logVAR.addHandler(logFile_streamHandler)
```

Functions Used:

1. distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):

This function calulates the ground distance between two points.

This function is a approximation therefore valid for only short distance.

In [ ]:
```
def distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):
        disLatitude = locPOINT2.lat - locPOINT1.lat
        disLongitude = locPOINT2.lon - locPOINT1.lon
        return math.sqrt((disLatitude*disLatitude) + (disLongitude*disLo
ngitude)) * 1.113195e5
```

2. calculateDistanceToCurrentPoint():

```
This function returns the distance to the current waypoint in meters.

If Home Location is given as input the function returns None
```

In [ ]:
```
def calculateDistanceToCurrentPoint():
        nextVehCommandInt = simDRONE.commands.next
        if nextVehCommandInt==0:
                return None
        nextVehCommand=simDRONE.commands[nextVehCommandInt-1] #zero inde
x
        latitute = nextVehCommand.x
        longitude = nextVehCommand.y
        altitude = nextVehCommand.z
        nextPointLoc = LocationGlobalRelative(latitute,longitude,altitud
e)
        disPoint = distanceBetweenTwoGeoPoints(simDRONE.location.global_
frame, nextPointLoc)
        return disPoint
```

3. armVehicleThenTakeOFF(flyingALT):

This function takes a altitude as a parameter then arms the simulated drone and then fly to the given altitude.

In [ ]:
```
def armVehicleThenTakeOFF(flyingALT):
        while not simDRONE.is_armable:
                logVAR.warning("Waiting for drone to get ready -----")
                time.sleep(1.0)
```

Change the mode of drone to GUIDED :

In [ ]:
```
        while (simDRONE.mode.name != "GUIDED"):
                simDRONE.mode = VehicleMode("GUIDED")
                time.sleep(0.2)
```

Now we confirm that simulated drone is armed before taking off

In [ ]:
```
        while not simDRONE.armed:
                simDRONE.armed = True
                logVAR.warning("Wait for simulated drone to get armed")
                time.sleep(0.5)

        print("Simulate Drone is taking off..")
        logVAR.info("Simulate Drone is taking off..")
        simDRONE.simple_takeoff(flyingALT)
```

Now we add a check to see whether drone has reached the safe height:

In [ ]:
```
        while True:
                height = flyingALT*0.95
                if simDRONE.location.global_relative_frame.alt >= heigh
t:
                        print("Drone has reached the height of %f" % (fl
yingALT))
                        logVAR.info("Drone has reached the height of %f"
% (flyingALT))
                        break
                logVAR.info("Height: %f < %f" % (simDRONE.location.globa
l_relative_frame.alt,height))
                time.sleep(1.0)
```

4. print_simDRONE_parameters():

This function list all the parameters of the simulated drone and stores it in log file.

In [ ]:
```
def print_simDRONE_parameters():
        logVAR.info ("Listing all the current simulated drone parameters
(`simDRONE.parameters`):")
        for para, ivalue in simDRONE.parameters.iteritems():
                logVAR.info (" Parameter : %s Current Value : %s" % (par
a,ivalue))
```

Main Body :

In [ ]:
```
startingLatitude = 0.0          #latitute variable
startingLongitude = 0.0         #longitude variable
startingAltitude = 0.0          #altitude variable
waypoint_file = ""      #stores the waypoint file name
```

Takes the lattitude, longitude and altitude value from USER and check if USER enters the correct values or not.

In [ ]:
```
while True:
        try:
                startingLatitude = float(input("Please enter the latitut
e of starting point:\n"))
                logVAR.debug("USER entered latitute value: %s",str(start
ingLatitude))
                if(startingLatitude<0 or startingLatitude>90):
                        print("Latitude value must be between 0 and 90")
                        continue
                startingLongitude = float(input("Please enter the longit
ude of starting point:\n"))
                logVAR.debug("USER entered longitude value: %s",str(star
tingLongitude))
                if(startingLongitude<0 or startingLongitude>180):
                        print("Langitude value must be between 0 and 18
0")
                        continue
                startingAltitude = float(input("Please enter the altitud
e for the drone:\n"))
                logVAR.debug("USER entered altitude value: %s",str(start
ingAltitude))
                if(startingAltitude<0):
                        print("Altitude value must be positive")
                        continue
                break
        except:
                logVAR.error("Oops!  That was no valid lat/lon or altitu
de.  Try again...")
```

Takes the waypoint file name from USER

In [ ]:
```
while True:
        waypoint_file = raw_input("Enter the waypoint file name with ext
ension:\n")
        if os.path.exists(waypoint_file):
                break
        else:
                print("Enter file does not exists. Please re enter corre
ct file")
                logVAR.error("Enter file does not exists.")
                continue
```

Now we set up parsing to get the connection string from user

In [ ]:
```
parsingVAR = argparse.ArgumentParser(description=' Seed Plantation using
drone.')
parsingVAR.add_argument('--connect', help="simDRONE connection string. S
ITL is automatically started if connection string not specified.")
argsVAR = parsingVAR.parse_args()
userConString = argsVAR.connect
sitlSIM = None
```

Start the SITL if user do not specify the connection string for the drone

In [ ]:
```
if not userConString:
        import dronekit_sitl
        sitlSIM = dronekit_sitl.start_default(lat=startingLatitude,lon=s
tartingLongitude)
        userConString = sitlSIM.connection_string()
```

Now connect to the simulated drone

In [ ]:
```
print('Connecting to simulated drone on: %s' % userConString)
logVAR.info('Connecting to simulated drone on: %s' % userConString)
simDRONE = connect(userConString, wait_ready=True)
```

Log drone parameters:

In [ ]:
```
print_simDRONE_parameters()
```

Download the simulated drone commands

In [ ]:
```
droneCMDS = simDRONE.commands
droneCMDS.wait_ready()
droneCMDS = simDRONE.commands
droneCMDS.clear()
line_count = 0    #Variable that keep track of total commands
```

Add command for starting location:

In [ ]:
```
droneCMD = Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_A
LT , mavutil.mavlink.MAV_CMD_NAV_WAYPOINT , 0, 0, 0, 0, 0, 0,startingLat
itude, startingLongitude,startingAltitude)
droneCMDS.add(droneCMD)
```

add command for all waypoints:

In [ ]:
```
with open(waypoint_file,"r") as way_p:
        for pt in way_p:
                current_line = pt.split(",")
                line_count +=1
                lat = float(current_line[0])
                lon = float(current_line[1])
                logVAR.debug ("Point: %f %f" %(lat, lon))
                droneCMD = Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_G
LOBAL_RELATIVE_ALT , mavutil.mavlink.MAV_CMD_NAV_WAYPOINT , 0, 0, 5, 0,
0, 0,lat, lon, startingAltitude)
                droneCMDS.add(droneCMD)
way_p.close()
```

Add command for returing to base:

In [ ]:
```
droneCMD = Command( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_A
LT, mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 0, 0, 0, 0,startingLatitu
de, startingLongitude,startingAltitude)
droneCMDS.add(droneCMD)
```

Now we upload all the waypoints to our simulated drone.

In [ ]:
```
print("Upload points to simulated drone..." )
logVAR.info("Upload points to simulated drone...")
droneCMDS.upload()
print("Drone is arming and taking off:")
logVAR.info("Drone is arming and taking off:")
armVehicleThenTakeOFF(startingAltitude)

print("Starting Seed Plantation mission")
logVAR.info("Starting Seed Plantation mission")
```

Reset command to first point i.e 0

In [ ]:
```
simDRONE.commands.next=0
```

To start the mission set the drone MODE to AUTO:

In [ ]:
```
while (simDRONE.mode.name != "AUTO"):
        simDRONE.mode = VehicleMode("AUTO")
        time.sleep(0.1)
```

Monitor mission

It calculates the distance to next waypoint at regular interval (here 1 sec) and if distance is < 1.5m we assume that drone has reached the point where it has to drop the seed and Seed Dropping is going on. (Thats why we see multiple dropping seed print statement in terminal)

When we reach the last point, RTL (Return to launch) command is executed by changing the drone mode to RTL.

In [ ]:
```
while True:
        nextVehCommandInt = simDRONE.commands.next
        print('Distance to next seed drop point (%s): %s' % (nextVehComm
andInt, calculateDistanceToCurrentPoint()))
        logVAR.info('Distance to next seed drop point (%s): %s' % (nextV
ehCommandInt, calculateDistanceToCurrentPoint()))
        if calculateDistanceToCurrentPoint()<1.5:
                print("Dropping Seed")
                logVAR.critical("Dropping Seed")
        if nextVehCommandInt==line_count+1:
                print("Drone is heading to start location or launch loca
tion")
                logVAR.info("Drone is heading to start location or launc
h location")
                break;
        time.sleep(1)

print('Return to base/helipad')
logVAR.critical("Return to base/helipad")
while (simDRONE.mode.name != "RTL"):
        simDRONE.mode = VehicleMode("RTL")
        time.sleep(0.1)
```

Close simulated drone object before terminating script

In [ ]:
```
print("Close simulated drone object")
logVAR.info("Close simulated drone object")
simDRONE.close()
```

Shut down simulator:

In [ ]:
```
if sitlSIM is not None:
        sitlSIM.stop()
print("Seed Plantation Completed...")
logVAR.info("Seed Plantation Completed...")
```

Chapter 4. Documentation of drone_GUIDED.py

Following chapters explain each line of the python script

drone_Guided.py

Importing header files:

In [ ]:
```
from __future__ import print_function
from pymavlink import mavutil
from dronekit import connect, VehicleMode, LocationGlobalRelative, Locat
ionGlobal,Command
import os
import json
import urllib
import math
import argparse
import time
import logging , logging.handlers
```

Logging configuration:

In [ ]:
```
logging.basicConfig(filename = "Master.log" , level = logging.DEBUG , fo
rmat = "%(levelname)s: %(filename)s: %(funcName)s: %(lineno)d:
        %(message)s")
logVAR = logging.getLogger(__name__)
logVAR.setLevel(logging.DEBUG)
logFileHand = logging.FileHandler("drone_seed_GUIDED.log")
logFileHand.setLevel(logging.DEBUG)
logFile_streamHandler = logging.StreamHandler()
logFile_streamHandler.setLevel(logging.ERROR)
logForVAR = logging.Formatter("%(levelname)s: %(filename)s: %(funcName)
s: %(lineno)d:                  %(message)s")
logFileHand.setFormatter(logForVAR)
logFile_streamHandler.setFormatter(logForVAR)
logVAR.addHandler(logFileHand)
logVAR.addHandler(logFile_streamHandler)
```

Custom Class for taking lat/lon and alt points

LAT_LON_ALT class takes three parameters x,y and z save them as lon,lat and alt variables of the class object.

In [ ]:
```
class LAT_LON_ALT:
        def __init__(self,x,y,z):
                self.lon = x
                self.lat = y
                self.alt = z
```

Functions Used:

1. distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):

This function calulates the ground distance between two points.

This function is a approximation therefore valid for only short distance.

In [ ]:
```
def distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):
        disLatitude = locPOINT2.lat - locPOINT1.lat
        disLongitude = locPOINT2.lon - locPOINT1.lon
        return math.sqrt((disLatitude*disLatitude) + (disLongitude*disLo
ngitude)) * 1.113195e5
```

2. armVehicleThenTakeOFF(flyingALT):

This function takes a altitude as a parameter then arms the simulated drone and then fly to the given altitude.

In [ ]:
```
def armVehicleThenTakeOFF(flyingALT):
        # Wait for autopilot to get ready
        while not simDRONE.is_armable:
                logVAR.warning("Waiting for drone to get ready -----")
                time.sleep(1.0)

        # Change the mode of drone to GUIDED :
        while (simDRONE.mode.name != "GUIDED"):
                simDRONE.mode = VehicleMode("GUIDED")
                time.sleep(0.2)
        # Now we confirm that simulated drone is armed before taking off
        while not simDRONE.armed:
                simDRONE.armed = True
                logVAR.warning("Wait for simulated drone to get armed")
                time.sleep(0.5)

        print("Simulate Drone is taking off..")
        logVAR.info("Simulate Drone is taking off..")
        simDRONE.simple_takeoff(flyingALT)

        # Now we add a check to see whether drone has reached the safe h
eight:

        while True:
                height = flyingALT*0.95
                if simDRONE.location.global_relative_frame.alt >= heigh
t:
                        print("Drone has reached the height of %f" % (fl
yingALT))
                        logVAR.info("Drone has reached the height of %f"
% (flyingALT))
                        break
                logVAR.info("Height: %f < %f" % (simDRONE.location.globa
l_relative_frame.alt,height))
                time.sleep(1.0)
```

3. goto(targetLocation):

At first we store the target location in Vehicle Global Relative Frame object and calculate the target distance.

Then using the mavlink and vehicle message factory send command to drone to move to next target location.

There is another way to send command to drone by using the inbuild function vehicle_simple_goto(). (But we are using our custom command)

Note: Take care of the coordinate system followed by different msg command (like WGS84 or Lat/lon system etc)

In [ ]:
```
def goto(targetLocation):
        # send command to simulated drone
        logVAR.debug("Target location lat: %f , lon: %f , alt: %f" % (ta
rgetLocation.lat,targetLocation.lon,targetLocation.alt))
        vc_in_loc = simDRONE.location.global_relative_frame
        simDRONE_initialLocation = LAT_LON_ALT(vc_in_loc.lon,vc_in_loc.l
at,vc_in_loc.alt)
        targetDistance = distanceBetweenTwoGeoPoints(simDRONE_initialLoc
ation, targetLocation)
        msg = simDRONE.message_factory.set_position_target_global_int_en
code( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, 0b0000
111111111000, targetLocation.lat*1e7, targetLocation.lon*1e7, targetLoca
tion.alt, 0, 0, 0, 0, 0, 0, 0, 0)
        simDRONE.send_mavlink(msg)
        logVAR.debug("Send Command Message to drone")
        # target = LocationGlobal(targetLocation.lat,targetLocation.lon,
targetLocation.alt)
        # simDRONE.airspeed=15
        # simDRONE.simple_goto(target)
```

CRITICAL :

There may be case that our msg command send above is dropped due to network failure. In that case drone will be stuck at a point.

To handle this we monitor the drone for 5 sec after sending the command. If drone doesnot move (i.e. targetDistance is still greater than 90% of that of at first sec) then we resend the msg command to drone.

Following code even handle the drone Mode change if any.

In [ ]:

```
        fiveSecondCheck = targetDistance
        fiveCounter = 1
        logVAR.debug("fiveSecondCheck distance: %f " % (fiveSecondChec
k))
        logVAR.debug("fiveCounter value: %d " % (fiveCounter))
        while True:
                logVAR.debug("mode: %s" % simDRONE.mode.name)
                while (simDRONE.mode.name != "GUIDED"):
                        simDRONE.mode = VehicleMode("GUIDED")
                        time.sleep(0.1)

                if fiveCounter == 1:
                        vc_loc = simDRONE.location.global_relative_frame
                        simDRONE_currentLocation = LAT_LON_ALT(vc_loc.lo
n,vc_loc.lat,vc_loc.alt)
                        fiveSecondCheck = distanceBetweenTwoGeoPoints(si
mDRONE_currentLocation, targetLocation)
                        logVAR.debug("fiveSecondCheck distance: %f " %
(fiveSecondCheck))
                        logVAR.debug("fiveCounter value: %d " % (fiveCou
nter))

                if fiveCounter >=5:
                        logVAR.debug("fiveSecondCheck distance: %f " %
(fiveSecondCheck))
                        logVAR.debug("fiveCounter value: %d " % (fiveCou
nter))
                        fiveCounter = 1
                        vc_loc = simDRONE.location.global_relative_frame
                        simDRONE_currentLocation = LAT_LON_ALT(vc_loc.lo
n,vc_loc.lat,vc_loc.alt)
                        currentDistanceToTarget = distanceBetweenTwoGeoP
oints(simDRONE_currentLocation, targetLocation)
                        logVAR.debug("fiveSecondCheck currentDistanceToT
arget distance: %f " % (currentDistanceToTarget))
                        if currentDistanceToTarget >= 0.9* fiveSecondChe
ck:
                                #resend the msg command to drone
                                simDRONE.send_mavlink(msg)
                                logVAR.critical("Last command message dr
opped. Resending the command message to drone")
                                logVAR.debug("Resend the command message
to drone.")

                vc_loc = simDRONE.location.global_relative_frame
                simDRONE_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_lo
c.lat,vc_loc.alt)
                remDistance=distanceBetweenTwoGeoPoints(simDRONE_current
Location, targetLocation)
                logVAR.info("Distance to next seed drop point: %f" % (re
mDistance))
                print("Distance to next seed drop point: %f" % (remDista
nce))
                if remDistance <= 1:
                        logVAR.info("Reached drop point")
                        break
                fiveCounter += 1
                time.sleep(1)
```

4. print_simDRONE_parameters():

This function list all the parameters of the simulated drone and stores it in log file

In [ ]:
```
def print_simDRONE_parameters():
        logVAR.info ("Listing all the current simulated drone parameters
(`simDRONE.parameters`):")
        for para, ivalue in simDRONE.parameters.iteritems():
                logVAR.info (" Parameter : %s Current Value : %s" % (par
a,ivalue))
```

5. startMission(startingLocation):

This function controls the planned mission of drone. Collect all the waypoints from the file and use goto() function to give commands to drone.

Once the drone reaches the required location we can drop the seed.

In [ ]:
```
def startMission(startingLocation):
        with open(waypoint_file,"r") as waypointFile:
                for pt in waypointFile:
                        current_line = pt.split(",")
                        nextLocation = LAT_LON_ALT(float(current_line
[1]),float(current_line[0]),startingLocation.alt)
                        logVAR.debug("Next location lat: %f , lon: %f ,
alt: %f",nextLocation.lat,nextLocation.lon,nextLocation.alt)
                        goto(nextLocation)
                        print("Dropping Seed")
                        logVAR.info("Dropping Seed")
        waypointFile.close()
```

Main Body :

In [ ]:
```
startingLocation = LAT_LON_ALT(0.0,0.0,0.0) #startingLocation variable
waypoint_file = ""      #stores the waypoint file name
```

Takes the lat lon and alt value from USER

In [ ]:
```
while True:
        try:
                startingLocation.lat = float(input("Please enter the lat
itute of starting point:\n"))
                logVAR.debug("USER entered latitute value: %s",str(start
ingLocation.lat))
                if(startingLocation.lat<0 or startingLocation.lat>90):
                        print("Latitude value must be between 0 and 90")
                        continue
                startingLocation.lon = float(input("Please enter the lon
gitude of starting point:\n"))
                logVAR.debug("USER entered longitude value: %s",str(star
tingLocation.lon))
                if(startingLocation.lon<0 or startingLocation.lon>180):
                        print("Langitude value must be between 0 and 18
0")
                        continue
                startingLocation.alt = float(input("Please enter the alt
itude for the drone:\n"))
                logVAR.debug("USER entered altitude value: %s",str(start
ingLocation.alt))
                if(startingLocation.alt<0):
                        print("Altitude value must be positive")
                        continue
                break
        except:
                logVAR.error("Oops!  That was no valid lat/lon or altitu
de.  Try again...")
```

Takes the waypoint file name from USER

In [ ]:
```
while True:
        waypoint_file = raw_input("Enter the waypoint file name with ext
ension:\n")
        if os.path.exists(waypoint_file):
                break
        else:
                print("Enter file does not exists. Please re enter corre
ct file")
                logVAR.error("Enter file does not exists.")
                continue
```

Now we set up parsing to get the connection string from user

In [ ]:
```
parsingVAR = argparse.ArgumentParser(description=' Demonstrates Seed Pla
ntation Mission in GUIDED mode.')
parsingVAR.add_argument('--connect', help="simDRONE connection string. S
ITL is automatically started if connection string not specified.")
argsVAR = parsingVAR.parse_args()
userConString = argsVAR.connect
sitlSIM = None
```

Start the SITL is user do not specify the connection string for the drone

In [ ]:
```
if not userConString:
        import dronekit_sitl
        sitlSIM = dronekit_sitl.start_default(lat=startingLocation.lat,l
on=startingLocation.lon)
        userConString = sitlSIM.connection_string()
```

Now connect to the simulated drone

In [ ]:
```
print('Connecting to simulated drone on: %s' % userConString)
logVAR.info('Connecting to simulated drone on: %s' % userConString)
simDRONE = connect(userConString, wait_ready=True)
```

Log simulated drone parameters:

In [ ]:
```
print_simDRONE_parameters()
```

In [ ]:
```
print("Drone is arming and taking off:")
logVAR.info("Drone is arming and taking off:")
armVehicleThenTakeOFF(startingLocation.alt)
```

Start the mission by calling the startMission() function

After completion of mission RTL (Return to Launch)

In [ ]:
```
print("Starting mission")
logVAR.info("Starting mission")

startMission(startingLocation)


print('Return to base/helipad')
logVAR.critical("Return to base/helipad")
while (simDRONE.mode.name != "RTL"):
        simDRONE.mode = VehicleMode("RTL")
        time.sleep(0.1)
```

Close simulated drone object before terminating script

In [ ]:
```
print("Close simulated drone object")
logVAR.info("Close simulated drone object")
simDRONE.close()
```

Shut down simulator.

In [ ]:
```
if sitlSIM is not None:
        sitlSIM.stop()
print("Seed Plantation Completed...")
logVAR.info("Seed Plantation Completed...")
```

Chapter 5. Documentation of waypoint_square.py

Following chapters explain each line of the python script

waypoint_square.py

In this algorithm we are given a square land that drone has to traverse. We traverse along the sides of the square in a zic-zac manner and add the point in our output file. Finally printing the output file.

importing headers

In [ ]:
```
from __future__ import print_function
import math
```

Custom Class for taking lat/lon points

POINT class takes two parameters x and y and save them as lon and lat variables of the class object.

In [ ]:
```
class POINT:
        def __init__(self,x,y):
                self.lat = x
                self.lon =y
```

Functions used:

1. calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

This function is used to calculate new GEO Point which is 'y' meters north and 'x' meters east of a Reference point.

Knowing the lat/long of reference point and y and x distance , it returns a POINT class object with new location lat/long value.

This function is an approximation therefore valid over small distances (1m to 1km). It is not valid when calculating points close to the earth poles.

In [1]:
```
def calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

        #Radius of earth
        earthRad=6378137.0

        #New point offset calculated in radian
        tempLAT = yNORTH/earthRad
        tempLON = xEAST/(earthRad*math.cos(math.pi*initialPOINT.lat/18
0))

        #Now calculate the new point in decimal degrees
        finalLAT = initialPOINT.lat + (tempLAT * 180/math.pi)
        finalLON = initialPOINT.lon + (tempLON * 180/math.pi)
        finalLOCATION = POINT(finalLAT,finalLON)
        return finalLOCATION
```

2. distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):

This function calulates the ground distance between two points.

This function is a approximation therefore valid for only short distance.

In [2]:
```
def get_distance_metres(aLocation1, aLocation2):
        disLatitude = locPOINT2.lat - locPOINT1.lat
        disLongitude = locPOINT2.lon - locPOINT1.lon
        return math.sqrt((disLatitude*disLatitude) + (disLongitude*disLo
ngitude)) * 1.113195e5
```

3. def generate_points(start_point,edge_size,seed_distance,polygon_hull)

This function calculates the waypoints in a spiral manner for plantation of seeds in a square field.

