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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT REPORT
SESSION (2019-20)
VIII SEM

Submitted To:

Dr. 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.

Drones are the tools that have evolved rapidly over time and widely used in many sectors, from archaeology to environment. Drones will be an essential applicable tool for conservation and revegetation practices. It can provide not only efficient and effective methods but also a cost-effective and less harsh panacea to the environmental managers. Their logistic abilities, agility, and image quality make them helpful tools.
Revegetation programs defined the forest rehabilitation approaches and technical intervention methods such as seed planting. In India, various tree species are being used for revegetation. However, current ongoing revegetation practices are facing challenges. The challenges are related to the preparation and operation phase. Revegetation practices require a comprehensive assessment of lands in the preparation stage. The traditional revegetation practices by commissioning direct seeding through hand sowing are also facing difficulty. This method requires laborers. Furthermore, the coverage of laborers is restrained by access. Consequently, sowing cannot reach areas that are obstructed by hills and rivers, keeping in mind the safety of the workers. Concerning efficiency, it is estimated that a drone can distribute at least 10 seeds per minute. Due to its nature, drones have the capability in terms of optical and transportation features. Hence, through this project, we introduce the use of a drone to support revegetation practices. The project aims to develop a quadcopter drone that is capable of dispersing seeds in revegetation fields with potential for revegetation.

Agriculture in India constitutes more than 60% of occupation. It serves as the backbone of our country's economy. It is essential to improve the productivity and efficiency of agriculture. 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 are installed from **pip** on all platforms. On Linux you will first 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 create apps that run on an onboard companion computer and communicate with the ArduPilot flight controller using a low-latency link. Onboard apps can significantly enhance the autopilot, adding greater intelligence to vehicle behaviour, and performing tasks that are computationally intensive or time-sensitive (for example, computer vision, path planning, or 3D modelling). DroneKit-Python can also be used for ground station apps, communicating with vehicles over a higher latency RF-link.

The API communicates with vehicles over MAVLink. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

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

2.1.1 API Features

The API provides classes and methods to:

- Connect to a vehicle (or multiple vehicles) from a script
- Get and set vehicle state/telemetry and parameter information.
- Receive asynchronous notification of state changes.
- Guide a UAV to specified position (GUIDED mode).
- Send arbitrary custom messages to control UAV movement and other hardware (GUIDED mode).
- Create and manage waypoint missions (AUTO mode).
- Override RC channel settings

2.1.2 Launching scripts

DroneKit-Python 2.0 apps are run from an ordinary terminal. For example:

```
python some_python_script.py
```

2.2 Setting up a Simulated Vehicle (SITL)

The SITL (Software in The Loop) simulator allows you to create and test DroneKit-Python apps without a real vehicle (and from the comfort of your own developer desktop!).

SITL can run natively on Linux (x86 architecture only), 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, fastest and easiest way to run SITL on Windows, Linux (x86 architecture only), or Mac OS X. It is installed from Python's pip tool on all platforms, and works by downloading and running pre-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

- It is a command-line, console-based app. There are plugins included in MAVProxy to provide a basic GUI.
- Can be networked and run over any number of computers.
- It's portable; it should run on any POSIX OS with python, pyserial, and select() function calls, which means Linux, OS X, Windows, and others.
- The light-weight design means it can run on small netbooks with ease.
- It supports loadable modules, and has modules to support console/s, moving maps, joysticks, antenna trackers, etc.
- Tab-completion of commands

2.4 APM Planner

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

2.4.1 Features

- Configure and calibrate your ArduPilot or PX4 autopilot for autonomous vehicle control.
- Plan a mission with GPS waypoints and control events.
- Connect a 3DR Radio to view live data and initiate commands 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, Command, VehicleMode, LocationGlobalRelati
ve
from pymavlink import mavutil
import os
import json, urllib, math
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")
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logFile_handler = logging.FileHandler("drone_seed_AUTO.log")
logFile_handler.setLevel(logging.DEBUG)
logFile_streamHandler = logging.StreamHandler()
logFile_streamHandler.setLevel(logging.ERROR)
logging_formatter = logging.Formatter("%(levelname)s: %(filename)s: %(fu
ncName)s: %(lineno)d:                    %(message)s")
logFile_handler.setFormatter(logging_formatter)
logFile_streamHandler.setFormatter(logging_formatter)
logger.addHandler(logFile_handler)
logger.addHandler(logFile_streamHandler)
```

Functions Used:

1. get_distance_metres(aLocation1, aLocation2):

Returns the ground distance in metres between two LocationGlobal objects.

This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

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

In [ ]:
```
def get_distance_metres(aLocation1, aLocation2):
        dlat = aLocation2.lat - aLocation1.lat
        dlong = aLocation2.lon - aLocation1.lon
        return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

2. distance_to_current_waypoint():

Gets distance in metres to the current waypoint. It returns "None" for the first waypoint (Home location).