```
Input:
    alocation: location of centre of square in lat/lon
    aSize: half the side of square
    seed_distance: distance between two waypoints or seed plantation distance
Output:
    A txt file named:  waypoint_square.txt, stores the waypoints generated in a
sequence.
```

Opening the output file for writing purpose then calculating the left most bottom corner point of square and adding first point to file.

In [4]:
```
def generate_points(aLocation,aSize,seed_distance):

        f = open("waypoint_square.txt","w+")
        temp1 = calNewGeoLocationNE(aLocation, -aSize, -aSize)
        f.write(str(temp1.lat) + "," + str(temp1.lon) + '\n')
```

Here step size which is the number of jumps required to cover one side of the square = aSize/seed_distance.

Outer for loop moves us in horizontal direction i.e left - right.

Here we move only halfway as we have two inner loops which traverse from bottom-top and top-bottom.

In [ ]:
```
        for i in range (aSize/seed_distance):
                temp2 = temp1
```

Inner for loop 1. it moves us from bottom-top.

As we move from bottom to top we simultaniously add the points in our output file.

In [ ]:
```
                for j in range(2*aSize/seed_distance):
                        newpoint = calNewGeoLocationNE(temp2, seed_dista
nce, 0)
                        temp2 = newpoint
                        f.write(str(temp2.lat) + "," + str(temp2.lon) +
'\n')
```

Right shift operation is done here.

Here we make a move to the right and add the point to file.

In [ ]:
```
                shift1 = calNewGeoLocationNE(temp2, 0, seed_distance)
                temp2  = shift1
                f.write(str(temp2.lat) + "," + str(temp2.lon) + '\n')
```

Inner loop 2. here the movement is top-bottom

As we move from top to bottom we simultaniously add the points in our output file.

In [ ]:
```
                for j in range(2*aSize/seed_distance):
                        newpoint = calNewGeoLocationNE(temp2, -seed_dist
ance, 0)
                        temp2 = newpoint
                        f.write(str(temp2.lat) + "," + str(temp2.lon) +
'\n')
```

Right shift operation is done here.

Here we make a move to the right and add the point in our output file.

In [ ]:
```
                shift2 = calNewGeoLocationNE(temp2, 0, seed_distance)
                temp1 = shift2
                f.write(str(temp1.lat) + "," + str(temp1.lon) + '\n')
```

This loop computes the last waypoints along the last boundary of the convex structure.

In the above loops we move up-right-down-right. If we continue to move this way we will miss the last boundary of our square. So this loop does that for us.

In [ ]:
```
        for j in range(2*aSize/seed_distance):
                        newpoint = calNewGeoLocationNE(temp1, seed_dista
nce, 0)
                        temp1 = newpoint
                        f.write(str(temp1.lat) + "," + str(temp1.lon) +
'\n')
```

Main Body:

Taking user input for land info:

User enters the latitude and longitude of the center of the land under consideration. We further check if the values entered is valid or not.

In [ ]:
```
print("   This code generates the required waypoint for the drone \n")
print("Please enter the geo location of the centre of your field: \n")
in_lat = 0.0
in_lon = 0.0
while True:
        try:
                in_lat = float(input("Please enter the latitute of centr
e:\n"))
                if(in_lat<0 or in_lat>90):
                        print("Latitude value must be between 0 and 90")
                        continue
                in_lon = float(input("Please enter the longitude of cent
re:\n"))
                if(in_lon<0 or in_lon>180):
                        print("Langitude value must be between 0 and 18
0")
                        continue
                break
        except:
                print("Oops!  That was no valid lat/lon.  Try again...")
```

Create out custom POINT object for input:

In [ ]:
```
initial_location = POINT(in_lat,in_lon)
```

Taking user input for size of land:

User enters distance between the center and boundary of square land. We further check if the values entered is valid or not.

In [ ]:
```
side_size = 0
while True:
        try:
                side_size = int(input("Please enter the distance between
the center and field edge (i.e. side/2):\n"))
                break
        except:
                print("Oops!  Please insert positive interger value.  Tr
y again...")
```

Taking the seed distance as an input from the user

This distance plays a significant role while computing the waypoints. We also check if the distance is valid or not and display appropriate messages.

In [ ]:
```
distance = 0
while True:
        try:
                distance = int(input("Please enter the distance between
two waypoints:\n"))
                break
        except:
                print("Oops!  That was no valid distance.  Try agai
n...")
```

Calling the required function.

In [ ]:
```
generate_points(initial_location,side_size,distance)
print("\n   Waypoints are generated and stored in waypoint_square.txt fi
le. \n")
```

Chapter 6. Documentation of waypoint _convex.py

Following chapters explain each line of the python script

waypoint_convex.py

In this algorithm we try to fit a larger square on a convex structure and then traverse along the sides of the square in a zic-zac manner and while traversing check if the point lies within the convex structure or not. If it lies within the convex structure we add the point in our output file. Finally printing the output file.

importing headers

In [ ]:
```
from __future__ import print_function
import math
import numpy as np
import os
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
np.set_printoptions(precision=12)
```

Custom Class for taking lat/lon points

LAT_LON class takes two parameters x and y and save them as lon and lat variables of the class object.

In [ ]:
```
class LAT_LON:
        def __init__(self,x,y):
                self.lon = x
                self.lat = y
```

Functions used:

1. calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

This function is used to calculate new GEO Point which is 'y' meters north and 'x' meters east of a Reference point.

Knowing the lat/long of reference point and y and x distance , it returns a POINT class object with new location lat/long value.

This function is an approximation therefore valid over small distances (1m to 1km). It is not valid when calculating points close to the earth poles.

In [1]:
```
def calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

        #Radius of earth
        earthRad=6378137.0

        #New point offset calculated in radian
        tempLAT = yNORTH/earthRad
        tempLON = xEAST/(earthRad*math.cos(math.pi*initialPOINT.lat/18
0))

        #Now calculate the new point in decimal degrees
        finalLAT = initialPOINT.lat + (tempLAT * 180/math.pi)
        finalLON = initialPOINT.lon + (tempLON * 180/math.pi)
        finalLOCATION = LAT_LON(finalLON,finalLAT)
        return finalLOCATION
```

2. distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):

This function calulates the ground distance between two points.

This function is a approximation therefore valid for only short distance.

In [2]:
```
def distanceBetweenTwoGeoPoints(locPOINT1, locPOINT2):

        disLatitude = locPOINT2.lat - locPOINT1.lat
        disLongitude = locPOINT2.lon - locPOINT1.lon
        return math.sqrt((disLatitude*disLatitude) + (disLongitude*disLo
ngitude)) * 1.113195e5
```

3. def generate_points(start_point,edge_size,seed_distance,polygon_hull)

This function calculates the waypoints in a spiral manner for plantation of seeds in a convex field.

```
Input:
    start_point: location of starting point of fitted square in lat/lon
    edge_size: side length of square fitted on convex  figure
    seed_distance: distance between two waypoints or seed plantation distance
    polygon_hull: A shapely Polygon object that contains the  actual convex fig
ure

Output:
    A txt file named:  waypoint_convex.txt, stores the waypoints generated in a
sequence.
```

Opening the output file for writing purpose and checking if the initial point lies within the convex structure, if so add it to the output file as seed must be thrown on this spot.

In [4]:
```
def generate_points(start_point,edge_size,seed_distance,polygon_hull):

        output_file = open("waypoint_convex.txt","w+")
        func_tempVar1 = start_point
        shapely_tempVar1 = Point(func_tempVar1.lon,func_tempVar1.lat)
        if (polygon_hull.contains(shapely_tempVar1) or shapely_tempVar1.
within(polygon_hull) or polygon_hull.touches(shapely_tempVar1) ):
                output_file.write(str(func_tempVar1.lat) + "," + str(fun
c_tempVar1.lon) + '\n')
```

Adjusting the step size which is the number of jumps required to cover one side of the square.

In [ ]:
```
        step_size = edge_size/seed_distance
```

Outer for loop moves us in horizontal direction i.e left - right.

Here we move only halfway the stepsize as we have two inner loops which traverse from bottom-top and top-bottom.

In [ ]:
```
        for i in range (step_size/2):
                func_tempVar2 = func_tempVar1
```

Inner for loop 1. it moves us from bottom-top.

As we move from bottom to top we simultaniously check if the points lie within the convex structure if so we add them in our output file.

In [ ]:
```
                for j in range(step_size):
                        func_newpoint = calNewGeoLocationNE(func_tempVar
2, seed_distance, 0)
                        func_tempVar2 = func_newpoint
                        shapely_tempVar1 = Point(func_tempVar2.lon,func_
tempVar2.lat)
                        if (polygon_hull.contains(shapely_tempVar1) or s
hapely_tempVar1.within(polygon_hull) or polygon_hull.touches(shapely_tem
pVar1) ):
                                output_file.write(str(func_tempVar2.lat)
+ "," + str(func_tempVar2.lon) + '\n')
```

Right shift operation is done here.

Here we make a move to the right and check if the point lies inside or not.

In [ ]:
```
                func_shift1 = calNewGeoLocationNE(func_tempVar2, 0, seed
_distance)
                func_tempVar2 = func_shift1
                shapely_tempVar1 = Point(func_tempVar2.lon,func_tempVar
2.lat)
                if (polygon_hull.contains(shapely_tempVar1) or shapely_t
empVar1.within(polygon_hull) or polygon_hull.touches(shapely_tempVar1)
):
                        output_file.write(str(func_tempVar2.lat) + "," +
str(func_tempVar2.lon) + '\n')
```

Inner loop 2. here the movement is top-bottom

As we move from top to bottom we simultaniously check if the points lie within the convex structure if so we add them in our output file.

In [ ]:
```
                for j in range(step_size):
                            func_newpoint = calNewGeoLocationNE(func_tempVar
2, -seed_distance, 0)
                            func_tempVar2 = func_newpoint
                            shapely_tempVar1 = Point(func_tempVar2.lon,func_
tempVar2.lat)
                            if (polygon_hull.contains(shapely_tempVar1) or s
hapely_tempVar1.within(polygon_hull) or polygon_hull.touches(shapely_tem
pVar1) ):
                                        output_file.write(str(func_tempVar2.lat)
+ "," + str(func_tempVar2.lon) + '\n')
```

Right shift operation is done here.

Here we make a move to the right and check if the point lies inside or not.

In [ ]:
```
                func_shift2 = calNewGeoLocationNE(func_tempVar2, 0, seed
_distance)
                func_tempVar1 = func_shift2
                shapely_tempVar1 = Point(func_tempVar1.lon,func_tempVar
1.lat)
                if (polygon_hull.contains(shapely_tempVar1) or shapely_t
empVar1.within(polygon_hull) or polygon_hull.touches(shapely_tempVar1)
):
                            output_file.write(str(func_tempVar1.lat) + "," +
str(func_tempVar1.lon) + '\n')
```

This loop computes the last waypoints along the last boundary of the convex structure.

In the above loops we move up-right-down-right. If we continue to move this way we will miss the last boundary of our square fitting our convex structure. So this loop does that for us.

In [ ]:
```
        for j in range(step_size):
                func_newpoint = calNewGeoLocationNE(func_tempVar1, seed_
distance, 0)
                func_tempVar1 = func_newpoint
                shapely_tempVar1 = Point(func_tempVar1.lon,func_tempVar
1.lat)
                if (polygon_hull.contains(shapely_tempVar1) or shapely_t
empVar1.within(polygon_hull) or polygon_hull.touches(shapely_tempVar1)
):
                            output_file.write(str(func_tempVar1.lat) + "," +
str(func_tempVar1.lon) + '\n')
```

Main Body:

Taking user input for the file_name

This file contains the latitude and longitude of the land under consideration. We further check if the file name entered is valid or not.

In [ ]:
```
input_file = ""
while True:
        input_file = raw_input("Enter the file name with extension conta
ining lat long of corners of polygon:\n")
        if os.path.exists(input_file):
                break
        else:
                print("Enter file does not exists. Please re enter corre
ct file")
                continue
```

Taking the seed distance as an input from the user

This distance plays a significant role while computing the waypoints. We also check if the distance is valid or not and display appropriate messages.

In [ ]:
```
while True:
        try:
                seed_distance = int(input("Please enter the distance bet
ween two waypoints:\n"))
                break
        except:
                print("Oops!  That was no valid distance.  Try agai
n...")
```

Creating a latitude longitude list

Here we read from the input file the latitude and longitude of the land under the consideration and add them to the latlon_list.

In [ ]:
```
latlon_list = []
with open(input_file,"r") as input_f:
        for line in input_f:
                current_line = line.split(",")
                latlon_list.append([float(current_line[1]),float(current
_line[0])])
```

Converting the latlon_list to a numpy array and contructing the convex hull using the function Polygon.

In [ ]:
```
latlon_num = np.array(latlon_list )#,dtype = np.float64)
polygon_hull = Polygon(latlon_num)
```

Finding the minimum and maximum co-ordinates.

This helps us in find the square that is capable of containing the entire convex structure into it.

In [ ]:
```
min_col = np.amin(latlon_num,axis = 0)
max_col = np.amax(latlon_num,axis = 0)
```

Finding the 3 points using the min_col and max_col

This helps us to get the side lenght of the largest square that can fit our convex structure.

In [ ]:
```
t1 = LAT_LON(min_col[0],min_col[1])
t2 = LAT_LON(min_col[0],max_col[1])
t3 = LAT_LON(max_col[0],min_col[1])
```

Calling the get_distance_meters function.

cal_d contains the maximum distance of the of the three points we computed, which ultimately helps us to get the side of the square. The above way point function works on an assumption that the lenght of the side of the square should be even. So we have placed a check for that.

In [ ]:
```
cal_d = max(distanceBetweenTwoGeoPoints(t1,t2),distanceBetweenTwoGeoPoin
ts(t1,t3))
total_step = math.ceil(cal_d/seed_distance)
if total_step%2 == 0:
        cal_d = seed_distance*(total_step)
else:
        cal_d = seed_distance*(total_step+1)
```

Initialising the starting point for waypoint calculation and calling the required function.

In [ ]:
```
start_point = t1
generate_points(start_point,int(cal_d),seed_distance,polygon_hull)
print("\n   Waypoints are generated and stored in waypoint_square.txt fi
le. \n")
```

Chapter 7. Documentation of input_generator.py

Following chapters explain each line of the python script

input_generator.py

This is a python script used to get the latitude and longitude of the corner points of the field when user knows the distance of each point with respect to one reference point. (Ref Point lat/long must be known)

importing headers

In [1]:
```
from __future__ import print_function
import math
```

Custom Class for taking lat/lon points

POINT class takes two parameters x and y and save then as lon and lat variables of the class object.

In [2]:
```
class POINT:
        def __init__(self,x,y):
                self.lat = x
                self.lon = y
```

Functions used:

1. calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

This function is used to calculate new GEO Point which is 'y' meters north and 'x' meters east of a Reference point.

Knowing the lat/long of reference point and y and x distance , it returns a POINT class object with new location lat/long value.

This function is an approximation therefore valid over small distances (1m to 1km). It is not valid when calculating points close to the earth poles.

In [3]:
```
def calNewGeoLocationNE(initialPOINT, yNORTH, xEAST):

        #Radius of earth
        earthRad=6378137.0

        #New point offset calculated in radian
        tempLAT = yNORTH/earthRad
        tempLON = xEAST/(earthRad*math.cos(math.pi*initialPOINT.lat/18
0))

        #Now calculate the new point in decimal degrees
        finalLAT = initialPOINT.lat + (tempLAT * 180/math.pi)
        finalLON = initialPOINT.lon + (tempLON * 180/math.pi)
        finalLOCATION = POINT(finalLAT,finalLON)
        return finalLOCATION
```

Main Body:

Taking user input for land info:

User enters the latitude and longitude of the reference point of the land under consideration. We further check if the values entered is valid or not.

In [5]:
```
print("   This code generates the input lat long values \n")
print("Please enter the geo location of the reference point of your fiel
d: \n")
in_lat = 0.0
in_lon = 0.0
while True:
        try:
                in_lat = float(input("Please enter the latitute of poin
t:\n"))
                if(in_lat<0 or in_lat>90):
                        print("Latitude value must be between 0 and 90")
                        continue
                in_lon = float(input("Please enter the longitude of poin
t:\n"))
                if(in_lon<0 or in_lon>180):
                        print("Langitude value must be between 0 and 18
0")
                        continue
                break
        except:
                print("Oops!  That was no valid lat/lon.  Try again...")
```

```
   This code generates the input lat long values

Please enter the geo location of the reference point of your field:

Please enter the latitute of point:
10.0
Please enter the longitude of point:
10.0
```

Create out custom POINT object for input:

In [6]:
```
first_point = POINT(in_lat,in_lon)
```

Create a input.txt file to store all the calculated points.

Store the first point taken from user in file.

In [7]:
```
input_file = open("input.txt","w+")
input_file.write(str(first_point.lat) + "," + str(first_point.lon) + '\n
')
```

Taking user input for number of points to be generated:

In [8]:
```
number_of_points = int(input("Please enter the more number of point to g
enerate:\n"))
```

```
Please enter the more number of point to generate:
5
```

User inputs the distance of each point from reference point

User inputs the each corner point distance in form of (NORTH,EAST) i.e. if next point is 10 m north and 20 m east of initial reference , user enters 10 20 when asked.

In [9]:
```
for i in range(number_of_points):
        print("Enter the next point distance in meter (north,east) from
refrence point:\n")
        move_north = int(input("North distance to point: "))
        move_east = int(input("East distance to point: "))
        new_point = calNewGeoLocationNE(first_point,move_north,move_eas
t)
        input_file.write(str(new_point.lat) + "," + str(new_point.lon) +
'\n')

print("All generated points are stored in input.txt.\n")
```

```
Enter the next point distance in meter (north,east) from refrence point:

North distance to point: -50
East distance to point: 50
Enter the next point distance in meter (north,east) from refrence point:

North distance to point: 0
East distance to point: 100
Enter the next point distance in meter (north,east) from refrence point:

North distance to point: 50
East distance to point: 100
Enter the next point distance in meter (north,east) from refrence point:

North distance to point: 100
East distance to point: 50
Enter the next point distance in meter (north,east) from refrence point:

North distance to point: 50
East distance to point: 0
All generated points are stored in input.txt.
```

Chapter 8. Snapshots of Input and Output

This chapter includes all the inputs to the various file along with their outputs.

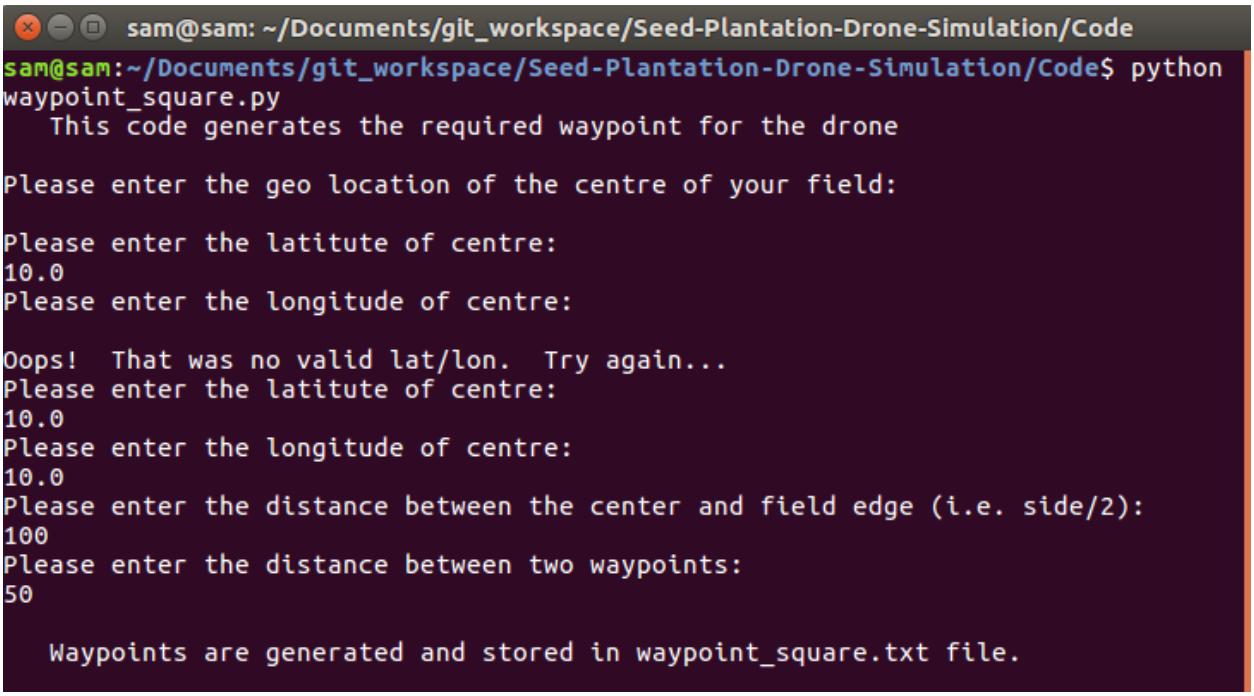
Helps in better understanding of how the code works with different types of inputs.

8.1 Square Field (AUTO mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Square%20AUTO

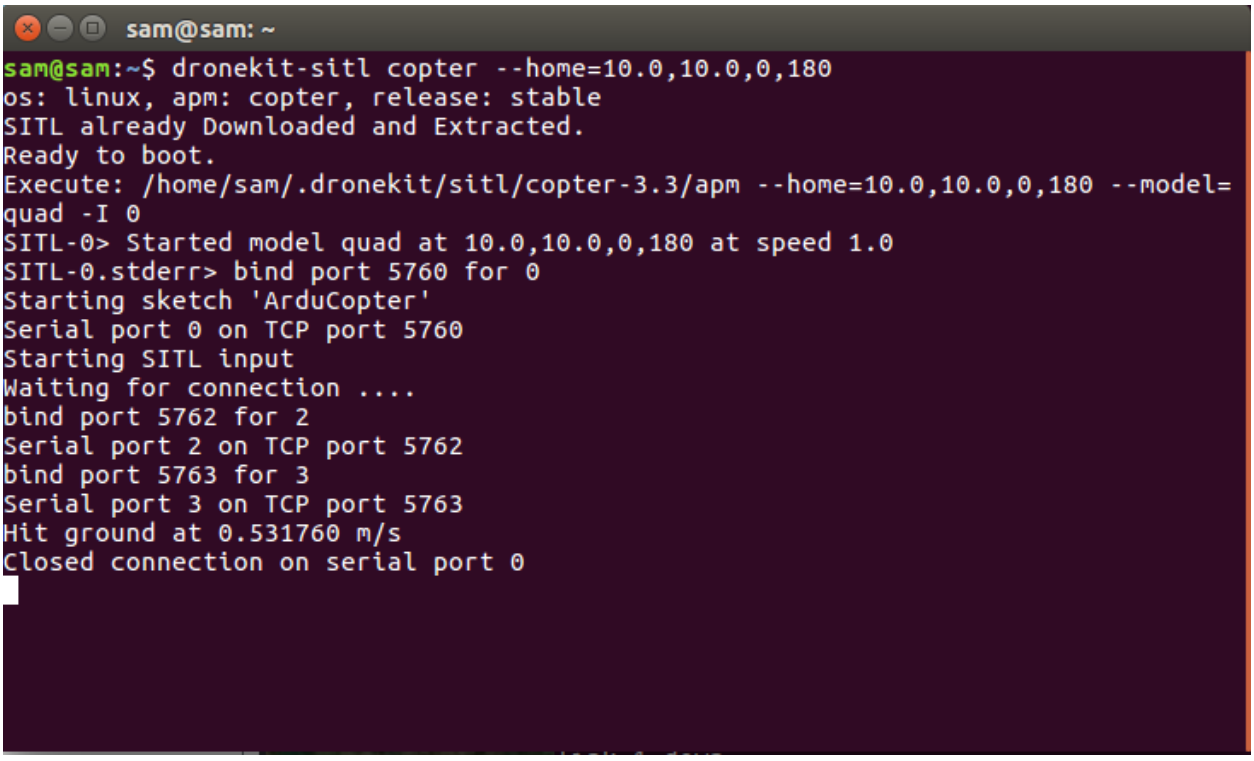
8.1.1 Square waypoint generator terminal

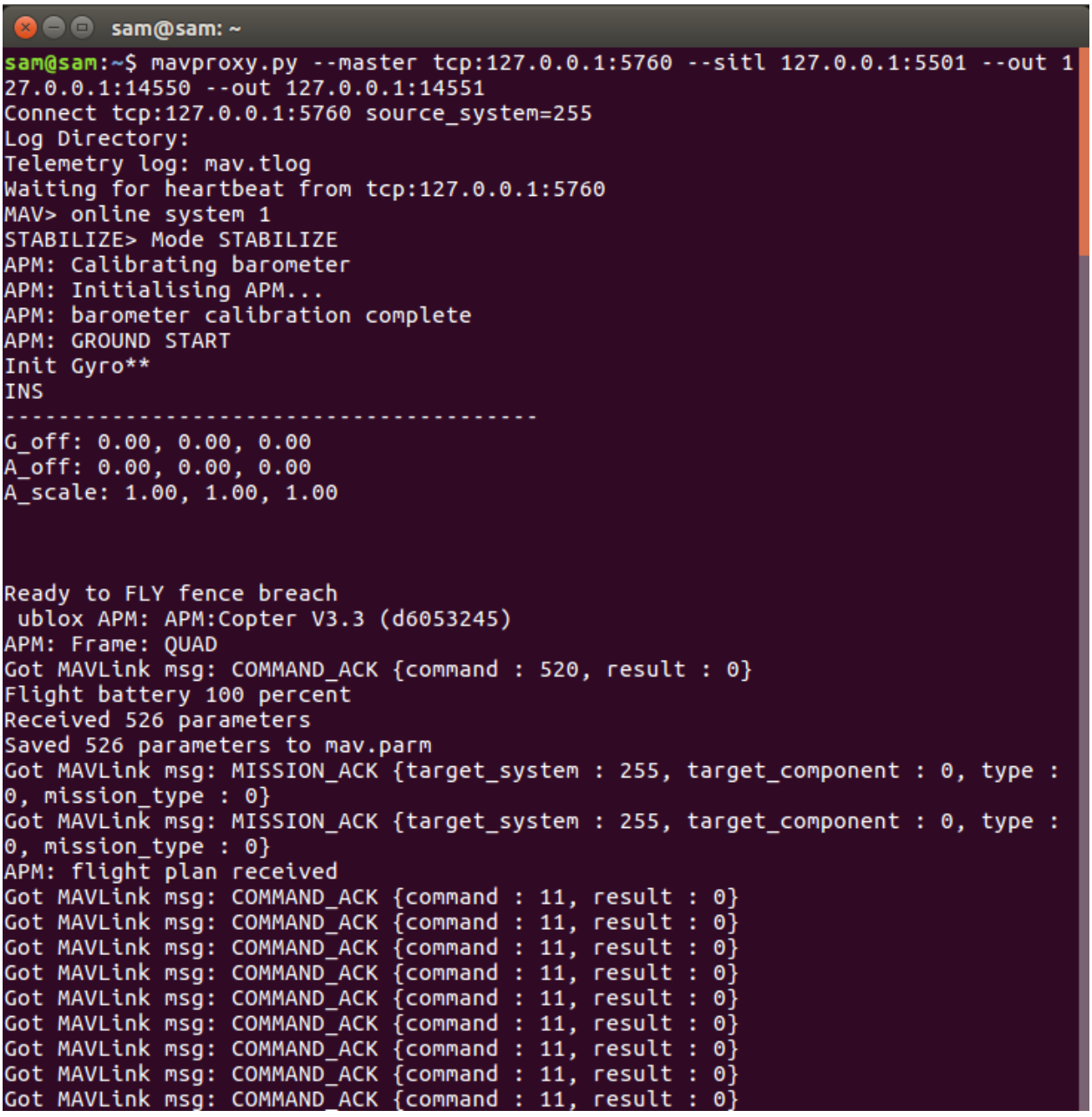


8.1.2 Waypoint square text file

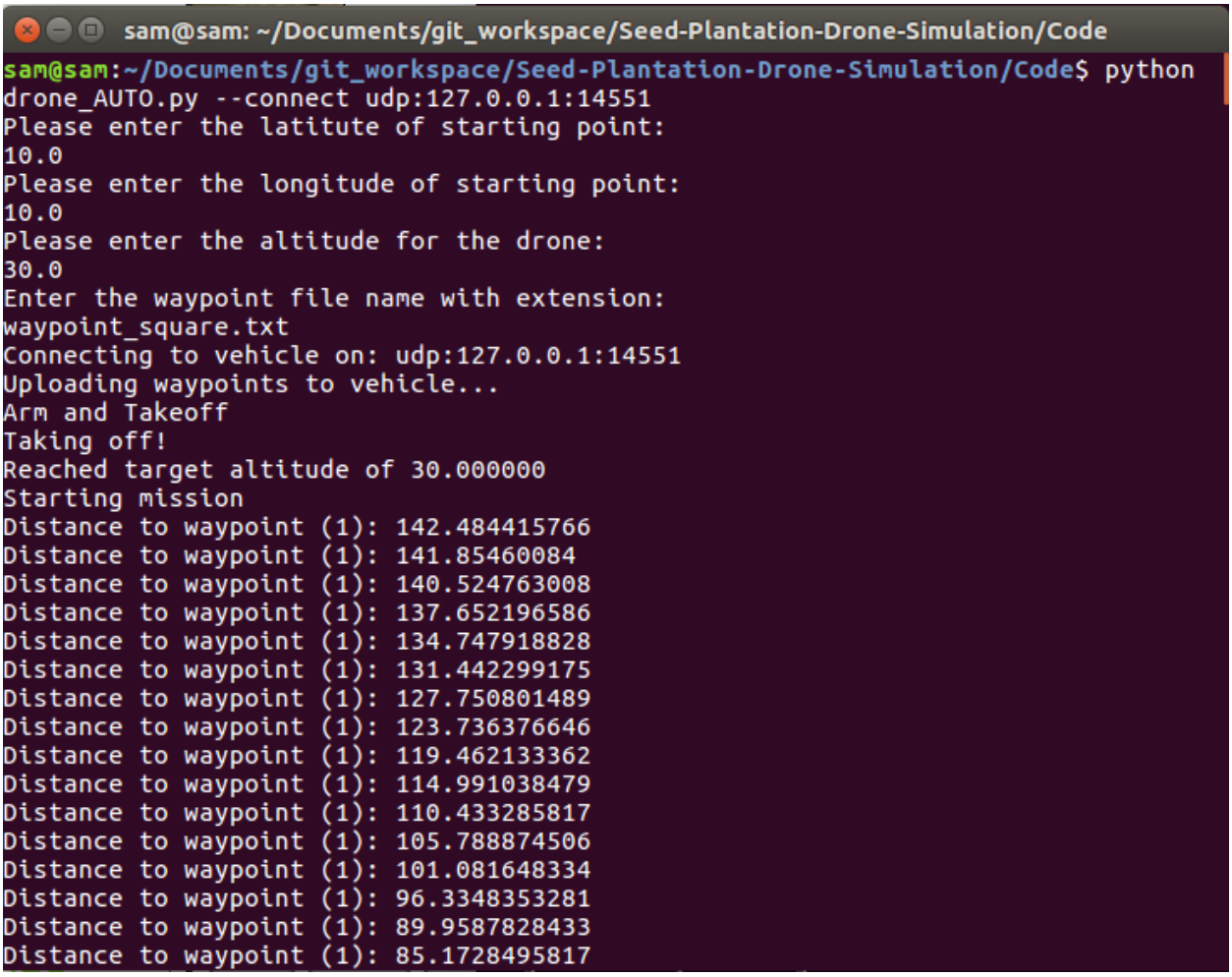
```
9.99910168472,9.99908782675
9.99955084236,9.99908782675
10.0,9.99908782675
10.0004491576,9.99908782675
10.0008983153,9.99908782675
10.0008983153,9.99954391464
10.0004491576,9.99954391464
10.0,9.99954391464
9.99955084236,9.99954391464
9.99910168472,9.99954391464
9.99910168472,10.0
9.99955084236,10.0
10.0,10.0
10.0004491576,10.0
10.0008983153,10.0
10.0008983153,10.0004560879
10.0004491576,10.0004560879
10.0,10.0004560879
9.99955084236,10.0004560879
9.99910168472,10.0004560879
9.99910168472,10.0009121732
9.99955084236,10.0009121732
10.0,10.0009121732
10.0004491576,10.0009121732
10.0008983153,10.0009121732
```

8.1.3 Terminal outputs

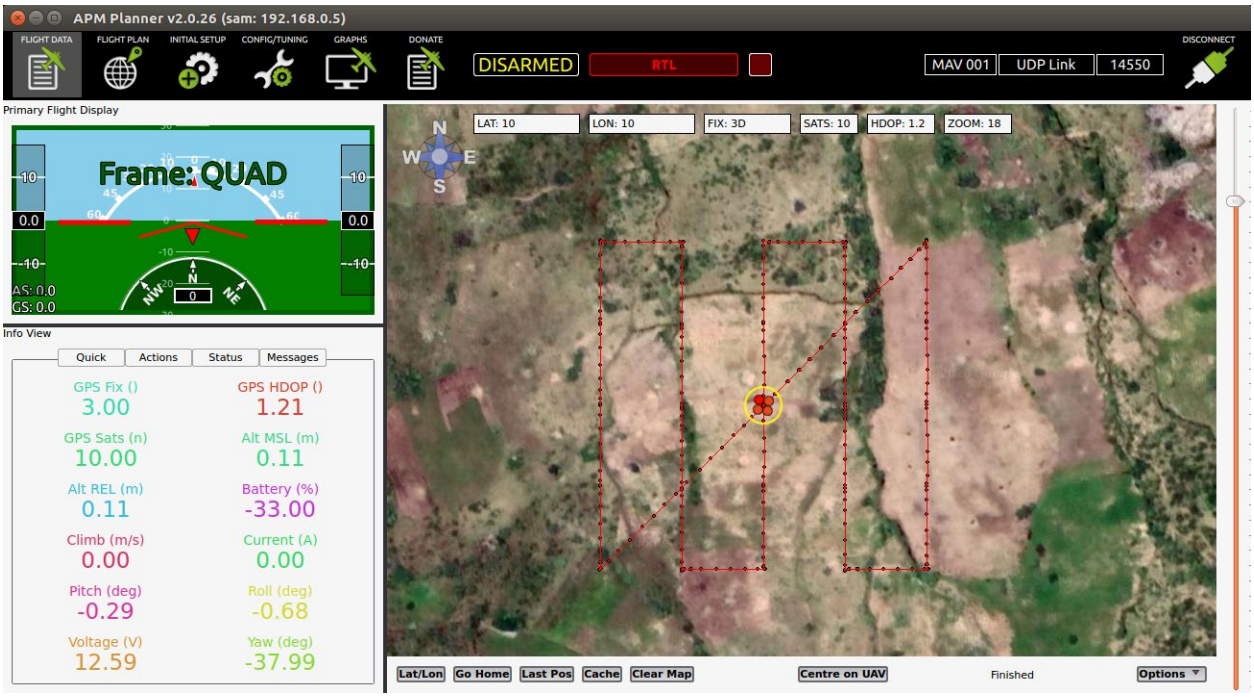



sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY fence breach
ublox APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Flight battery 100 percent
Received 526 parameters
Saved 526 parameters to mav.parm
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
APM: flight plan received
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}


sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_AUTO.py --connect udp:127.0.0.1:14551
Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
30.0
Enter the waypoint file name with extension:
waypoint_square.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Uploading waypoints to vehicle...
Arm and Takeoff
Taking off!
Reached target altitude of 30.000000
Starting mission
Distance to waypoint (1): 142.484415766
Distance to waypoint (1): 141.85460084
Distance to waypoint (1): 140.524763008
Distance to waypoint (1): 137.652196586
Distance to waypoint (1): 134.747918828
Distance to waypoint (1): 131.442299175
Distance to waypoint (1): 127.750801489
Distance to waypoint (1): 123.736376646
Distance to waypoint (1): 119.462133362
Distance to waypoint (1): 114.991038479
Distance to waypoint (1): 110.433285817
Distance to waypoint (1): 105.788874506
Distance to waypoint (1): 101.081648334
Distance to waypoint (1): 96.3348353281
Distance to waypoint (1): 89.9587828433
Distance to waypoint (1): 85.1728495817

8.1.4 Square APM Planner2 output



8.1.5 Square log file

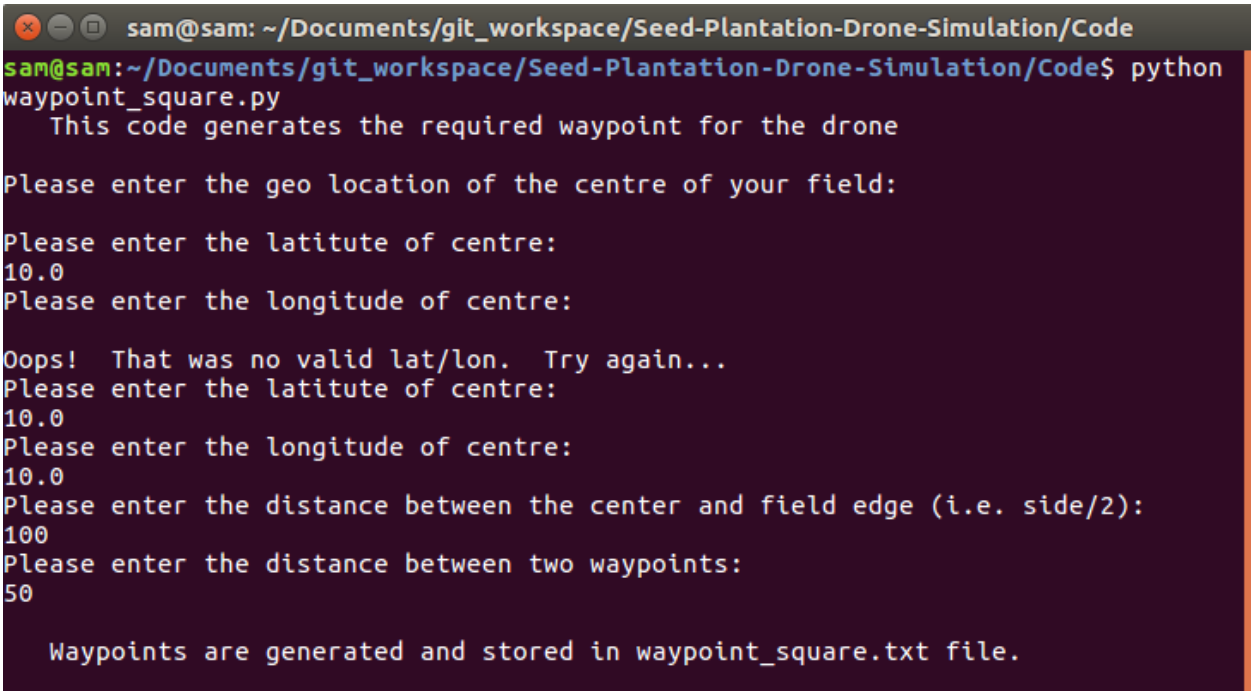
```
DEBUG: drone_AUTO.py: <module>: 141:                USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:                USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:                USER entered altitude value: 30.0
INFO: drone_AUTO.py: <module>: 187:                 Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: statustext_listener: 1065:                APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: statustext_listener: 1065:                Frame: QUAD
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
INFO: drone_AUTO.py: print_vehicle_attributes: 98:                Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_AUTO.py: print_vehicle_attributes: 99:                Autopilot capabilities (supports ftp): False
INFO: drone_AUTO.py: print_vehicle_attributes: 100:                Global Location: LocationGlobal:lat=10.0,lon=10.0,alt=None
INFO: drone_AUTO.py: print_vehicle_attributes: 101:                Global Location (relative altitude): LocationGlobalRelative:lat=10.
INFO: drone_AUTO.py: print_vehicle_attributes: 102:                Local Location: LocationLocal:north=None,east=None,down=None
INFO: drone_AUTO.py: print_vehicle_attributes: 103:                Attitude: Attitude:pitch=0.00139206054155,yaw=-3.13637590408,roll=0
INFO: drone_AUTO.py: print_vehicle_attributes: 104:                Velocity: [0.02, -0.03, 0.0]
INFO: drone_AUTO.py: print_vehicle_attributes: 105:                GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_AUTO.py: print_vehicle_attributes: 106:                Groundspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 107:                Airspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 108:                Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
INFO: drone_AUTO.py: print_vehicle_attributes: 109:                Battery: Battery:voltage=12.587,current=0.0,level=100
INFO: drone_AUTO.py: print_vehicle_attributes: 110:                EKF OK?: True
INFO: drone_AUTO.py: print_vehicle_attributes: 111:                Last Heartbeat: 0.605894084
```

8.2 Square Field (GUIDED mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Square%20GUIDED

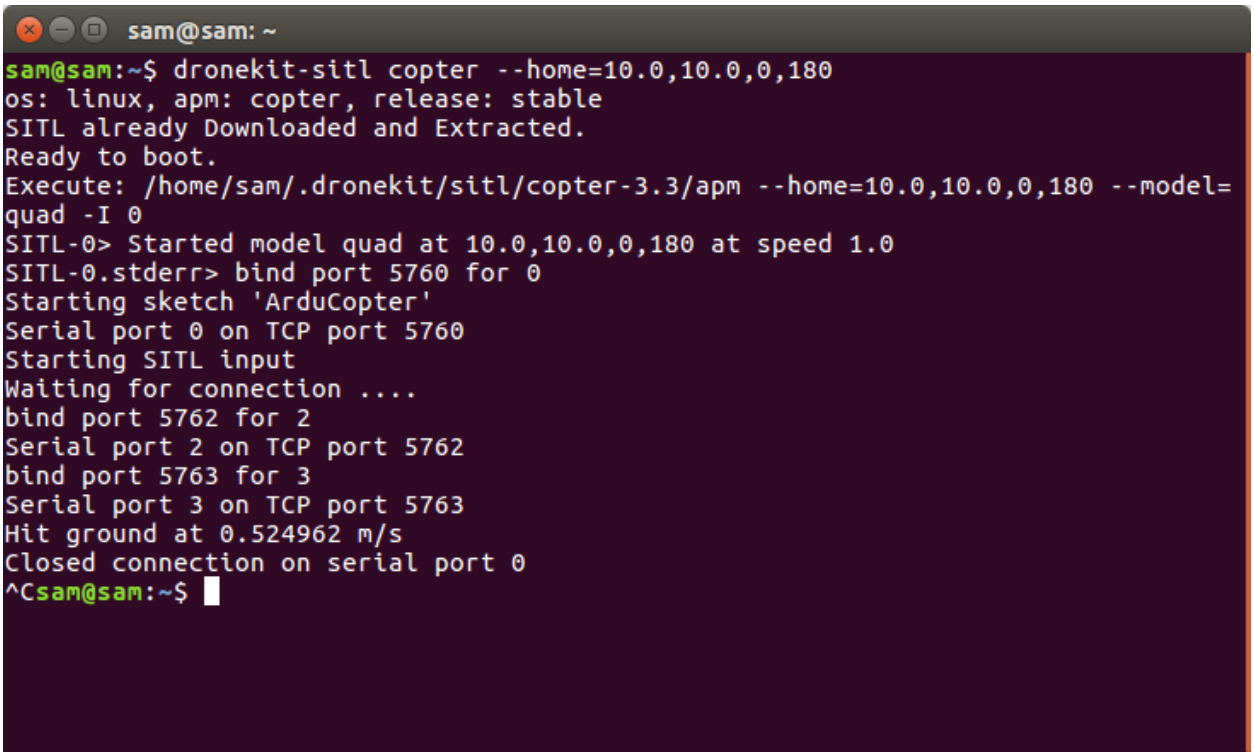
8.2.1 Square waypoint generator terminal

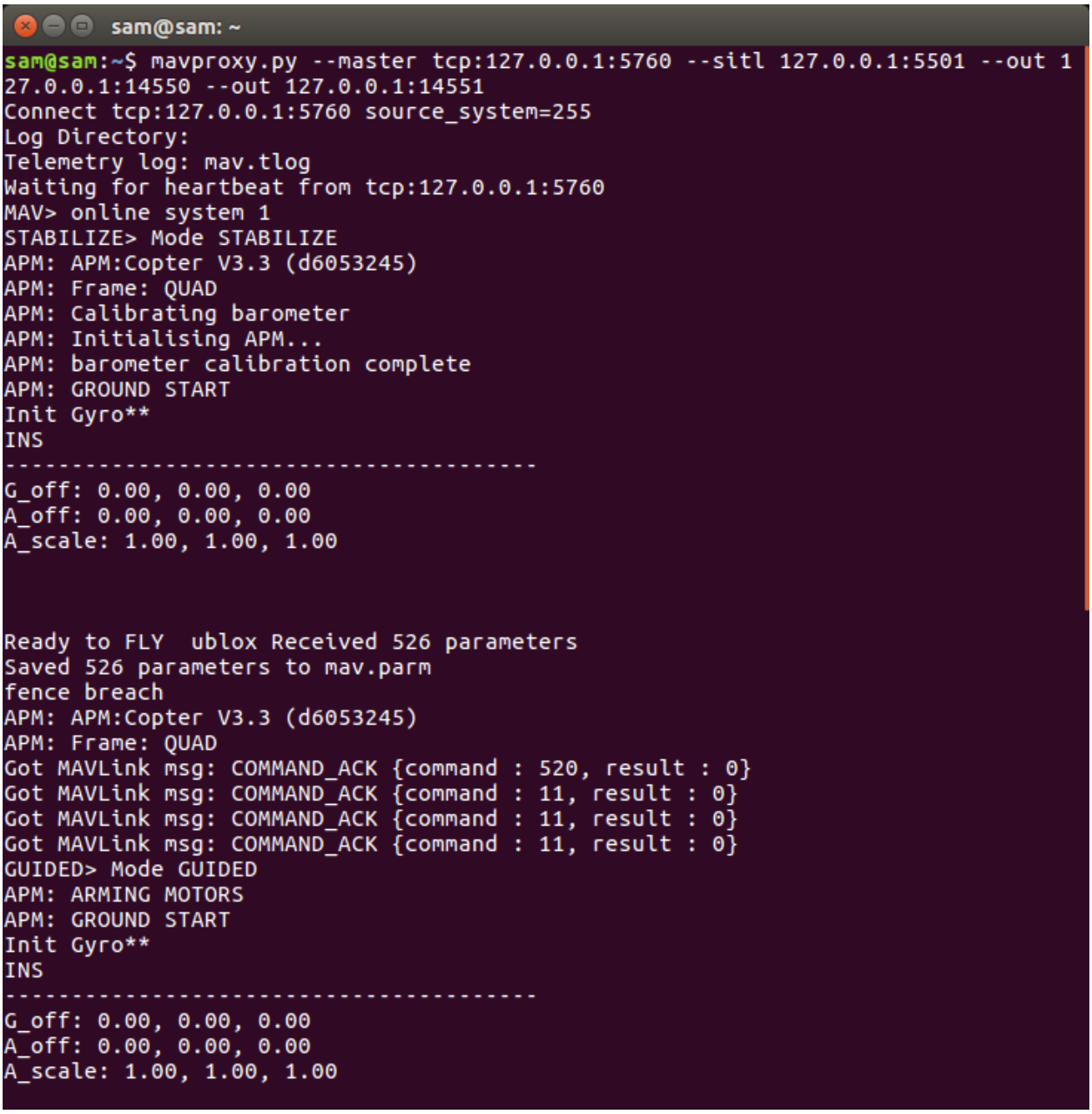


8.2.2 Waypoint square text file