In [ ]:
```
def distance_to_current_waypoint():
        nextwaypoint = vehicle.commands.next
        if nextwaypoint==0:
                return None
        missionitem=vehicle.commands[nextwaypoint-1] #commands are zero
indexed
        lat = missionitem.x
        lon = missionitem.y
        alt = missionitem.z
        targetWaypointLocation = LocationGlobalRelative(lat,lon,alt)
        distancetopoint = get_distance_metres(vehicle.location.global_fr
ame, targetWaypointLocation)
        return distancetopoint
```

3. arm_and_takeoff(aTargetAltitude):

Arms vehicle and fly to a target altitude. Don't try to arm until autopilot is ready.

In [ ]:
```
def arm_and_takeoff(aTargetAltitude):
        while not vehicle.is_armable:
                logger.warning("Waiting for vehicle to initialise...")
                time.sleep(1)
```

Set mode to GUIDED for arming and takeoff:

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

Confirm vehicle armed before attempting to take off:

In [ ]:
```
        while not vehicle.armed:
                vehicle.armed = True
                logger.warning("Waiting for arming...")
                time.sleep(1)
        print("Taking off!")
        logger.info("Taking off!")
        vehicle.simple_takeoff(aTargetAltitude) # Take off to target alt
itude
```

Wait until the vehicle reaches a safe height before allowing next command to process:

In [ ]:
```
        while True:
                requiredAlt = aTargetAltitude*0.95
                #Break and return from function just below target altitu
de.
                if vehicle.location.global_relative_frame.alt>=requiredA
lt:
                        print("Reached target altitude of %f" % (aTarget
Altitude))
                        logger.info("Reached target altitude of %f" % (a
TargetAltitude))
                        break
                logger.info("Altitude: %f < %f" % (vehicle.location.glob
al_relative_frame.alt,requiredAlt))
                time.sleep(1)
```

4. print_vechicle_attributes():

This function list all the attributes of the vehicle and stores it in log file:

In [ ]:
```
def print_vehicle_attributes():
        logger.info("Autopilot Firmware version: %s" % vehicle.version)
        logger.info("Autopilot capabilities (supports ftp): %s" % vehicl
e.capabilities.ftp)
        logger.info("Global Location:INFO:__main__: Key:BATT_CURR_PIN Va
lue:12.0 %s" % vehicle.location.global_frame)
        logger.info("Global Location (relative altitude): %s" % vehicle.
location.global_relative_frame)
        logger.info("Local Location: %s" % vehicle.location.local_frame)
        logger.info("Attitude: %s" % vehicle.attitude)
        logger.info("Velocity: %s" % vehicle.velocity)
        logger.info("GPS: %s" % vehicle.gps_0)
        logger.info("Groundspeed: %s" % vehicle.groundspeed)
        logger.info("Airspeed: %sINFO:__main__:Distance to waypoint (2):
50.5458561177" % vehicle.airspeed)
        logger.info("Gimbal status: %s" % vehicle.gimbal)
        logger.info("Battery: %s" % vehicle.battery)
        logger.info("EKF OK?: %s" % vehicle.ekf_ok)
        logger.info("Last Heartbeat: %s" % vehicle.last_heartbeat)
        logger.info("Rangefinder: %s" % vehicle.rangefinder)
        logger.info("Rangefinder distance: %s" % vehicle.rangefinder.dis
tance)
        logger.info("Rangefinder voltage: %s" % vehicle.rangefinder.volt
age)
        logger.info("Heading: %s" % vehicle.heading)
        logger.info("Is Armable?: %s" % vehicle.is_armable)
        logger.info("System status: %s" % vehicle.system_status.state)
        logger.info("Mode: %s" % vehicle.mode.name)
        logger.info("Armed: %s" % vehicle.armed)
```

5. print_vechicle_parameters():

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

In [ ]:
```
def print_vehicle_parameters():
        logger.info ("Print all parameters (`vehicle.parameters`):")
        for key, value in vehicle.parameters.iteritems():
                logger.info (" Key:%s Value:%s" % (key,value))
```

Main Body :