```
9.99910168472,9.99908782675
9.99955084236,9.99908782675
10.0,9.99908782675
10.0004491576,9.99908782675
10.0008983153,9.99908782675
10.0008983153,9.99954391464
10.0004491576,9.99954391464
10.0,9.99954391464
9.99955084236,9.99954391464
9.99910168472,9.99954391464
9.99910168472,10.0
9.99955084236,10.0
10.0,10.0
10.0004491576,10.0
10.0008983153,10.0
10.0008983153,10.0004560879
10.0004491576,10.0004560879
10.0,10.0004560879
9.99955084236,10.0004560879
9.99910168472,10.0004560879
9.99910168472,10.0009121732
9.99955084236,10.0009121732
10.0,10.0009121732
10.0004491576,10.0009121732
10.0008983153,10.0009121732
```

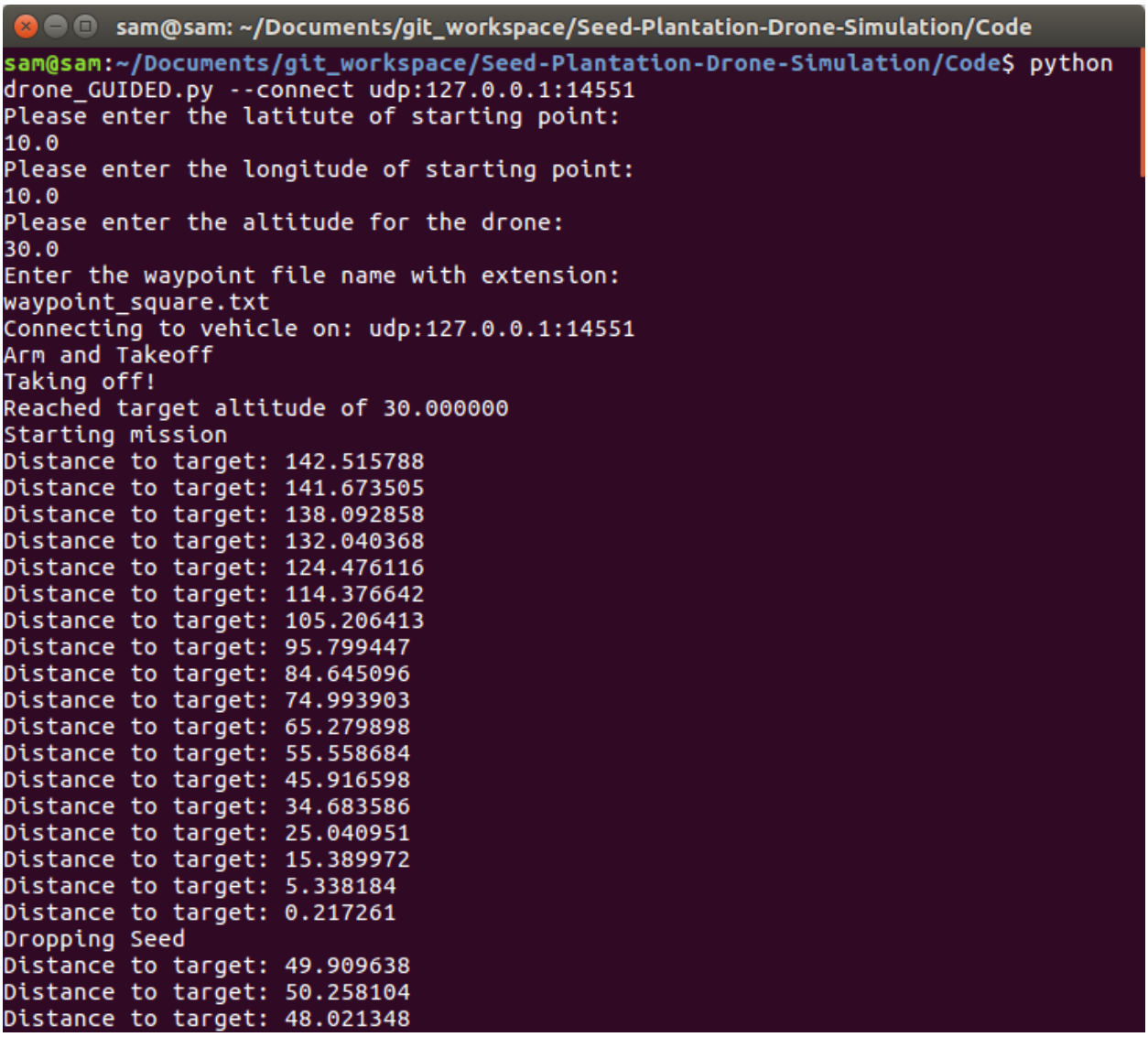
8.2.3 Terminal outputs



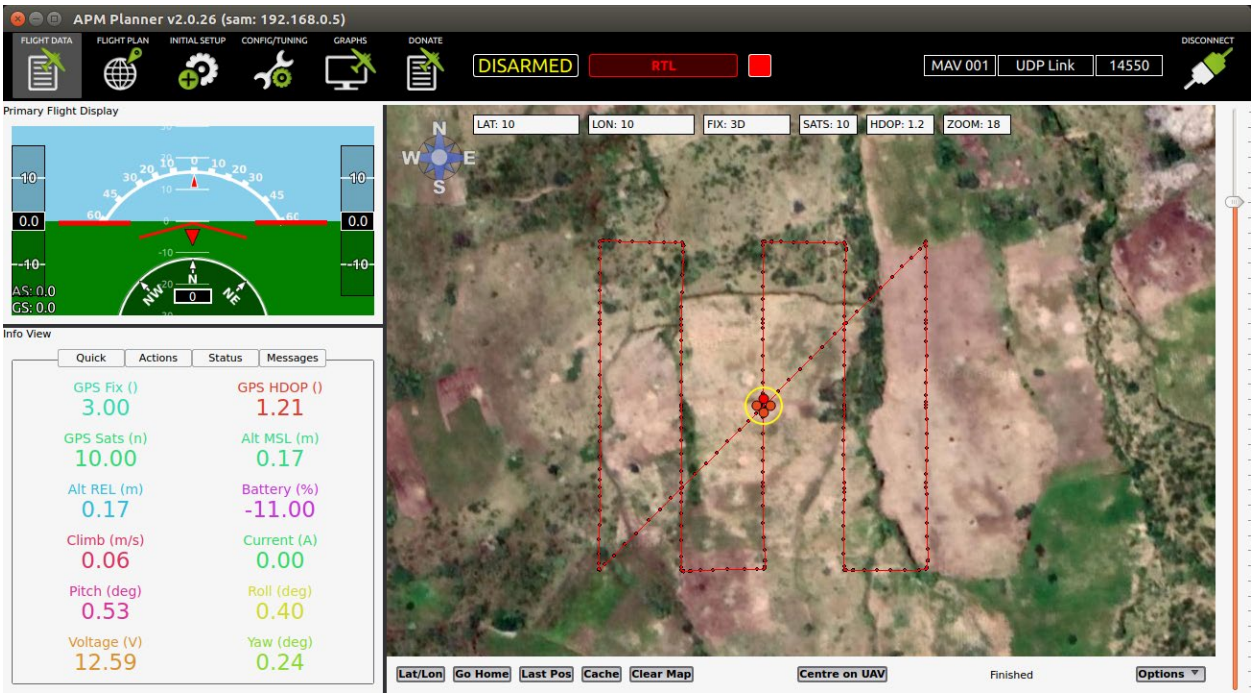

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY ublox Received 526 parameters
Saved 526 parameters to mav.parm
fence breach
APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
GUIDED> Mode GUIDED
APM: ARMING MOTORS
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00


sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_GUIDED.py --connect udp:127.0.0.1:14551
Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
30.0
Enter the waypoint file name with extension:
waypoint_square.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Arm and Takeoff
Taking off!
Reached target altitude of 30.000000
Starting mission
Distance to target: 142.515788
Distance to target: 141.673505
Distance to target: 138.092858
Distance to target: 132.040368
Distance to target: 124.476116
Distance to target: 114.376642
Distance to target: 105.206413
Distance to target: 95.799447
Distance to target: 84.645096
Distance to target: 74.993903
Distance to target: 65.279898
Distance to target: 55.558684
Distance to target: 45.916598
Distance to target: 34.683586
Distance to target: 25.040951
Distance to target: 15.389972
Distance to target: 5.338184
Distance to target: 0.217261
Dropping Seed
Distance to target: 49.909638
Distance to target: 50.258104
Distance to target: 48.021348

8.2.4 Square APM Planner2 output



8.2.5 Square log file

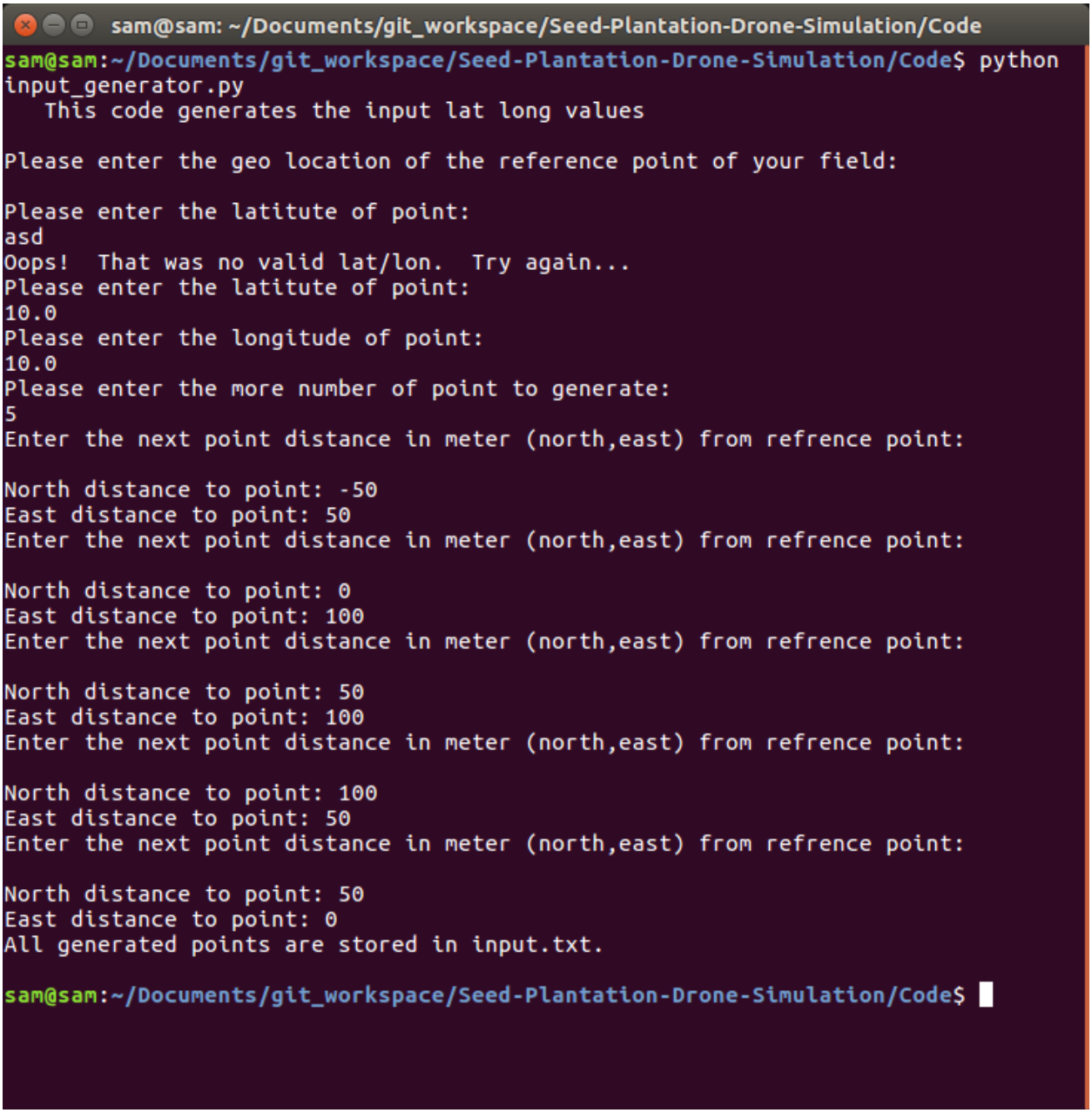
```
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 216:          USER entered latitute value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 221:          USER entered longitude value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 226:          USER entered altitude value: 30.0
INFO: drone_GUIDED.py: drone_GUIDED: <module>: 262:           Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: __init__: statustext_listener: 1065:           APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: __init__: statustext_listener: 1065:           Frame: QUAD
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:           Still waiting for data from vehicle: parameters
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 160:          Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 161:          Autopilot capabilities (supports ftp): False
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 162:          Global Location:LocationGlobal:lat=10.0,lon=10.0,al
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 163:          Global Location (relative altitude): LocationGlobal
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 164:          Local Location: LocationLocal:north=None,east=None,
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 165:          Attitude: Attitude:pitch=0.000782963936217,yaw=-3.1
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 166:          Velocity: [0.01, -0.01, 0.0]
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 167:          GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 168:          Groundspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 169:          Airspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 170:          Gimbal status: Gimbal: pitch=None, roll=None, yaw=N
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 171:          Battery: Battery:voltage=12.587,current=0.0,level=1
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 172:          EKF OK?: True
INFO: drone_GUIDED.py: drone_GUIDED: print_vechicle_attributes: 173:          Last Heartbeat: 0.657818574
```

8.3 Hexagon Field (AUTO mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Hexagon%20AUTO

8.3.1 Input Generator file

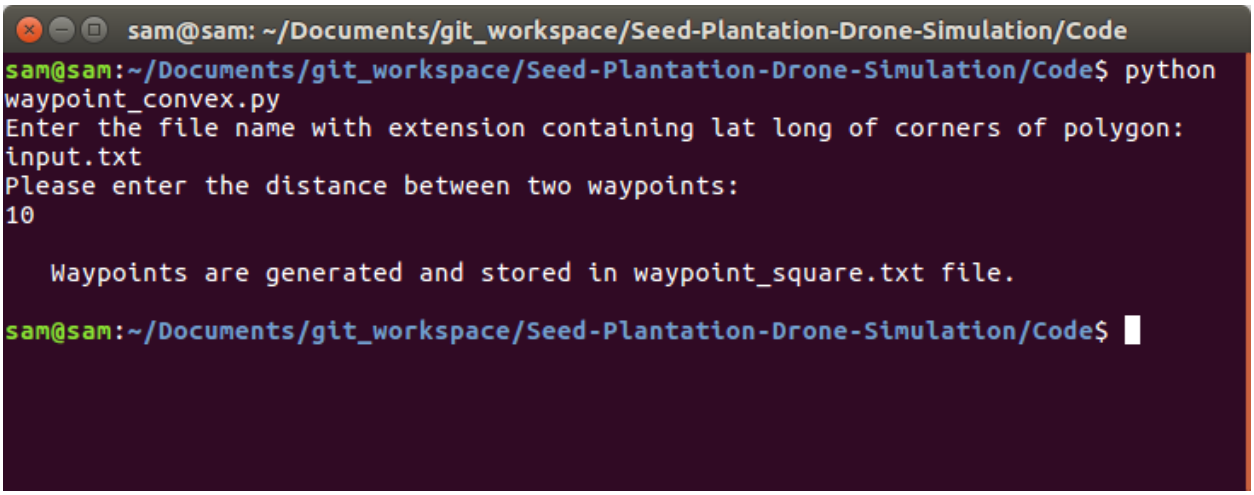


8.3.2 Input Text file