In [ ]:
```
start_lat = 0.0         #latitute variable
start_lon = 0.0         #longitude variable
start_alt = 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:
                start_lat = float(input("Please enter the latitute of st
arting point:\n"))
                logger.debug("USER entered latitute value: %s",str(start
_lat))
                if(start_lat<0 or start_lat>90):
                        print("Latitude value must be between 0 and 90")
                        continue
                start_lon = float(input("Please enter the longitude of s
tarting point:\n"))
                logger.debug("USER entered longitude value: %s",str(star
t_lon))
                if(start_lon<0 or start_lon>180):
                        print("Langitude value must be between 0 and 18
0")
                        continue
                start_alt = float(input("Please enter the altitude for t
he drone:\n"))
                logger.debug("USER entered altitude value: %s",str(start
_alt))
                if(start_alt<0):
                        print("Altitude value must be positive")
                        continue
                break
        except:
                logger.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")
                logger.error("Enter file does not exists.")
                continue
```

Set up option parsing to get connection string

In [ ]:
```
import argparse
parser = argparse.ArgumentParser(description='Demonstrates Seed Plantati
on Mission.')
parser.add_argument('--connect', help="vehicle connection target string.
If not specified, SITL automatically started and used.")
args = parser.parse_args()
connection_string = args.connect
sitl = None
```

Start SITL if no connection string specified

In [ ]:
```
if not connection_string:
        import dronekit_sitl
        sitl = dronekit_sitl.start_default(lat=start_lat,lon=start_lon)
        connection_string = sitl.connection_string()
```

Connect to the Vehicle

In [ ]:
```
print('Connecting to vehicle on: %s' % connection_string)
logger.info('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True)
```

Log vehicle attributes:

In [ ]:
```
print_vehicle_attributes()
```

Log vehicle parameters:

In [ ]:
```
print_vehicle_parameters()
```

Now download the vehicle waypoints

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

Add command for starting location:

In [ ]:
```
cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavut
il.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 0, 0, 0, 0,start_lat, start_lon,st
art_alt)
cmds.add(cmd)
```

Open the waypoint file and 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])
                logger.debug ("Point: %f %f" %(lat, lon))
                cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RE
LATIVE_ALT,mavutil.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 5, 0, 0, 0,lat, lo
n, start_alt)
                cmds.add(cmd)
        """
                Add the codes/ mechanism for dropping seed here. Depends
on hardware

        """
way_p.close()
```

Add command for last location i.e launch location:

In [ ]:
```
cmd = Command( 0,0,0,mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT,mavut
il.mavlink.MAV_CMD_NAV_WAYPOINT,0, 0, 0, 0, 0,start_lat, start_lon,st
art_alt)
cmds.add(cmd)
```

Upload clear message and command messages to vehicle.

In [ ]:
```
print("Uploading waypoints to vehicle..." )
logger.info("Uploading waypoints to vehicle...")
cmds.upload()
print("Arm and Takeoff")
logger.info("Arm and Takeoff")
arm_and_takeoff(start_alt)

print("Starting mission")
logger.info("Starting mission")
```

Reset mission set to first (0) waypoint

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

Set mode to AUTO to start mission:

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

Monitor mission

Calculate 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:
        nextwaypoint=vehicle.commands.next
        print('Distance to waypoint (%s): %s' % (nextwaypoint, distance_
to_current_waypoint()))
        logger.info('Distance to waypoint (%s): %s' % (nextwaypoint, dis
tance_to_current_waypoint()))
        if distance_to_current_waypoint()<1.5:
                print("Dropping Seed")
                logger.critical("Dropping Seed")
        if nextwaypoint==line_count+1:
                print("Exit 'standard' mission when start heading to fin
al waypoint or start location")
                logger.info("Exit 'standard' mission when start heading
to final waypoint or start location")
                break;
        time.sleep(1)

print('Return to launch')
logger.critical("Return to launch")
while (vehicle.mode.name != "RTL"):
        vehicle.mode = VehicleMode("RTL")
        time.sleep(0.1)
```

Finally close vehicle object before exiting script

In [ ]:
```
print("Close vehicle object")
logger.info("Close vehicle object")
vehicle.close()
```

Shut down simulator:

In [ ]:
```
if sitl is not None:
        sitl.stop()
print("Completed...")
logger.info("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 dronekit import connect, Command, VehicleMode, LocationGlobalRelati
ve, LocationGlobal
from pymavlink import mavutil
import os
import json, urllib, math
import time
import logging , logging.handlers
```

Logging configuration:

In [ ]:
```
logging.basicConfig(filename = "Master.log" , level = logging.DEBUG , fo
rmat = "%(levelname)s: %(filename)s: %(module)s: %(funcName)s: %(lineno)
d:                        %(message)s")
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logFile_handler = logging.FileHandler("drone_seed_GUIDED.log")
logFile_handler.setLevel(logging.DEBUG)
logFile_streamHandler = logging.StreamHandler()
logFile_streamHandler.setLevel(logging.ERROR)
logging_formatter = logging.Formatter("%(levelname)s: %(filename)s: %(mo
dule)s: %(funcName)s: %(lineno)d:                          %(message)s")
logFile_handler.setFormatter(logging_formatter)
logFile_streamHandler.setFormatter(logging_formatter)
logger.addHandler(logFile_handler)
logger.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. get_location_metres(original_location, dNorth, dEast):

Returns a LAT_LON_ALT object containing the latitude/longitude and altitude `dNorth` and `dEast` metres from the specified `original_location` . The function is useful when you want to move the vehicle around specifying locations relative to the current vehicle position. This function is relatively accurate over small distances (10m within 1km) except close to the poles.

The function does not change the altitude value

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

In [ ]:
```
def get_location_metres(original_location, dNorth, dEast):
        #Radius of "spherical" earth
        earth_radius=6378137.0

        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.la
t/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
        newlon = original_location.lon + (dLon * 180/math.pi)
        new_location = LAT_LON_ALT(newlon,newlat,original_location.alt)
        return new_location
```

2. get_distance_metres(aLocation1, aLocation2):

Returns the ground distance in metres between two LAT_LON_ALT objects.

This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

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

In [ ]:
```
def get_distance_metres(aLocation1, aLocation2):
    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

3. arm_and_takeoff(aTargetAltitude):

Arms vehicle and fly to a target altitude.

In [ ]:
```
def arm_and_takeoff(aTargetAltitude):
    # Don't try to arm until autopilot is ready
        while not vehicle.is_armable:
            logger.warning(" Waiting for vehicle to initialise...")
            time.sleep(1)
        # Set mode to GUIDED for arming and takeoff:
        while (vehicle.mode.name != "GUIDED"):
            vehicle.mode = VehicleMode("GUIDED")
            time.sleep(0.1)
        # Confirm vehicle armed before attempting to take off
        while not vehicle.armed:
            vehicle.armed = True
            logger.warning(" Waiting for arming...")
            time.sleep(1)
        print("Taking off!")
        logger.info("Taking off!")
        vehicle.simple_takeoff(aTargetAltitude) # Take off to target alt
itude

        # Wait until the vehicle reaches a safe height
        # before allowing next command to process.
        while True:
            requiredAlt = aTargetAltitude*0.95
            #Break and return from function just below target altitude.
            if vehicle.location.global_relative_frame.alt>=requiredAlt:
                print("Reached target altitude of %f" % (aTargetAltitud
e))
                logger.info("Reached target altitude of %f" % (aTargetAl
titude))
                break
            logger.info("Altitude: %f < %f" % (vehicle.location.global_r
elative_frame.alt,requiredAlt))
            time.sleep(1)
```

4. goto(targetLocation):

Send SET_POSITION_TARGET_GLOBAL_INT command to request the vehicle fly to a specified LocationGlobal. At time of writing, acceleration and yaw bits are ignored.

```
msg = vehicle.message_factory.set_position_target_global_int_encode(
    0,       # time_boot_ms (not used)
    0, 0,    # target system, target component
    mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, # frame
    0b0000111111111000, # type_mask (only speeds enabled)
    aLocation.lat, # lat_int - X Position in WGS84 frame in 1e7 * meters
    aLocation.lon, # lon_int - Y Position in WGS84 frame in 1e7 * meters
    aLocation.alt, # alt - Altitude in meters in AMSL altitude, not WGS84 if ab
solute or relative, above  terrain if GLOBAL_TERRAIN_ALT_INT
    0, # X velocity in NED frame in m/s
    0, # Y velocity in NED frame in m/s
    0, # Z velocity in NED frame in m/s
    0, 0, 0, # afx, afy, afz acceleration (not supported yet, ignored in GCS_Ma
vlink)
    0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
```