```
10.0,10.0
9.99955084236,10.0004560866
10.0,10.0009121732
10.0004491576,10.0009121732
10.0008983153,10.0004560866
10.0004491576,10.0
```

8.3.3 Hexagon Google Maps

8.3.4 Hexagon waypoint generator terminal

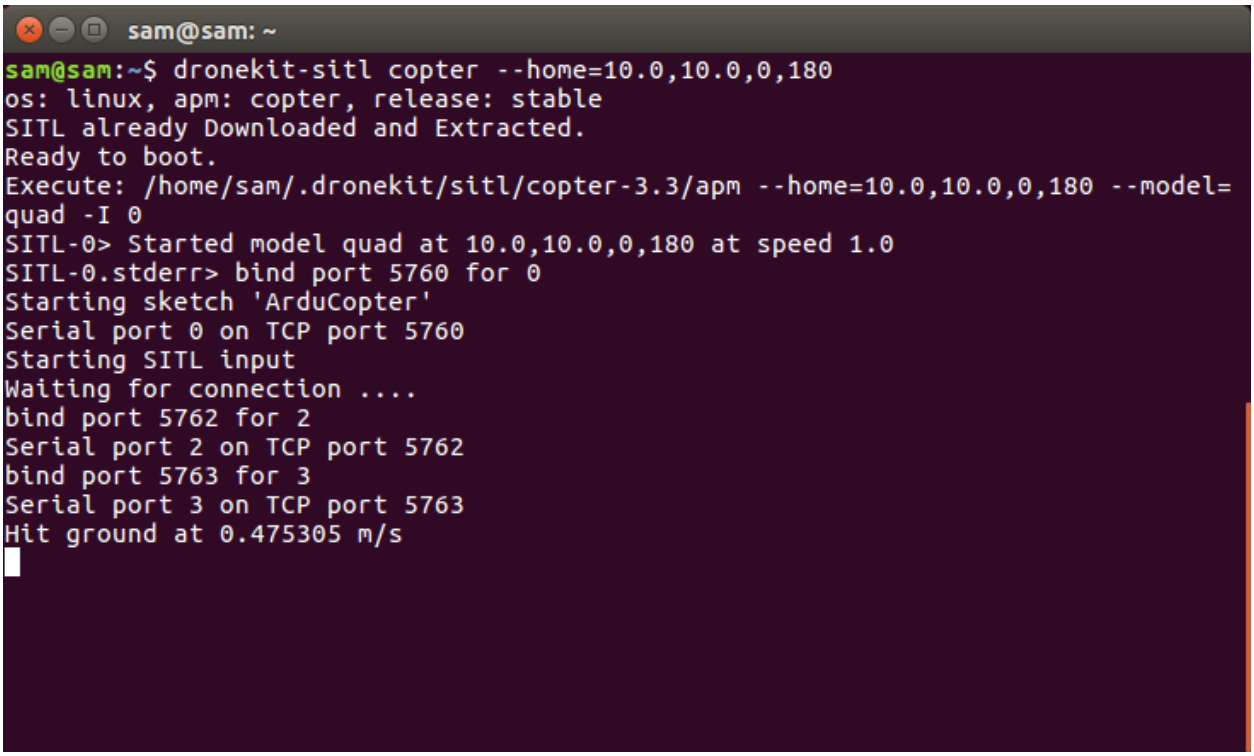


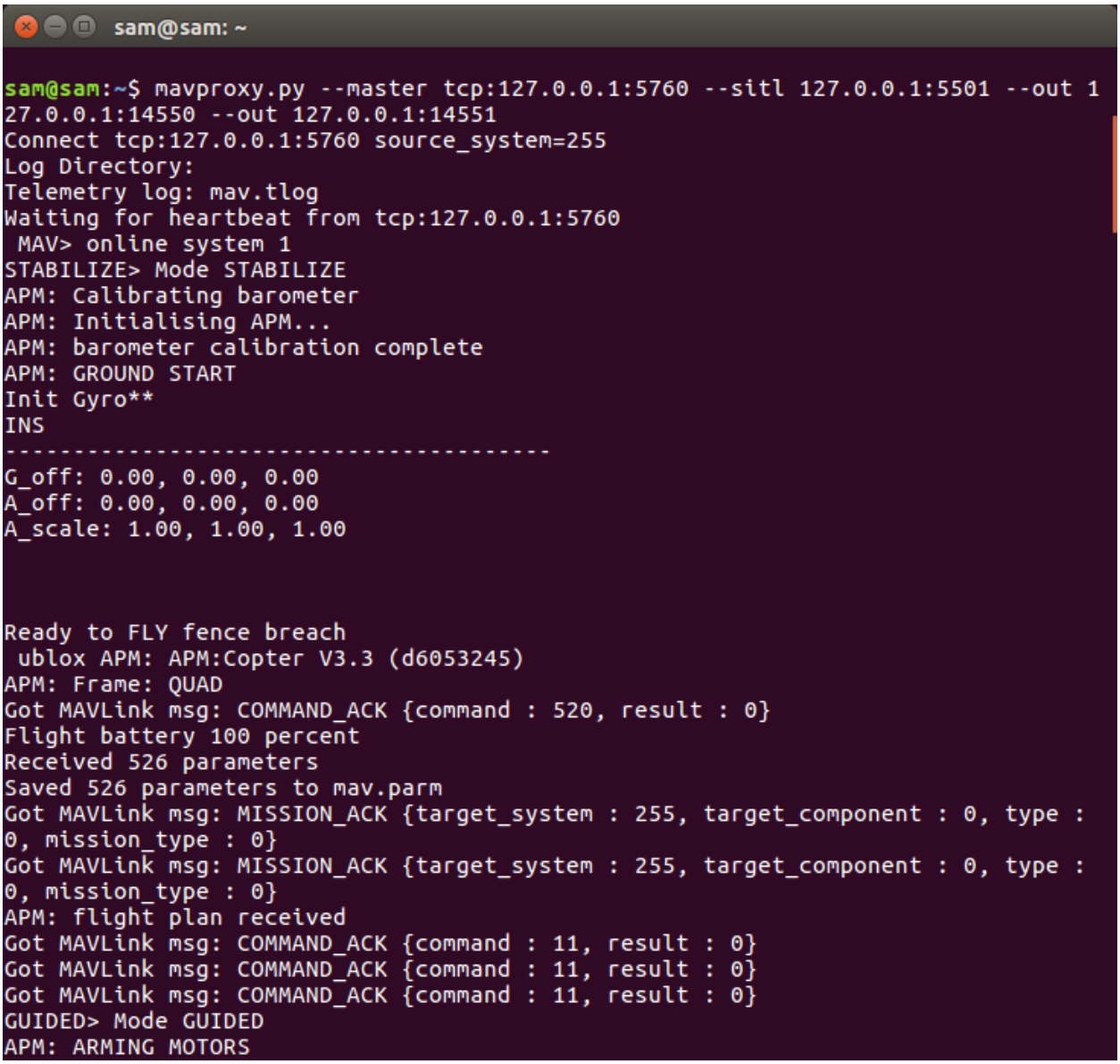
8.3.5 Waypoint hexagon text file

99 lines (99 sloc) | 2.57 KB

```
1   10.0,10.0
2   10.0000898315,10.0
3   10.0001796631,10.0
4   10.0002694946,10.0
5   10.0003593261,10.0
6   10.0005389892,10.0000912176
7   10.0004491576,10.0000912176
8   10.0003593261,10.0000912176
9   10.0002694946,10.0000912176
10  10.0001796631,10.0000912176
11  10.0000898315,10.0000912176
12  10.0,10.0000912176
13  9.99991016847,10.0000912176
14  9.99982033695,10.0001824348
15  9.99991016847,10.0001824348
16  10.0,10.0001824348
17  10.0000898315,10.0001824348
18  10.0001796631,10.0001824348
19  10.0002694946,10.0001824348
20  10.0003593261,10.0001824348
```

8.3.6 Terminal outputs



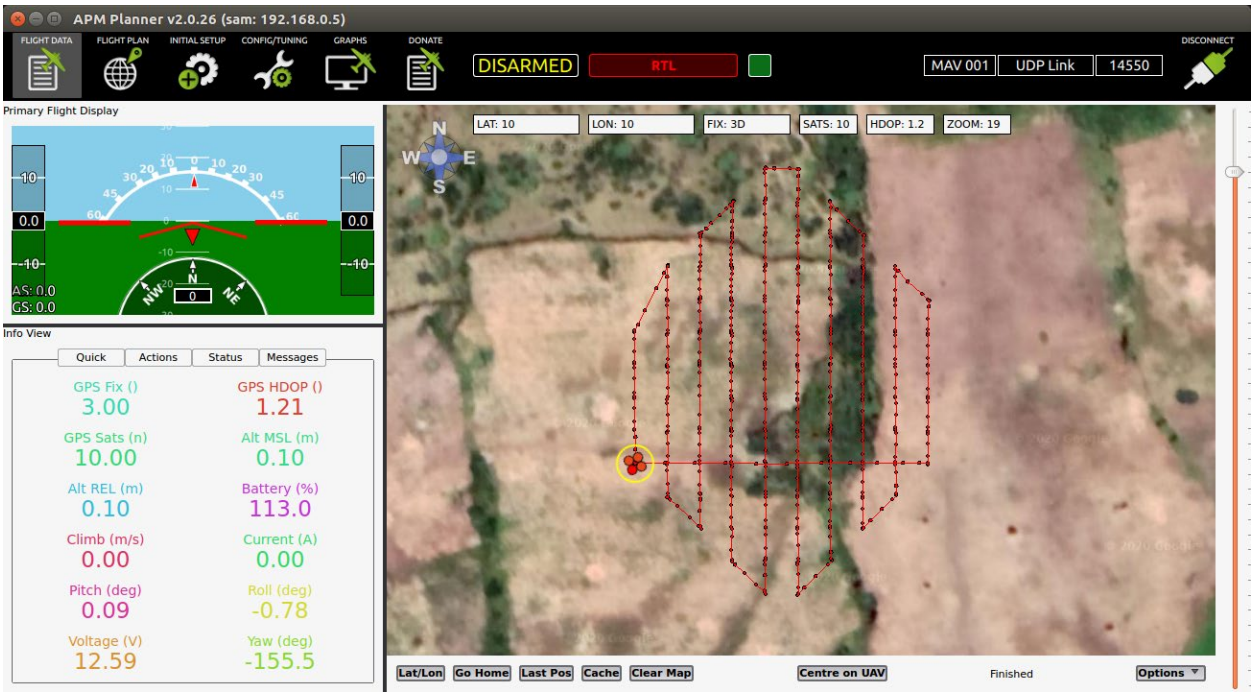

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY fence breach
ublox APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Flight battery 100 percent
Received 526 parameters
Saved 526 parameters to mav.parm
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
APM: flight plan received
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
GUIDED> Mode GUIDED
APM: ARMING MOTORS

```
sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ clear

sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_AUTO.py --connect udp:127.0.0.1:14551
Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
15.0
Enter the waypoint file name with extension:
waypoint_convex.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Uploading waypoints to vehicle...
Arm and Takeoff
Taking off!
Reached target altitude of 15.000000
Starting mission
Distance to waypoint (1): 0.0556597500572
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (1): 0.0333958499948
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (1): 0.0222639000624
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (1): 0.0333958499948
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (1): 0.01574295457
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (1): 0.0248917969477
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253:                         Dropping Seed
Distance to waypoint (2): 9.96662668138
Distance to waypoint (2): 9.24356802784
Distance to waypoint (2): 7.86521606743
Distance to waypoint (2): 5.84226137491
Distance to waypoint (2): 3.06176628747
Distance to waypoint (2): 1.49522930351
Dropping Seed
```

8.3.7 Hexagon APM Planner2 output



8.3.8 Hexagon log file

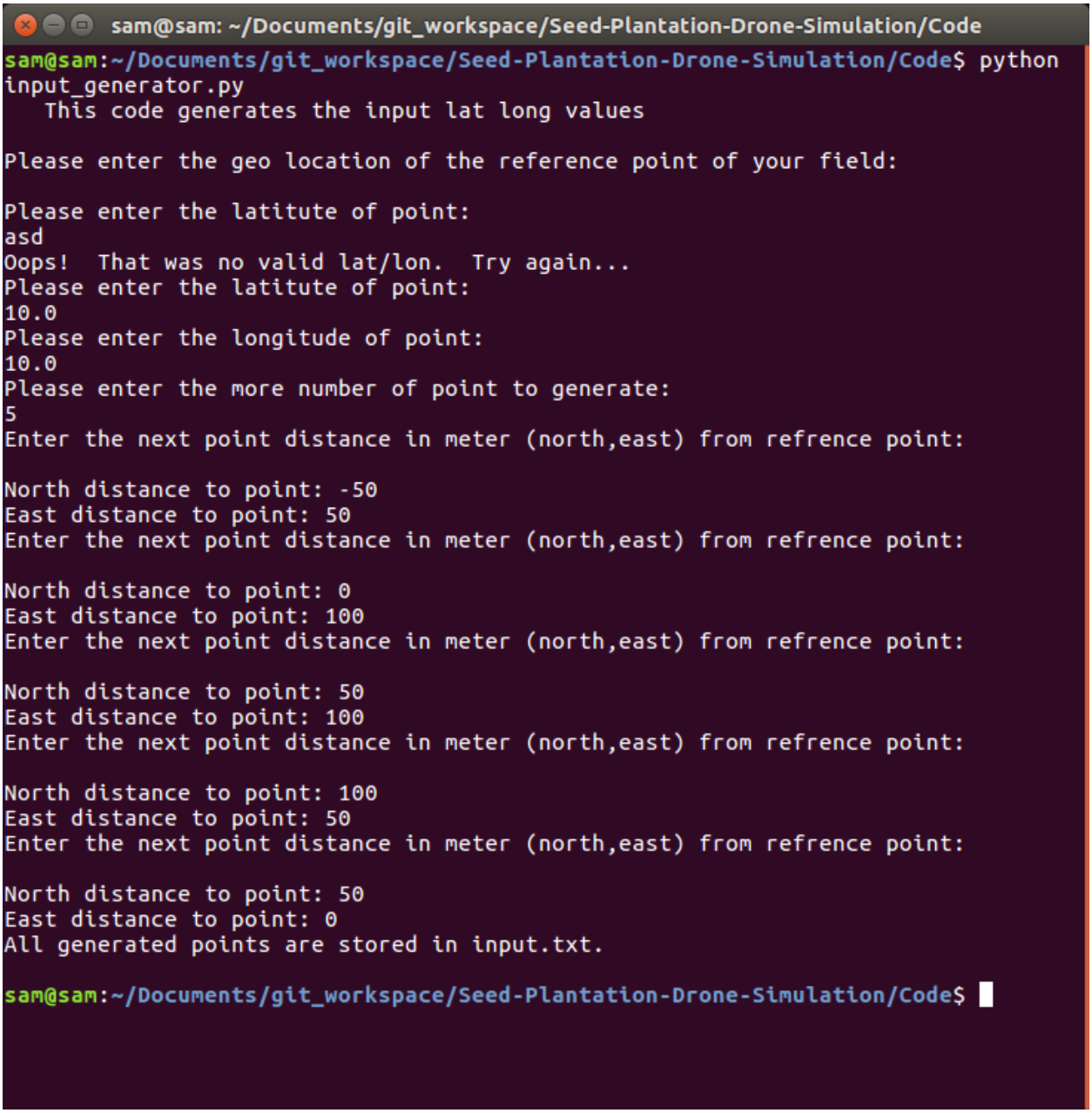
```
DEBUG: drone_AUTO.py: <module>: 141:             USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:             USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:             USER entered altitude value: 15.0
INFO: drone_AUTO.py: <module>: 187:              Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: statustext_listener: 1065:                APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: statustext_listener: 1065:                Frame: QUAD
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:                Still waiting for data from vehicle: parameters
INFO: drone_AUTO.py: print_vehicle_attributes: 98:               Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_AUTO.py: print_vehicle_attributes: 99:               Autopilot capabilities (supports ftp): False
INFO: drone_AUTO.py: print_vehicle_attributes: 100:              Global Location: LocationGlobal:lat=10.0,lon=10.0,alt=None
INFO: drone_AUTO.py: print_vehicle_attributes: 101:              Global Location (relative altitude): LocationGlobalRelative:lat=10.
INFO: drone_AUTO.py: print_vehicle_attributes: 102:              Local Location: LocationLocal:north=None,east=None,down=None
INFO: drone_AUTO.py: print_vehicle_attributes: 103:              Attitude: Attitude:pitch=0.00131169112865,yaw=-3.13779854774,roll=0
INFO: drone_AUTO.py: print_vehicle_attributes: 104:              Velocity: [0.01, -0.02, 0.0]
INFO: drone_AUTO.py: print_vehicle_attributes: 105:              GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_AUTO.py: print_vehicle_attributes: 106:              Groundspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 107:              Airspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 108:              Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
INFO: drone_AUTO.py: print_vehicle_attributes: 109:              Battery: Battery:voltage=12.587,current=0.0,level=100
INFO: drone_AUTO.py: print_vehicle_attributes: 110:              EKF OK?: True
INFO: drone_AUTO.py: print_vehicle_attributes: 111:              Last Heartbeat: 0.658461211
```

8.4 Hexagon Field (GUIDED mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Hexagon%20GUIDED

8.4.1 Input Generator file

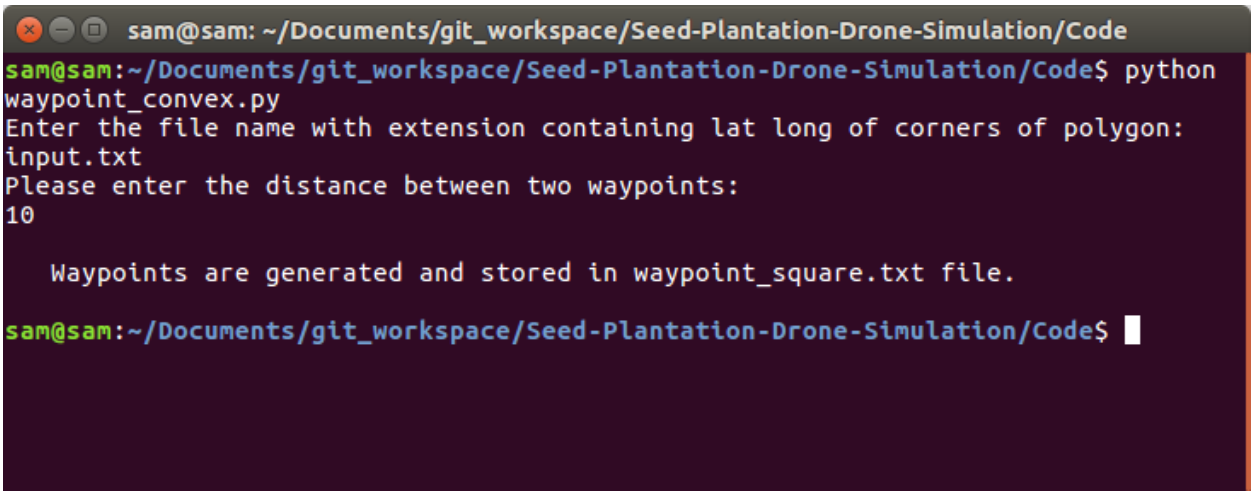


8.4.2 Input Text file