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 vehicle
        logger.debug("Target location lat: %f , lon: %f , alt: %f" % (ta
rgetLocation.lat,targetLocation.lon,targetLocation.alt))
        vc_in_loc = vehicle.location.global_relative_frame
        vehicle_initialLocation = LAT_LON_ALT(vc_in_loc.lon,vc_in_loc.la
t,vc_in_loc.alt)
        targetDistance = get_distance_metres(vehicle_initialLocation, ta
rgetLocation)
        msg = vehicle.message_factory.set_position_target_global_int_enc
ode( 0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT, 0b00001
11111111000, targetLocation.lat*1e7, targetLocation.lon*1e7, targetLocat
ion.alt, 0, 0, 0, 0, 0, 0, 0, 0)
        vehicle.send_mavlink(msg)
        logger.debug("Send Command Message to drone")
        # target = LocationGlobal(targetLocation.lat,targetLocation.lon,
targetLocation.alt)
        # vehicle.airspeed=15
        # vehicle.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
        logger.debug("fiveSecondCheck distance: %f " % (fiveSecondChec
k))
        logger.debug("fiveCounter value: %d " % (fiveCounter))
        while True:
            logger.debug("mode: %s" % vehicle.mode.name) #Stop action if
we are no longer in guided mode.
            while (vehicle.mode.name != "GUIDED"):
                vehicle.mode = VehicleMode("GUIDED")
                time.sleep(0.1)

            if fiveCounter == 1:
                vc_loc = vehicle.location.global_relative_frame
                vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.
lat,vc_loc.alt)
                fiveSecondCheck = get_distance_metres(vehicle_currentLoc
ation, targetLocation)
                logger.debug("fiveSecondCheck distance: %f " % (fiveSeco
ndCheck))
                logger.debug("fiveCounter value: %d " % (fiveCounter))

            if fiveCounter >=5:
                logger.debug("fiveSecondCheck distance: %f " % (fiveSeco
ndCheck))
                logger.debug("fiveCounter value: %d " % (fiveCounter))
                fiveCounter = 1
                vc_loc = vehicle.location.global_relative_frame
                vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.
lat,vc_loc.alt)
                currentDistanceToTarget = get_distance_metres(vehicle_cu
rrentLocation, targetLocation)
                logger.debug("fiveSecondCheck currentDistanceToTarget di
stance: %f " % (currentDistanceToTarget))
                if currentDistanceToTarget >= 0.9* fiveSecondCheck:
                    #resend the msg command to drone
                    vehicle.send_mavlink(msg)
                    logger.critical("Last command message dropped. Resen
ding the command message to drone")
                    logger.debug("Resend the command message to drone.")

            vc_loc = vehicle.location.global_relative_frame
            vehicle_currentLocation = LAT_LON_ALT(vc_loc.lon,vc_loc.lat,
vc_loc.alt)
            remainingDistance=get_distance_metres(vehicle_currentLocatio
n, targetLocation)
            logger.info("Distance to target: %f" % (remainingDistance))
            print("Distance to target: %f" % (remainingDistance))
            if remainingDistance <= 1: #Just below target, in case of un
dershoot.
                logger.info("Reached target")
                break
            fiveCounter += 1
            time.sleep(1)
```

5. print_vehicle_attributes():

This function list all the attributes of the vehicle and stores it in log file:

In [ ]:
```
def print_vehicle_attributes():
        logger.info("Autopilot Firmware version: %s" % vehicle.version)
        logger.info("Autopilot capabilities (supports ftp): %s" % vehicl
e.capabilities.ftp)
        logger.info("Global Location:%s" % vehicle.location.global_fram
e)
        logger.info("Global Location (relative altitude): %s" % vehicle.
location.global_relative_frame)
        logger.info("Local Location: %s" % vehicle.location.local_frame)
        logger.info("Attitude: %s" % vehicle.attitude)
        logger.info("Velocity: %s" % vehicle.velocity)
        logger.info("GPS: %s" % vehicle.gps_0)
        logger.info("Groundspeed: %s" % vehicle.groundspeed)
        logger.info("Airspeed: %s" % vehicle.airspeed)
        logger.info("Gimbal status: %s" % vehicle.gimbal)
        logger.info("Battery: %s" % vehicle.battery)
        logger.info("EKF OK?: %s" % vehicle.ekf_ok)
        logger.info("Last Heartbeat: %s" % vehicle.last_heartbeat)
        logger.info("Rangefinder: %s" % vehicle.rangefinder)
        logger.info("Rangefinder distance: %s" % vehicle.rangefinder.dis
tance)
        logger.info("Rangefinder voltage: %s" % vehicle.rangefinder.volt
age)
        logger.info("Heading: %s" % vehicle.heading)
        logger.info("Is Armable?: %s" % vehicle.is_armable)
        logger.info("System status: %s" % vehicle.system_status.state)
        logger.info("Mode: %s" % vehicle.mode.name)
        logger.info("Armed: %s" % vehicle.armed)
```

6. print_vehicle_parameters():

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

In [ ]:
```
def print_vehicle_parameters():
        logger.info ("Print all parameters (`vehicle.parameters`):")
        for key, value in vehicle.parameters.iteritems():
            logger.info (" Key:%s Value:%s" % (key,value))
```

7. 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)
                logger.debug("Next location lat: %f , lon: %f , alt: %
f",nextLocation.lat,nextLocation.lon,nextLocation.alt)
                goto(nextLocation)
                print("Dropping Seed")
                logger.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 latitut
e of starting point:\n"))
            logger.debug("USER entered latitute value: %s",str(startingL
ocation.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 longitu
de of starting point:\n"))
            logger.debug("USER entered longitude value: %s",str(starting
Location.lon))
            if(startingLocation.lon<0 or startingLocation.lon>180):
                print("Langitude value must be between 0 and 180")
                continue
            startingLocation.alt = float(input("Please enter the altitud
e for the drone:\n"))
            logger.debug("USER entered altitude value: %s",str(startingL
ocation.alt))
            if(startingLocation.alt<0):
                print("Altitude value must be positive")
                continue
            break
        except:
            logger.error("Oops!  That was no valid lat/lon or altitude.
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 correct f
ile")
            logger.error("Enter file does not exists.")
            continue
```

Set up option parsing to get connection string

In [ ]:
```
import argparse
parser = argparse.ArgumentParser(description='Demonstrates Seed Plantati
on Mission in GUIDED mode.')
parser.add_argument('--connect', help="vehicle connection target string.
If not specified, SITL automatically started and used.")
args = parser.parse_args()
connection_string = args.connect
sitl = None
```

Start SITL if no connection string specified

In [ ]:
```
if not connection_string:
        import dronekit_sitl
        sitl = dronekit_sitl.start_default(lat=startingLocation.lat,lon=
startingLocation.lon)
        connection_string = sitl.connection_string()
```

Connect to the Vehicle

In [ ]:
```
print('Connecting to vehicle on: %s' % connection_string)
logger.info('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True)
```

Log vehicle attributes:

In [ ]:
```
print_vehicle_attributes()
```

Log vehicle parameters:

In [ ]:
```
print_vehicle_parameters()
```

In [ ]:
```
print("Arm and Takeoff")
logger.info("Arm and Takeoff")
arm_and_takeoff(startingLocation.alt)
```

Start the mission by calling the startMission() function

After completion of mission RTL (Return to Launch)

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

startMission(startingLocation)


print('Return to launch')
logger.critical("Return to launch")
while (vehicle.mode.name != "RTL"):
        vehicle.mode = VehicleMode("RTL")
        time.sleep(0.1)
```

Close vehicle object before exiting script

In [ ]:
```
print("Close vehicle object")
logger.info("Close vehicle object")
vehicle.close()
```

Shut down simulator.

In [ ]:
```
if sitl is not None:
    sitl.stop()
print("Completed...")
logger.info("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. def get_location_metres(original_location, dNorth, dEast)

Returns a POINT object containing the latitude/longitude `dNorth` and `dEast` metres from the specified `original_location` . The function is useful when you want to move the vehicle around specifying locations relative to the current vehicle position. This function is relatively accurate over small distances (10m within 1km) except close to the poles.

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

In [1]:
```
def get_location_metres(original_location, dNorth, dEast):

        #Radius of "spherical" earth
        earth_radius=6378137.0

        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.la
t/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
        newlon = original_location.lon + (dLon * 180/math.pi)
        new_location = POINT(newlat,newlon)
        return new_location
```

2. def get_distance_metres(aLocation1, aLocation2)

Returns the ground distance in metres between two LAT_LON objects. This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