```
10.0,10.0
9.99955084236,10.0004560866
10.0,10.0009121732
10.0004491576,10.0009121732
10.0008983153,10.0004560866
10.0004491576,10.0
```

8.4.3 Hexagon Google Maps

8.4.4 Hexagon waypoint generator terminal

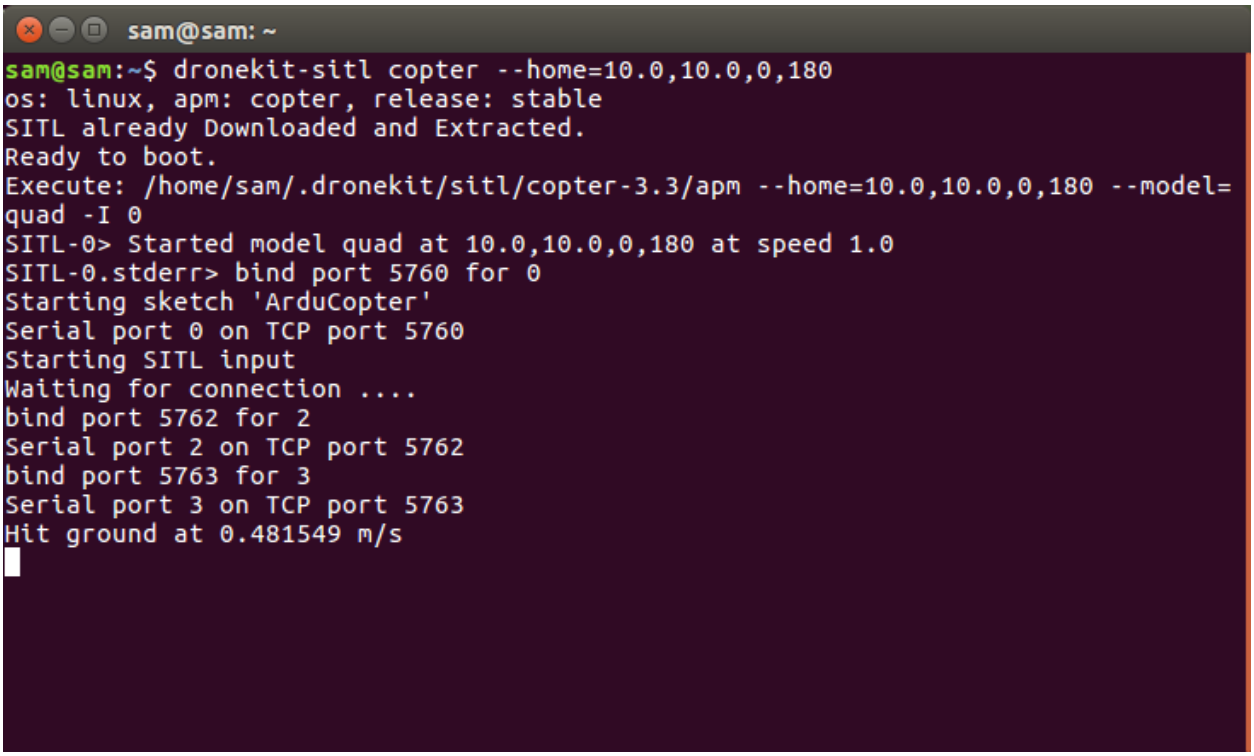


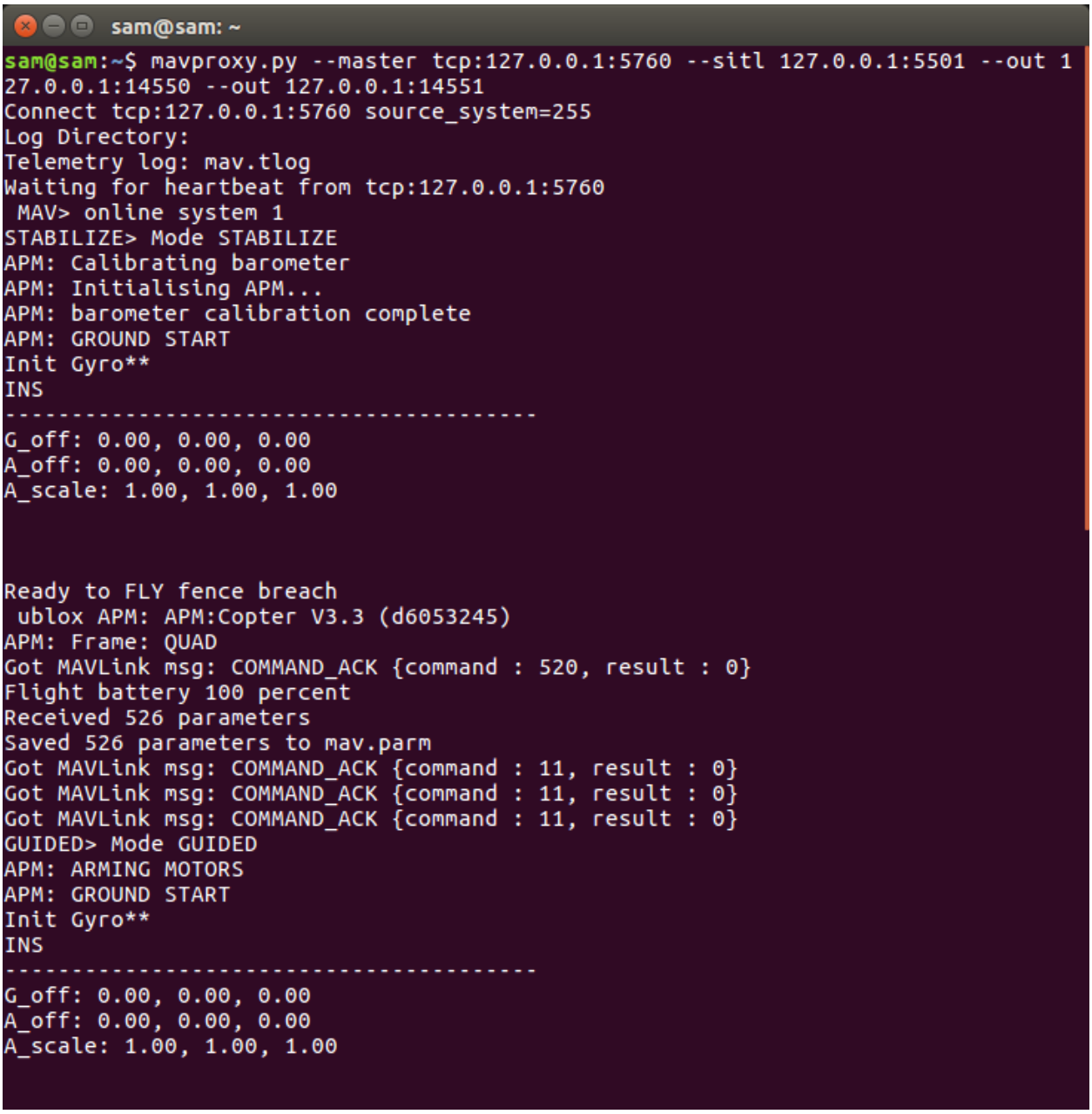
8.4.5 Waypoint hexagon text file

99 lines (99 sloc) | 2.57 KB

```
1   10.0,10.0
2   10.0000898315,10.0
3   10.0001796631,10.0
4   10.0002694946,10.0
5   10.0003593261,10.0
6   10.0005389892,10.0000912176
7   10.0004491576,10.0000912176
8   10.0003593261,10.0000912176
9   10.0002694946,10.0000912176
10  10.0001796631,10.0000912176
11  10.0000898315,10.0000912176
12  10.0,10.0000912176
13  9.99991016847,10.0000912176
14  9.99982033695,10.0001824348
15  9.99991016847,10.0001824348
16  10.0,10.0001824348
17  10.0000898315,10.0001824348
18  10.0001796631,10.0001824348
19  10.0002694946,10.0001824348
20  10.0003593261,10.0001824348
```

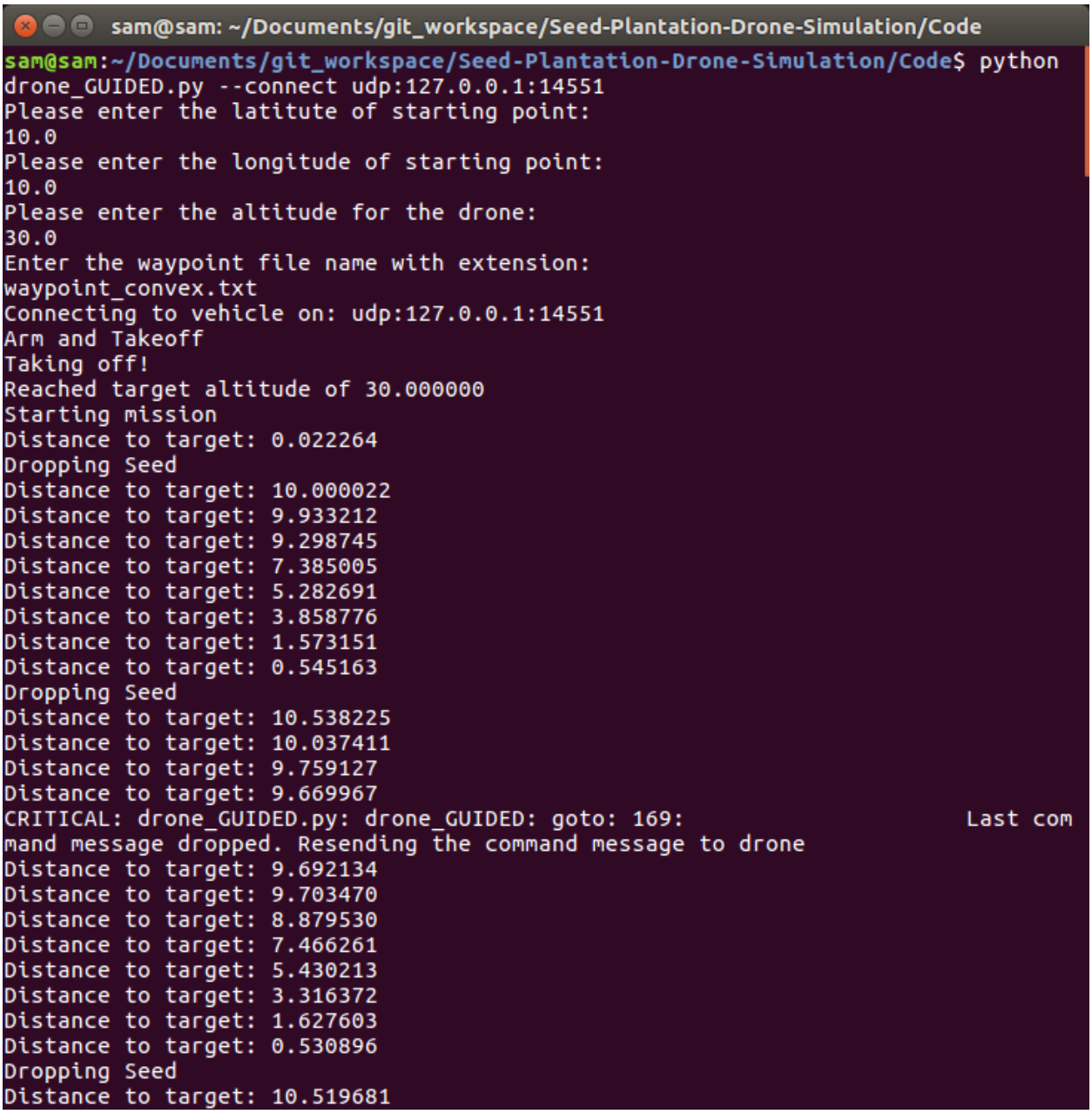
8.4.6 Terminal outputs



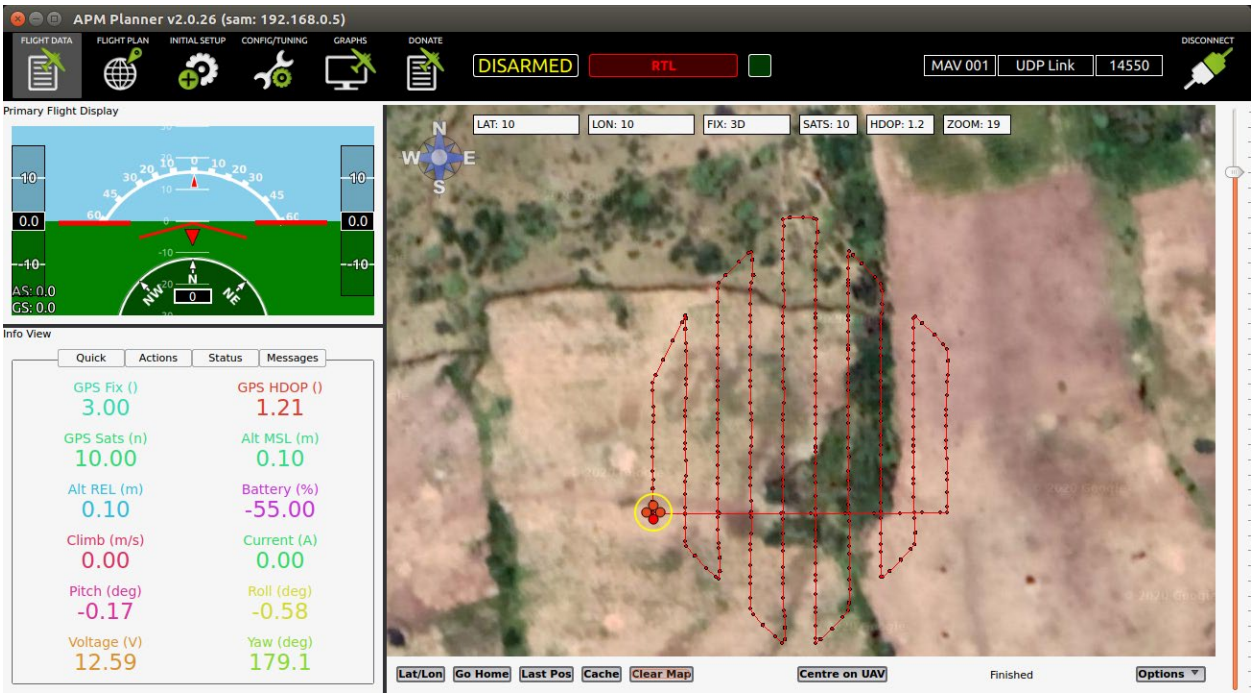

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY fence breach
ublox APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Flight battery 100 percent
Received 526 parameters
Saved 526 parameters to mav.parm
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
GUIDED> Mode GUIDED
APM: ARMING MOTORS
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00


sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_GUIDED.py --connect udp:127.0.0.1:14551
Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
30.0
Enter the waypoint file name with extension:
waypoint_convex.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Arm and Takeoff
Taking off!
Reached target altitude of 30.000000
Starting mission
Distance to target: 0.022264
Dropping Seed
Distance to target: 10.000022
Distance to target: 9.933212
Distance to target: 9.298745
Distance to target: 7.385005
Distance to target: 5.282691
Distance to target: 3.858776
Distance to target: 1.573151
Distance to target: 0.545163
Dropping Seed
Distance to target: 10.538225
Distance to target: 10.037411
Distance to target: 9.759127
Distance to target: 9.669967
CRITICAL: drone_GUIDED.py: drone_GUIDED: goto: 169: Last command message dropped. Resending the command message to drone
Distance to target: 9.692134
Distance to target: 9.703470
Distance to target: 8.879530
Distance to target: 7.466261
Distance to target: 5.430213
Distance to target: 3.316372
Distance to target: 1.627603
Distance to target: 0.530896
Dropping Seed
Distance to target: 10.519681

8.4.7 Hexagon APM Planner2 output



8.4.8 Hexagon log file

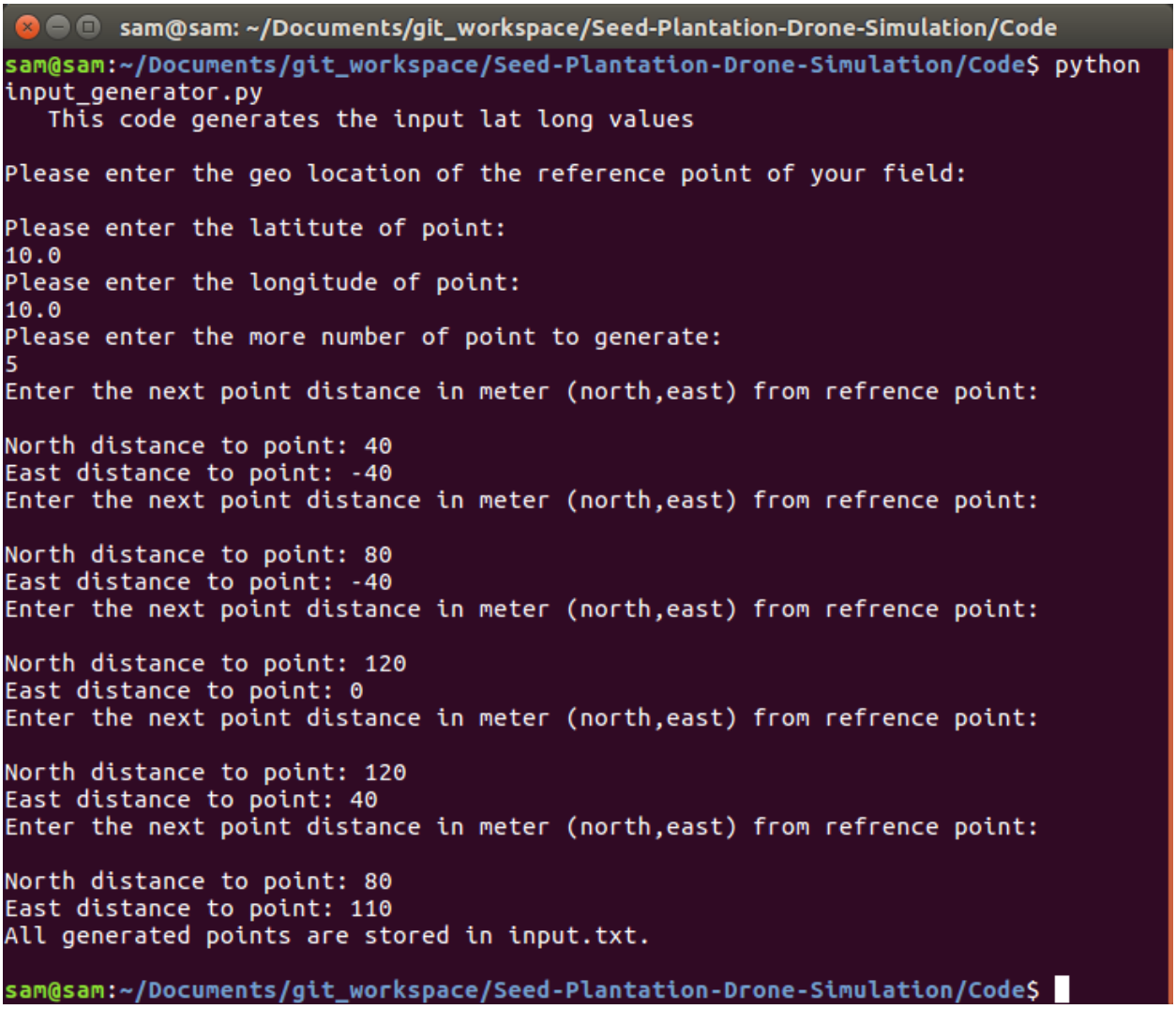
```
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 245:        USER entered latitute value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 250:        USER entered longitude value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 255:        USER entered altitude value: 30.0
INFO: drone_GUIDED.py: drone_GUIDED: <module>: 291:         Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: __init__: statustext_listener: 1065:         APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: __init__: statustext_listener: 1065:         Frame: QUAD
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:         Still waiting for data from vehicle: parameters
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 189:         Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 190:         Autopilot capabilities (supports ftp): False
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 191:         Global Location:LocationGlobal:lat=10.0,lon=10.0,al
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 192:         Global Location (relative altitude): LocationGlobal
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 193:         Local Location: LocationLocal:north=None,east=None,
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 194:         Attitude: Attitude:pitch=0.00102841900662,yaw=-3.13
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 195:         Velocity: [0.01, -0.01, 0.0]
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 196:         GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 197:         Groundspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 198:         Airspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 199:         Gimbal status: Gimbal: pitch=None, roll=None, yaw=N
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 200:         Battery: Battery:voltage=12.587,current=0.0,level=1
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 201:         EKF OK?: True
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 202:         Last Heartbeat: 0.669647895
```

8.5 Random Convex Polygonal Field (AUTO mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Random%20Convex%20Polygon%20AUTO

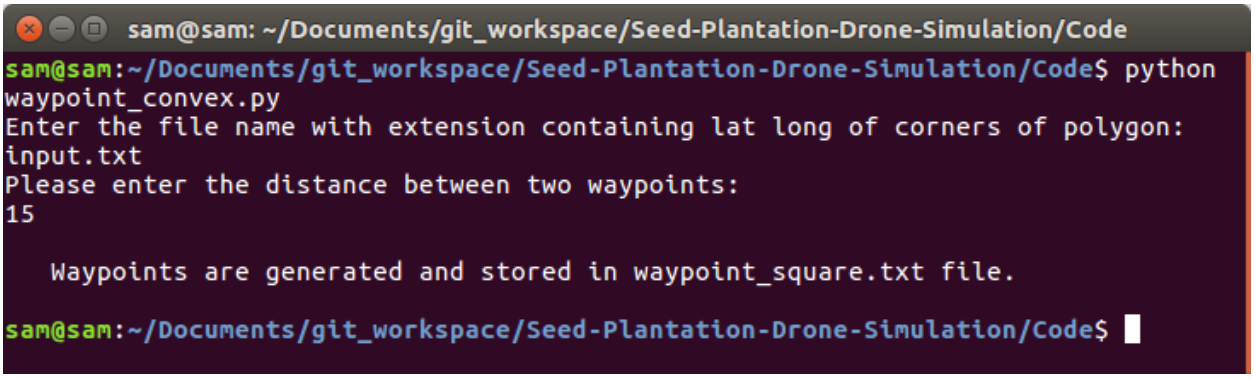
8.5.1 Input Generator file



8.5.2 Input Text file