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

In [2]:
```
def get_distance_metres(aLocation1, aLocation2):

        dlat = aLocation2.lat - aLocation1.lat
        dlong = aLocation2.lon - aLocation1.lon
        return math.sqrt((dlat*dlat) + (dlong*dlong)) * 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 corner points of square and adding first point to file.

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

        f = open("waypoint_square.txt","w+")
        point1 = get_location_metres(aLocation, -aSize, -aSize)
        point2 = get_location_metres(aLocation, aSize, -aSize)
        point3 = get_location_metres(aLocation, aSize, aSize)
        point4 = get_location_metres(aLocation, -aSize, aSize)
        temp1 = point1
        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 = get_location_metres(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 = get_location_metres(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 = get_location_metres(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 = get_location_metres(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 = get_location_metres(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. def get_location_metres(original_location, dNorth, dEast)

Returns a LAT_LON object containing the latitude/longitude `dNorth` and `dEast` metres from the specified `original_location` . The function is useful when you want to move the vehicle around specifying locations relative to the current vehicle position. This function is relatively accurate over small distances (10m within 1km) except close to the poles.

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

In [1]:
```
def get_location_metres(original_location, dNorth, dEast):

        #Radius of "spherical" earth
        earth_radius=6378137.0

        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.la
t/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
        newlon = original_location.lon + (dLon * 180/math.pi)
        new_location = LAT_LON(newlon,newlat)
        return new_location
```

2. def get_distance_metres(aLocation1, aLocation2)

Returns the ground distance in metres between two LAT_LON objects. This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

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

In [2]:
```
def get_distance_metres(aLocation1, aLocation2):

        dlat = aLocation2.lat - aLocation1.lat
        dlong = aLocation2.lon - aLocation1.lon
        return math.sqrt((dlat*dlat) + (dlong*dlong)) * 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 i in range (step_size/2):
                func_tempVar2 = func_tempVar1
                for j in range(step_size):
                        func_newpoint = get_location_metres(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 = get_location_metres(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 = get_location_metres(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 = get_location_metres(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 = get_location_metres(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(get_distance_metres(t1,t2),get_distance_metres(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. def get_location_metres(original_location, dNorth, dEast)

Returns a POINT object containing the latitude/longitude dNorth and dEast metres from the specified original_location. The function is useful when you want to move the vehicle around specifying locations relative to the current vehicle position. This function is relatively accurate over small distances (10m within 1km) except close to the poles.

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

In [3]:
```
def get_location_metres(original_location, dNorth, dEast):

        #Radius of "spherical" earth
        earth_radius=6378137.0

        #Coordinate offsets in radians
        dLat = dNorth/earth_radius
        dLon = dEast/(earth_radius*math.cos(math.pi*original_location.la
t/180))

        #New position in decimal degrees
        newlat = original_location.lat + (dLat * 180/math.pi)
        newlon = original_location.lon + (dLon * 180/math.pi)
        new_location = POINT(newlat,newlon)
        return new_location
```

2. def get_distance_metres(aLocation1, aLocation2)

Returns the ground distance in metres between two POINT class objects. This method is an approximation, and will not be accurate over large distances and close to the earth's poles.