```
10.0,10.0
10.0003593261,9.9996351307
10.0007186522,9.9996351307
10.0010779783,10.0
10.0010779783,10.0003648693
10.0007186522,10.0010033906
```

8.5.3 Convex Polygon waypoint generator terminal

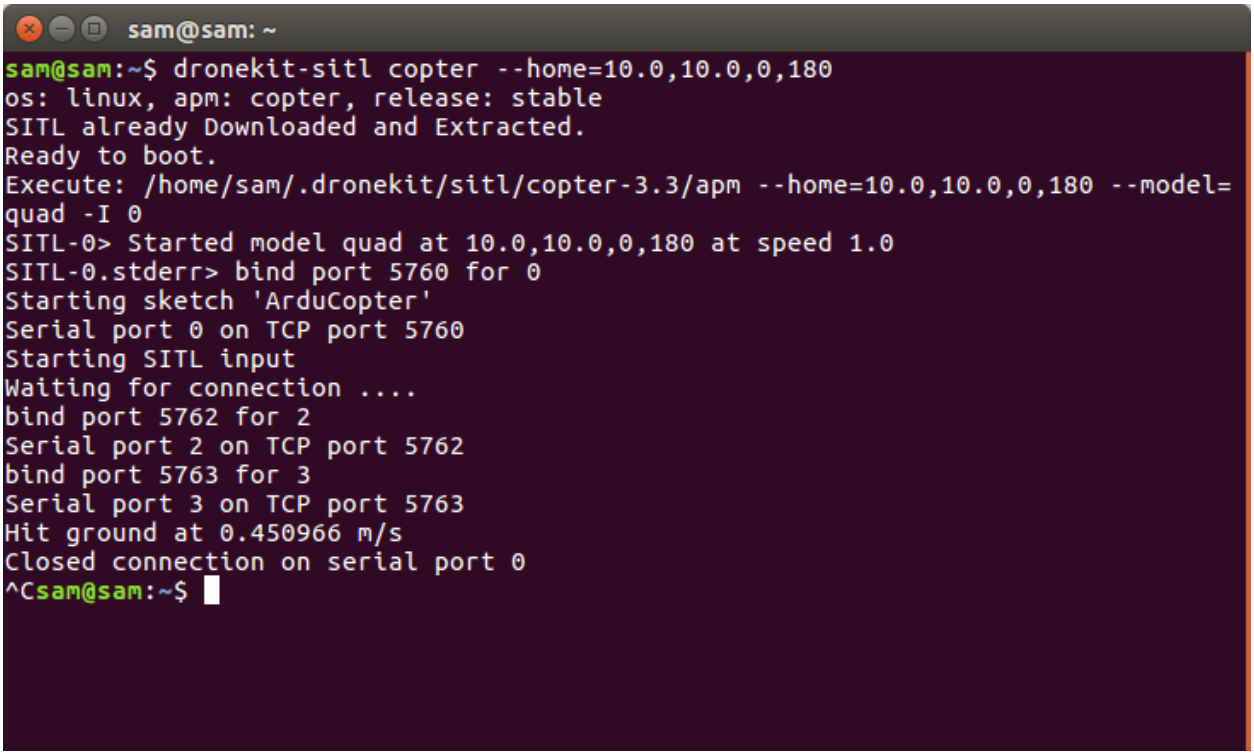


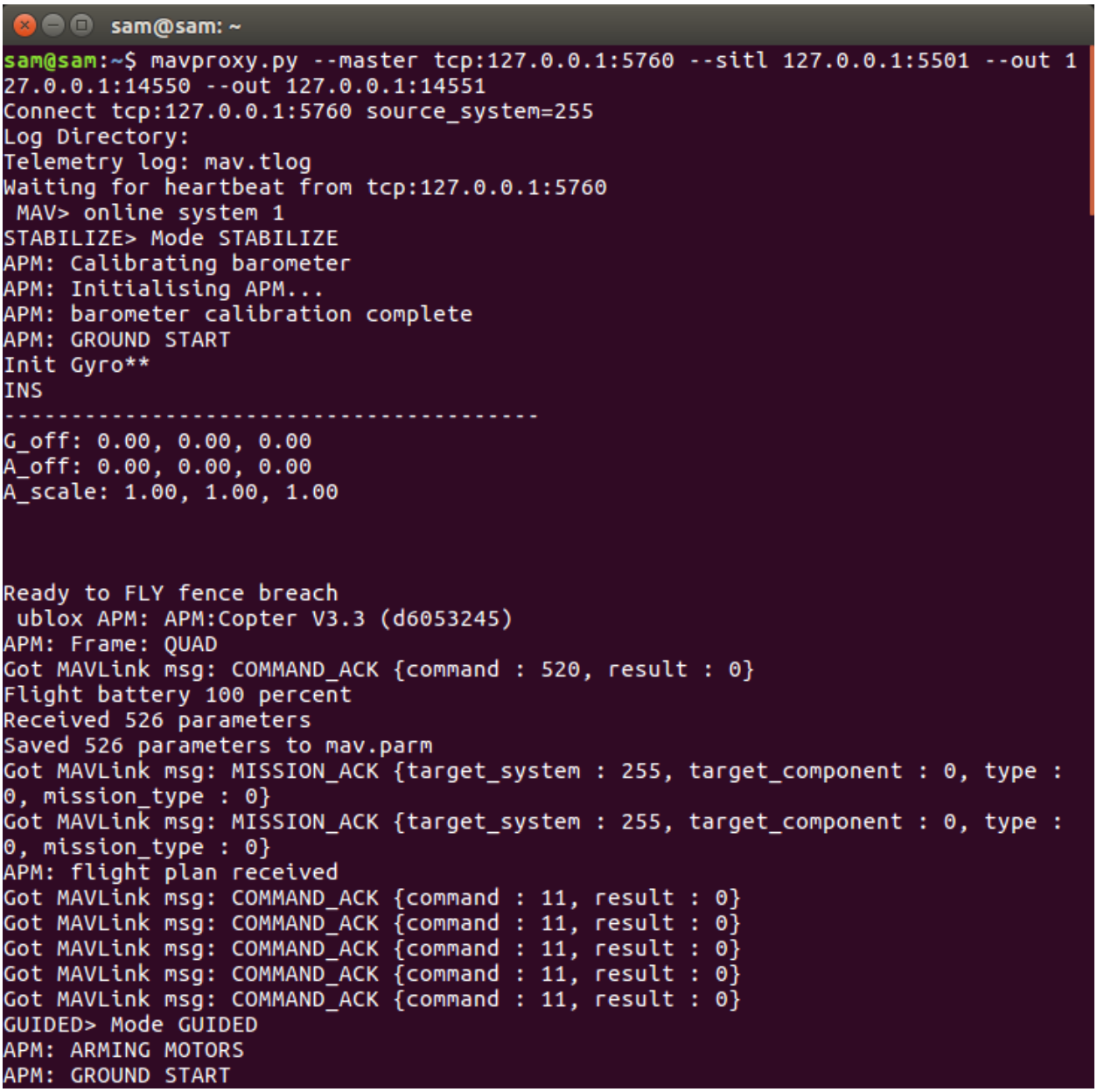
8.5.4 Waypoint convex text file

48 lines (48 sloc) | 1.28 KB

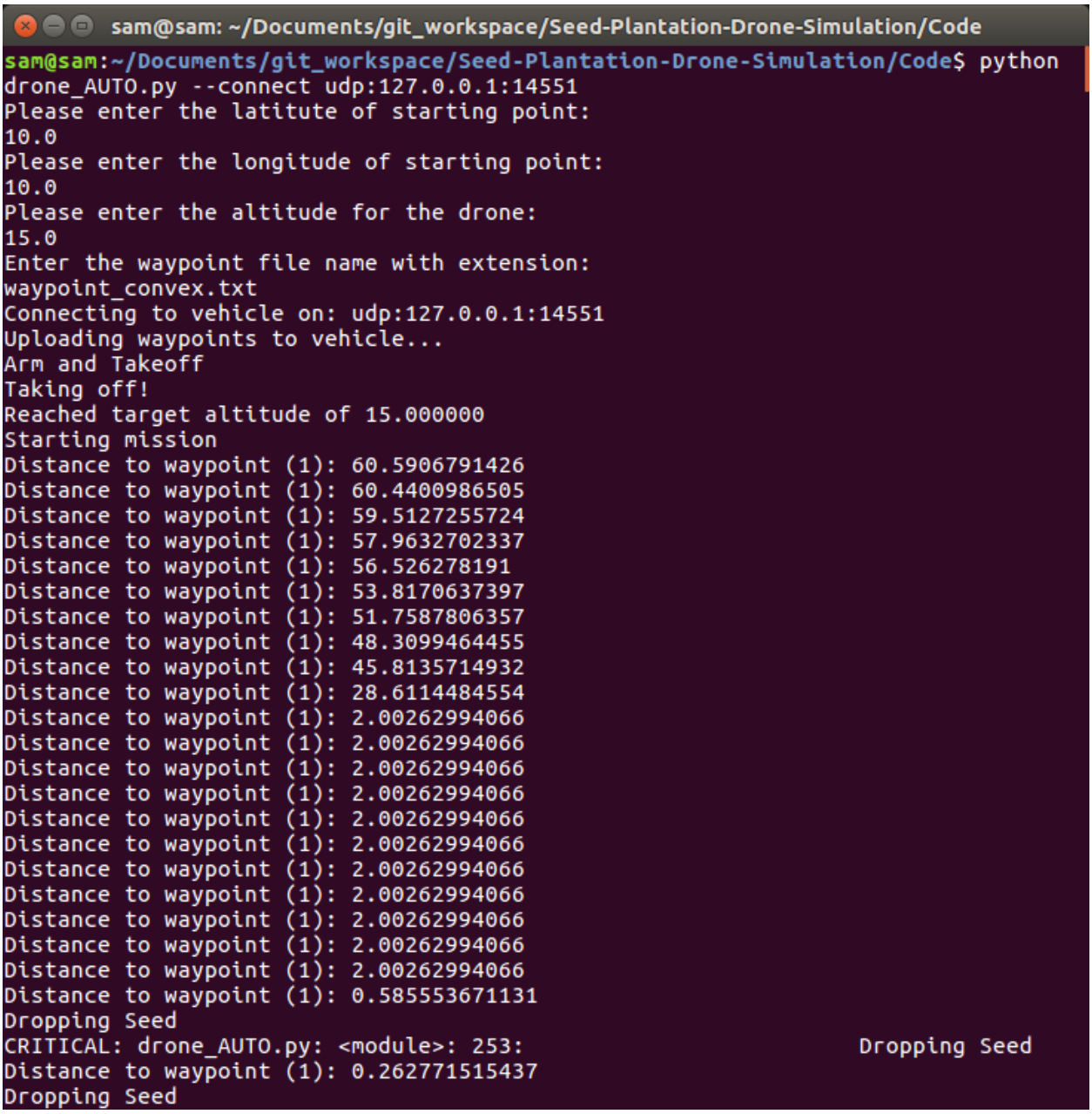
```
10.0004042419,9.9996351307
10.0005389892,9.9996351307
10.0006737365,9.9996351307
10.0008084838,9.99977195737
10.0006737365,9.99977195737
10.0005389892,9.99977195737
10.0004042419,9.99977195737
10.0002694946,9.99977195737
10.0001347473,9.99990878335
10.0002694946,9.99990878335
10.0004042419,9.99990878335
10.0005389892,9.99990878335
10.0006737365,9.99990878335
10.0008084838,9.99990878335
10.000943231,9.99990878335
10.000943231,10.00004561
10.0008084838,10.00004561
10.0006737365,10.00004561
10.0005389892,10.00004561
10.0004042419,10.00004561
```

8.5.5 Terminal outputs

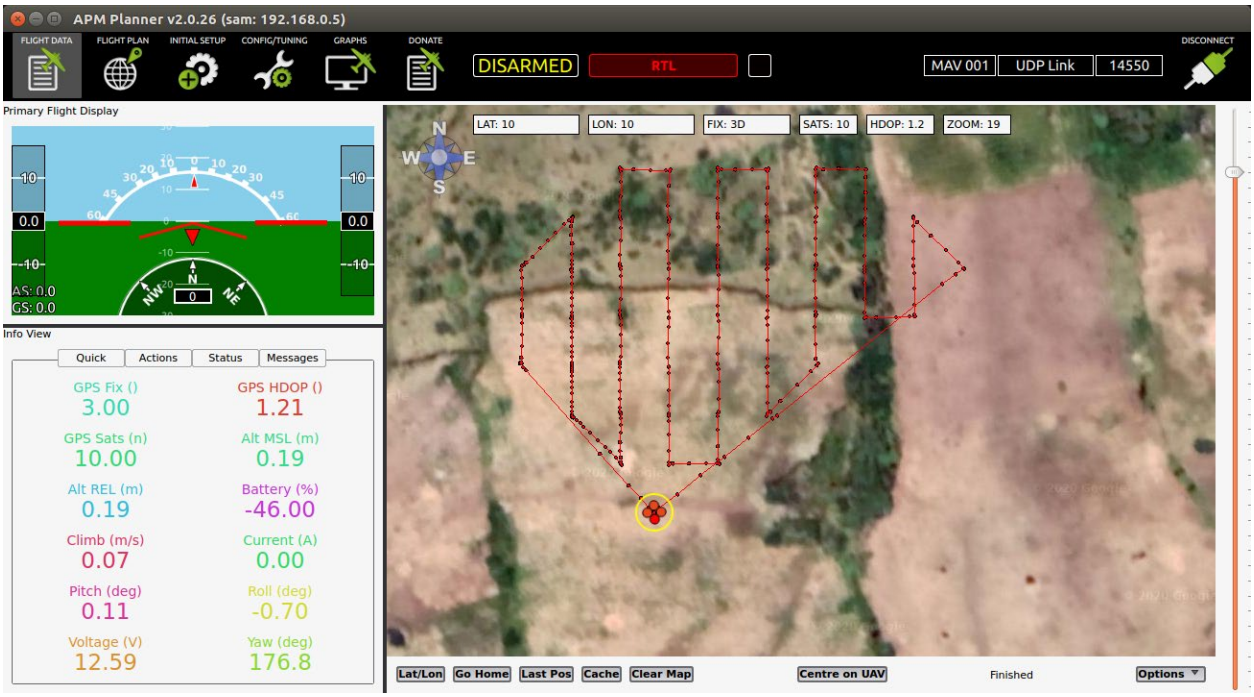



sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY fence breach
ublox APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Flight battery 100 percent
Received 526 parameters
Saved 526 parameters to mav.parm
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 255, target_component : 0, type :
0, mission_type : 0}
APM: flight plan received
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
GUIDED> Mode GUIDED
APM: ARMING MOTORS
APM: GROUND START


sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_AUTO.py --connect udp:127.0.0.1:14551
Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
15.0
Enter the waypoint file name with extension:
waypoint_convex.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Uploading waypoints to vehicle...
Arm and Takeoff
Taking off!
Reached target altitude of 15.000000
Starting mission
Distance to waypoint (1): 60.5906791426
Distance to waypoint (1): 60.4400986505
Distance to waypoint (1): 59.5127255724
Distance to waypoint (1): 57.9632702337
Distance to waypoint (1): 56.526278191
Distance to waypoint (1): 53.8170637397
Distance to waypoint (1): 51.7587806357
Distance to waypoint (1): 48.3099464455
Distance to waypoint (1): 45.8135714932
Distance to waypoint (1): 28.6114484554
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 2.00262994066
Distance to waypoint (1): 0.585553671131
Dropping Seed
CRITICAL: drone_AUTO.py: <module>: 253: Dropping Seed
Distance to waypoint (1): 0.262771515437
Dropping Seed

8.5.6 Convex Polygon APM Planner2 output



8.5.7 Convex Polygon log file

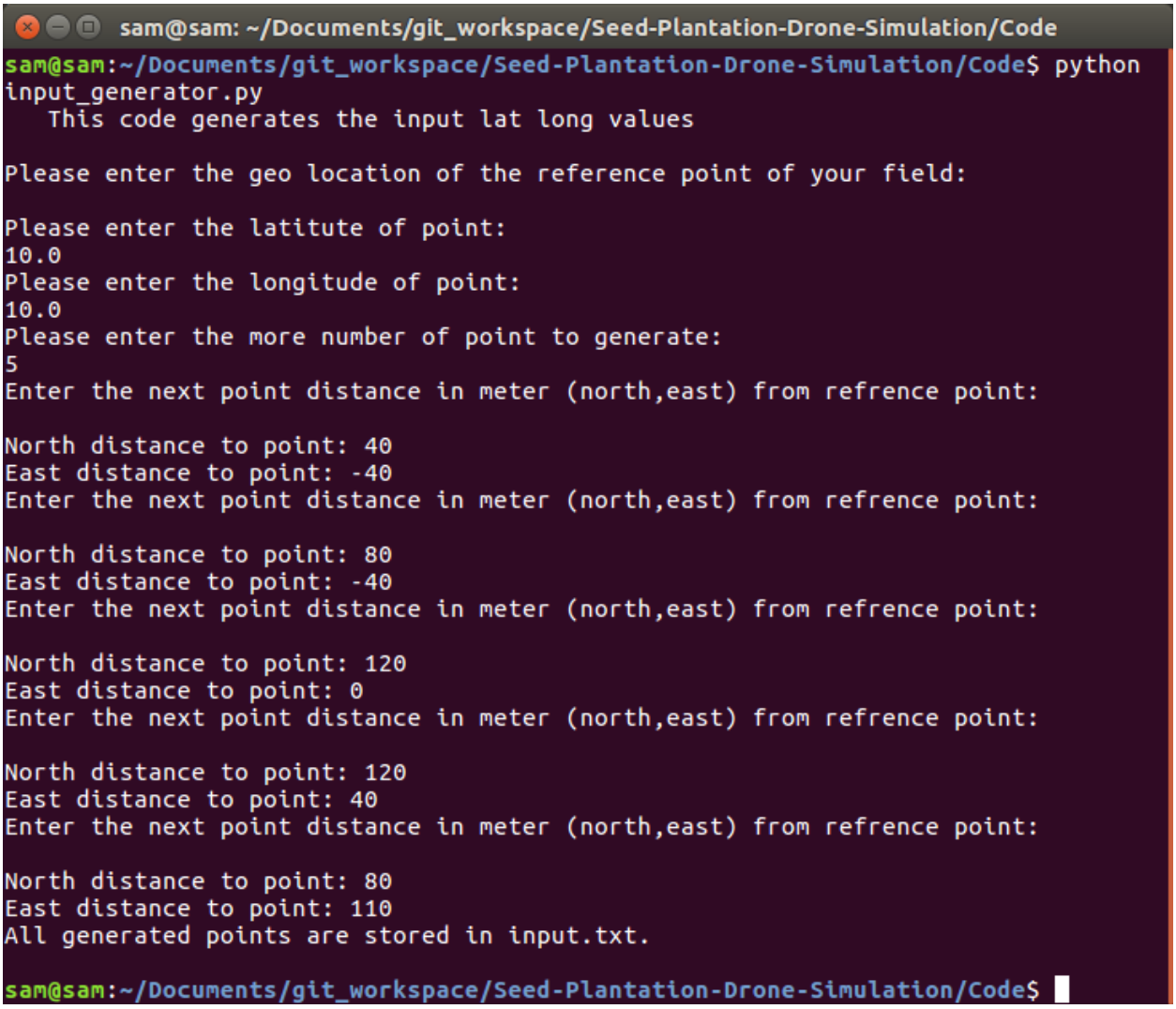
```
DEBUG: drone_AUTO.py: <module>: 141:                USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:                USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:                USER entered altitude value: 15.0
INFO: drone_AUTO.py: <module>: 187:                 Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: statustext_listener: 1065:               APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: statustext_listener: 1065:               Frame: QUAD
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
DEBUG: __init__.py: default_still_waiting_callback: 3082:               Still waiting for data from vehicle: parameters
INFO: drone_AUTO.py: print_vehicle_attributes: 98:              Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_AUTO.py: print_vehicle_attributes: 99:              Autopilot capabilities (supports ftp): False
INFO: drone_AUTO.py: print_vehicle_attributes: 100:             Global Location: LocationGlobal:lat=10.0,lon=10.0,alt=None
INFO: drone_AUTO.py: print_vehicle_attributes: 101:             Global Location (relative altitude): LocationGlobalRelative:lat=10.
INFO: drone_AUTO.py: print_vehicle_attributes: 102:             Local Location: LocationLocal:north=None,east=None,down=None
INFO: drone_AUTO.py: print_vehicle_attributes: 103:             Attitude: Attitude:pitch=0.00110931112431,yaw=-3.13855171204,roll=0
INFO: drone_AUTO.py: print_vehicle_attributes: 104:             Velocity: [0.01, -0.02, 0.0]
INFO: drone_AUTO.py: print_vehicle_attributes: 105:             GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_AUTO.py: print_vehicle_attributes: 106:             Groundspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 107:             Airspeed: 0.0
INFO: drone_AUTO.py: print_vehicle_attributes: 108:             Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
INFO: drone_AUTO.py: print_vehicle_attributes: 109:             Battery: Battery:voltage=12.587,current=0.0,level=100
INFO: drone_AUTO.py: print_vehicle_attributes: 110:             EKF OK?: True
INFO: drone_AUTO.py: print_vehicle_attributes: 111:             Last Heartbeat: 0.608151212999
```

8.6 Random Convex Polygonal Field (GUIDED mode) inputs and outputs

GitHub link:

https://github.com/sambhavesh/Seed-Plantation-Drone-Simulation/tree/master/Sample%20Input%20Output/Random%20Convex%20Polygon%20GUIDED

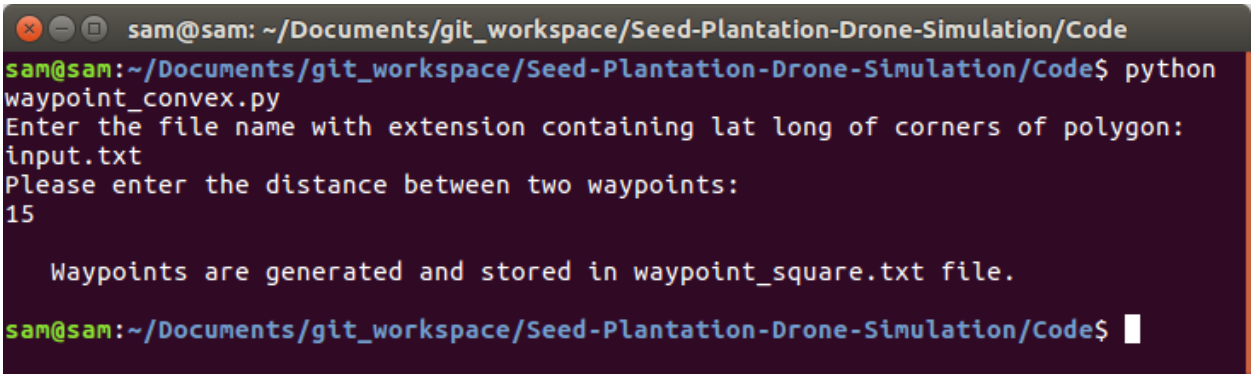
8.6.1 Input Generator file



8.6.2 Input Text file

```
10.0,10.0
10.0003593261,9.9996351307
10.0007186522,9.9996351307
10.0010779783,10.0
10.0010779783,10.0003648693
10.0007186522,10.0010033906
```

8.6.3 Convex Polygon waypoint generator terminal

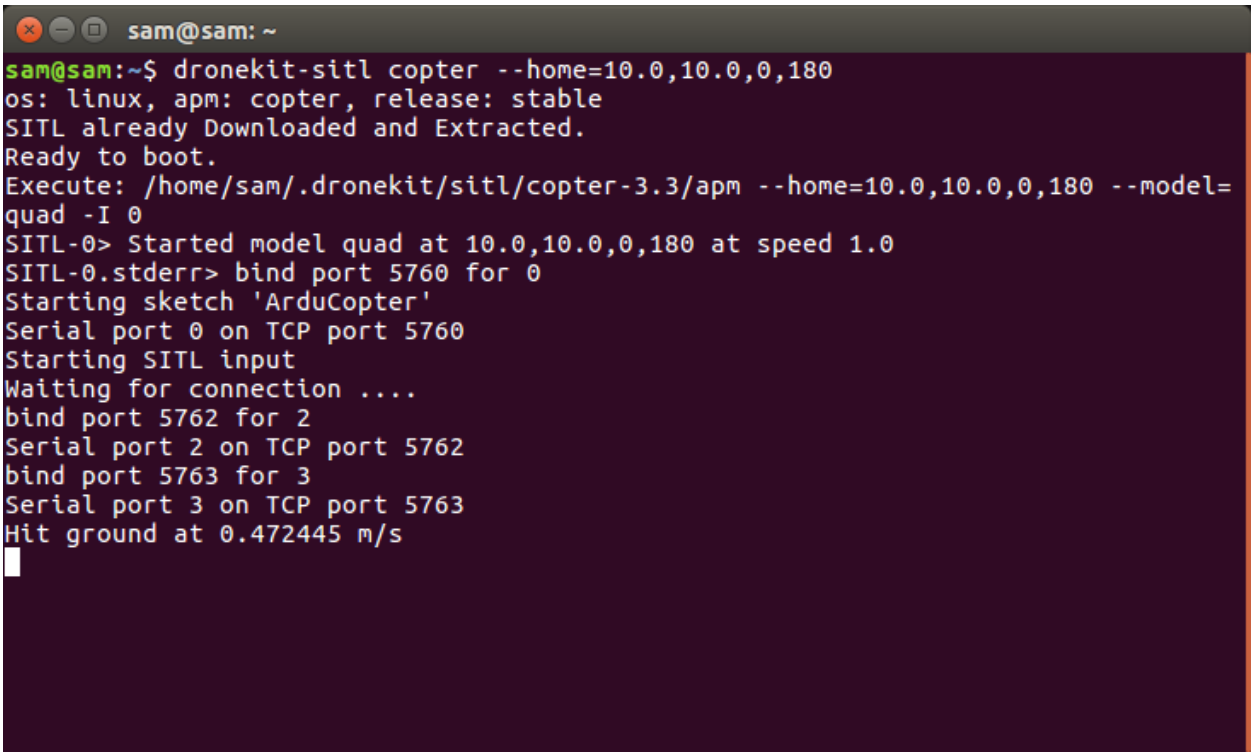


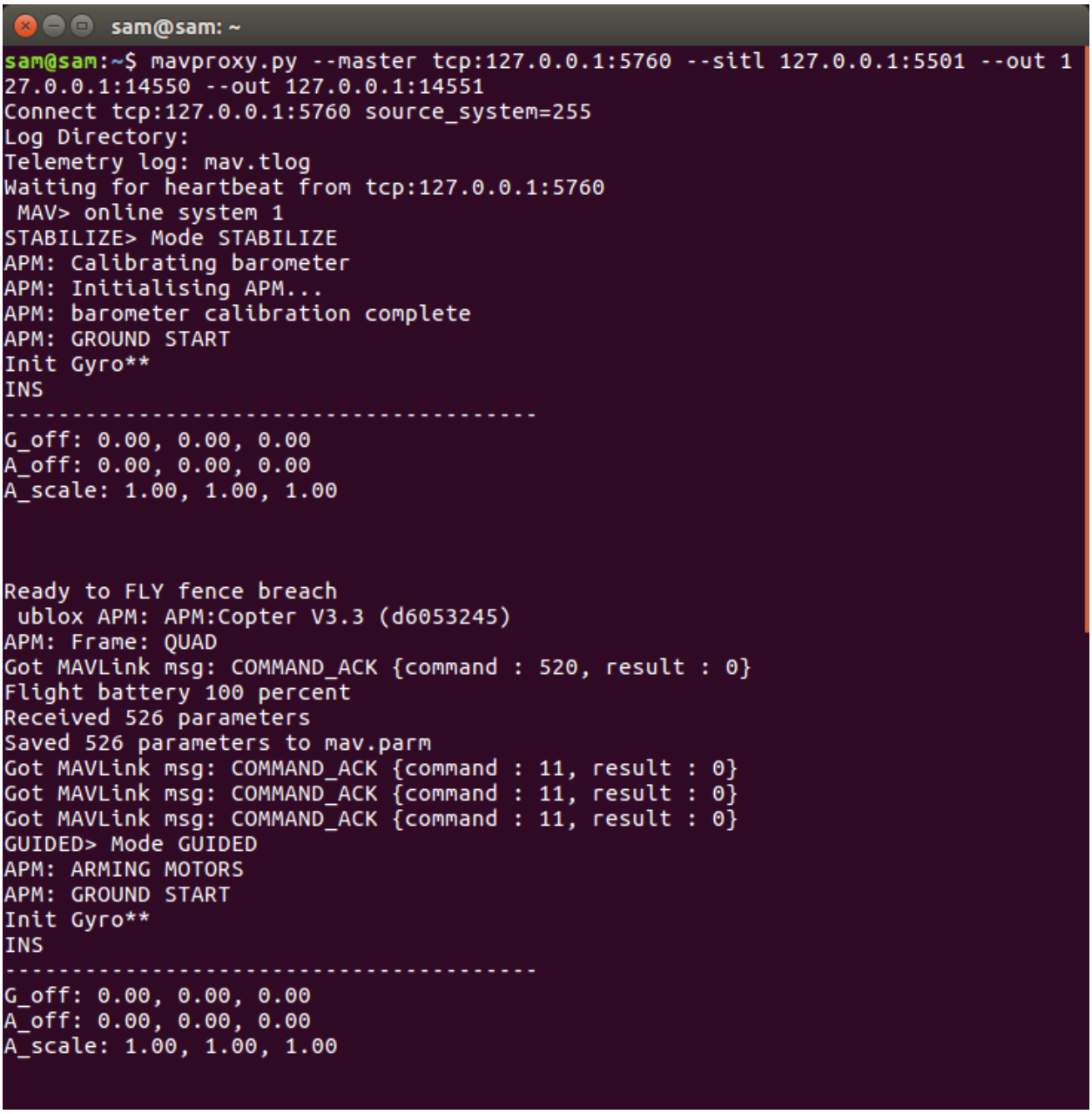
8.6.4 Waypoint convex text file

48 lines (48 sloc) | 1.28 KB

```
10.0004042419,9.9996351307
10.0005389892,9.9996351307
10.0006737365,9.9996351307
10.0008084838,9.99977195737
10.0006737365,9.99977195737
10.0005389892,9.99977195737
10.0004042419,9.99977195737
10.0002694946,9.99977195737
10.0001347473,9.99990878335
10.0002694946,9.99990878335
10.0004042419,9.99990878335
10.0005389892,9.99990878335
10.0006737365,9.99990878335
10.0008084838,9.99990878335
10.000943231,9.99990878335
10.000943231,10.00004561
10.0008084838,10.00004561
10.0006737365,10.00004561
10.0005389892,10.00004561
10.0004042419,10.00004561
```

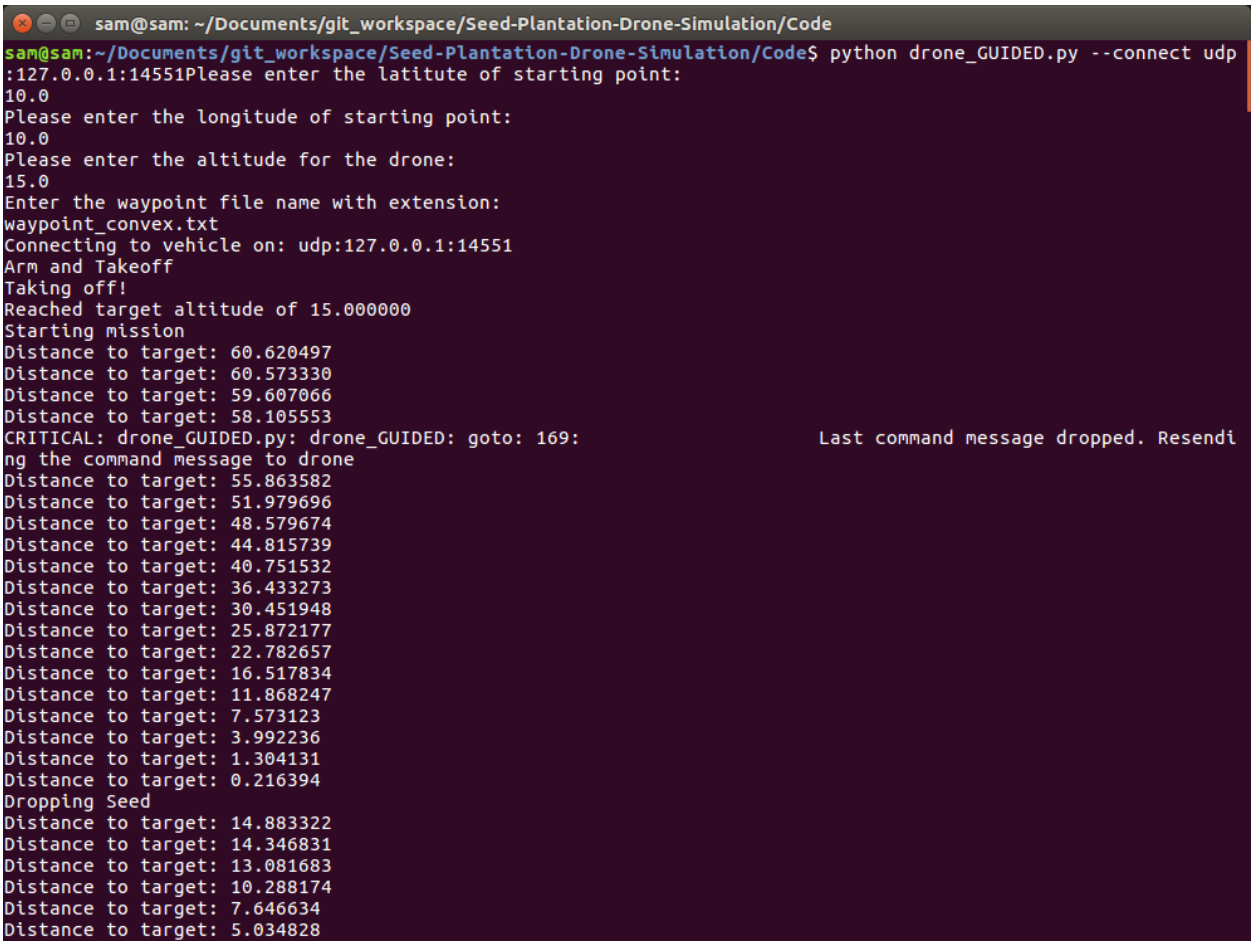
8.6.5 Terminal outputs



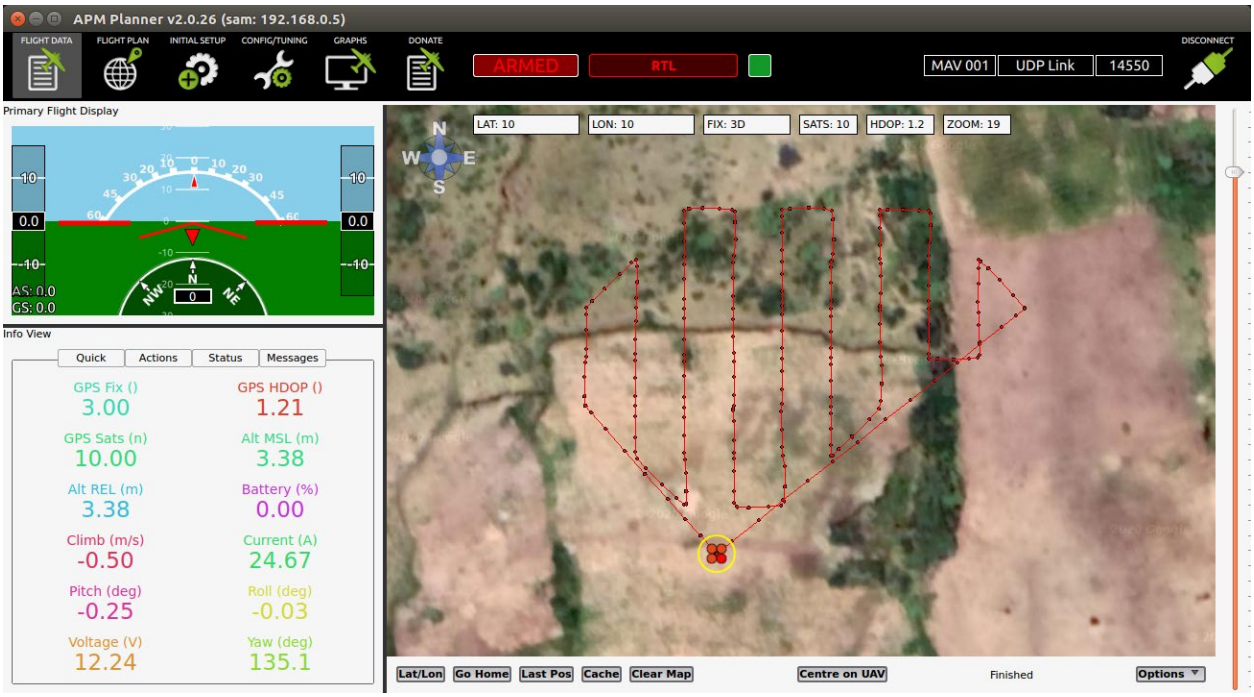

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
APM: Calibrating barometer
APM: Initialising APM...
APM: barometer calibration complete
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
Ready to FLY fence breach
ublox APM: APM:Copter V3.3 (d6053245)
APM: Frame: QUAD
Got MAVLink msg: COMMAND_ACK {command : 520, result : 0}
Flight battery 100 percent
Received 526 parameters
Saved 526 parameters to mav.parm
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
GUIDED> Mode GUIDED
APM: ARMING MOTORS
APM: GROUND START
Init Gyro**
INS

G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00


sam@sam: ~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code
sam@sam:~/Documents/git_workspace/Seed-Plantation-Drone-Simulation/Code$ python drone_GUIDED.py --connect udp
:127.0.0.1:14551Please enter the latitute of starting point:
10.0
Please enter the longitude of starting point:
10.0
Please enter the altitude for the drone:
15.0
Enter the waypoint file name with extension:
waypoint_convex.txt
Connecting to vehicle on: udp:127.0.0.1:14551
Arm and Takeoff
Taking off!
Reached target altitude of 15.000000
Starting mission
Distance to target: 60.620497
Distance to target: 60.573330
Distance to target: 59.607066
Distance to target: 58.105553
CRITICAL: drone_GUIDED.py: drone_GUIDED: goto: 169: Last command message dropped. Resendi
ng the command message to drone
Distance to target: 55.863582
Distance to target: 51.979696
Distance to target: 48.579674
Distance to target: 44.815739
Distance to target: 40.751532
Distance to target: 36.433273
Distance to target: 30.451948
Distance to target: 25.872177
Distance to target: 22.782657
Distance to target: 16.517834
Distance to target: 11.868247
Distance to target: 7.573123
Distance to target: 3.992236
Distance to target: 1.304131
Distance to target: 0.216394
Dropping Seed
Distance to target: 14.883322
Distance to target: 14.346831
Distance to target: 13.081683
Distance to target: 10.288174
Distance to target: 7.646634
Distance to target: 5.034828

8.6.6 Convex Polygon APM Planner2 output



8.6.7 Convex Polygon log file

```
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 245:                    USER entered latitute value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 250:                    USER entered longitude value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 255:                    USER entered altitude value: 15.0
INFO: drone_GUIDED.py: drone_GUIDED: <module>: 291:                     Connecting to vehicle on: udp:127.0.0.1:14551
CRITICAL: __init__.py: __init__: statustext_listener: 1065:                     APM:Copter V3.3 (d6053245)
CRITICAL: __init__.py: __init__: statustext_listener: 1065:                     Frame: QUAD
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
DEBUG: __init__.py: __init__: default_still_waiting_callback: 3082:                     Still waiting for data from vehicle: parameters
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 189:                     Autopilot Firmware version: APM:Copter-3.3.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 190:                     Autopilot capabilities (supports ftp): False
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 191:                     Global Location:LocationGlobal:lat=10.0,lon=10.0,al
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 192:                     Global Location (relative altitude): LocationGlobal
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 193:                     Local Location: LocationLocal:north=None,east=None,
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 194:                     Attitude: Attitude:pitch=0.000933106115554,yaw=-3.1
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 195:                     Velocity: [0.0, -0.01, 0.0]
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 196:                     GPS: GPSInfo:fix=3,num_sat=10
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 197:                     Groundspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 198:                     Airspeed: 0.0
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 199:                     Gimbal status: Gimbal: pitch=None, roll=None, yaw=N
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 200:                     Battery: Battery:voltage=12.587,current=0.0,level=1
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 201:                     EKF OK?: True
INFO: drone_GUIDED.py: drone_GUIDED: print_vehicle_attributes: 202:                     Last Heartbeat: 0.6662882
```

Chapter 9. Conclusion and Future Scope

In this project work, we designed the algorithm to spray seeds at regular intervals in fields whose parameters are provided by the user.

The time complexity of the proposed algorithm is O((l/d)^2) where l is the side of the fitted square and d is the distance between seeds dropped. So basically, the proposed algorithm works in O(n^2).

Though this algorithm is best to spray seeds in fields whose parameters are known.

There is certain research/work which needs to be done for better application of drone for reforestation or afforestation. Some of them include: -

Software Advancement

We as a human cannot reach many barren lands due to bad terrain or unavailability of paths, new research work can be put where we send drone beforehand to analyze the land piece and collect data about the quality of land and its dimension and weather conditions. In this research, a lot of IoT and AI Edge knowledge is required along with computer vision.

Hardware Advancement

A deep study needs to be done on how to carry seeds through drone to minimize the payload as well as maintain effective plantation rate also. As well as the firing speed needs to be controlled so as to maintain such a speed that the seed enters the soil otherwise an artificial pod needs to be made to contain the seed so as wherever it falls the tree is planted without being covered by soil.

References

1) https://dronekit-python.readthedocs.io/en/latest/

2) http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-of-meters

3) https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/common.py

4) https://ardupilot.org/copter/index.html