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

In [4]:
```
def get_distance_metres(aLocation1, aLocation2):

        dlat = aLocation2.lat - aLocation1.lat
        dlong = aLocation2.lon - aLocation1.lon
        return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

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 = get_location_metres(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.

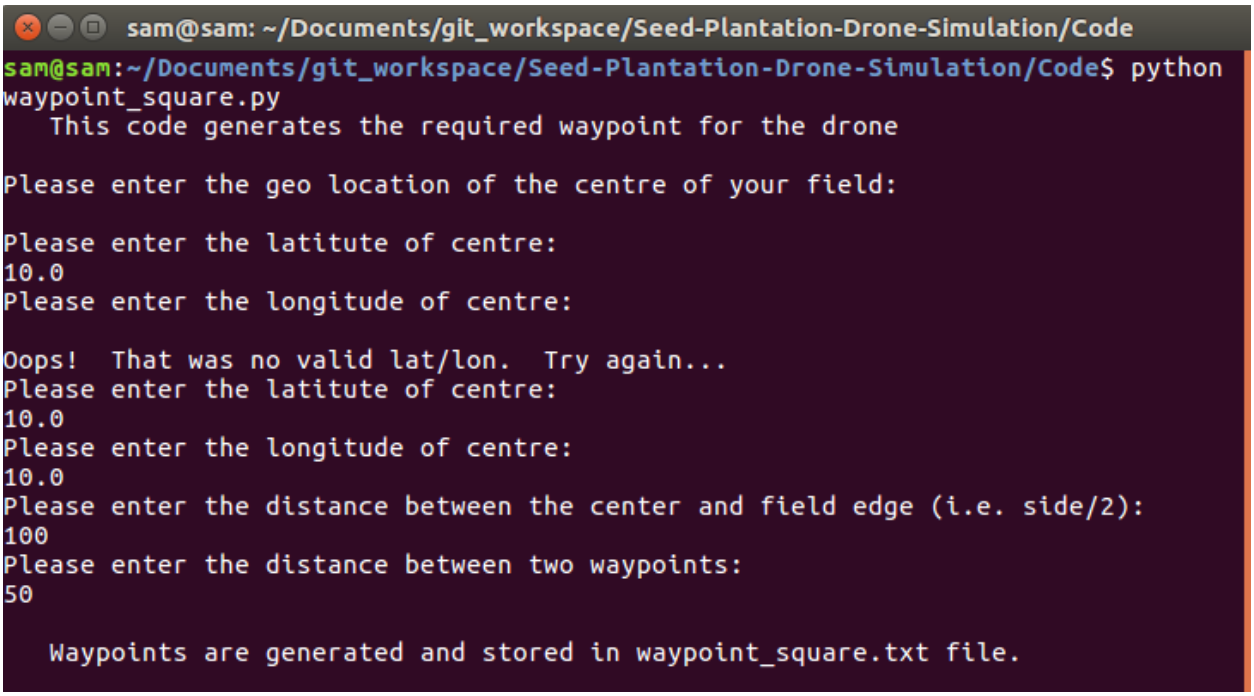
Helps in better understanding of how the code works with different types of inputs.

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

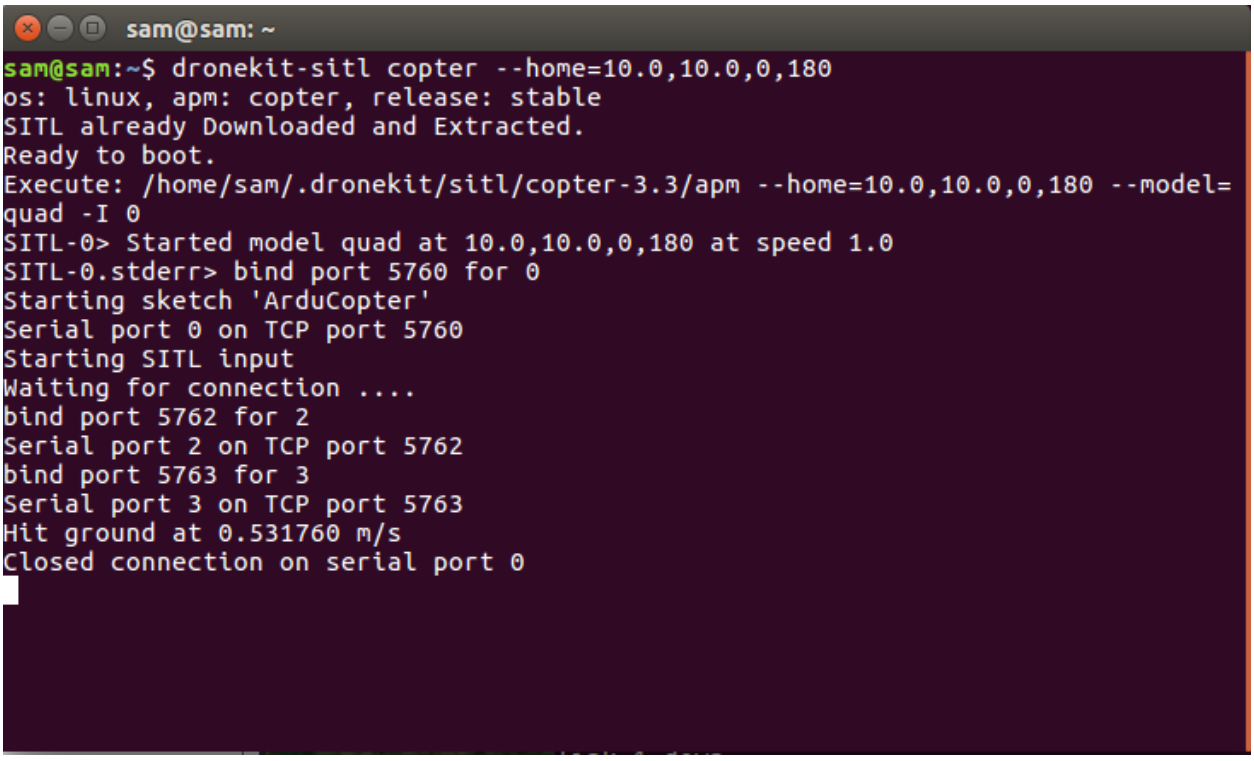
8.1.1 Square waypoint generator terminal

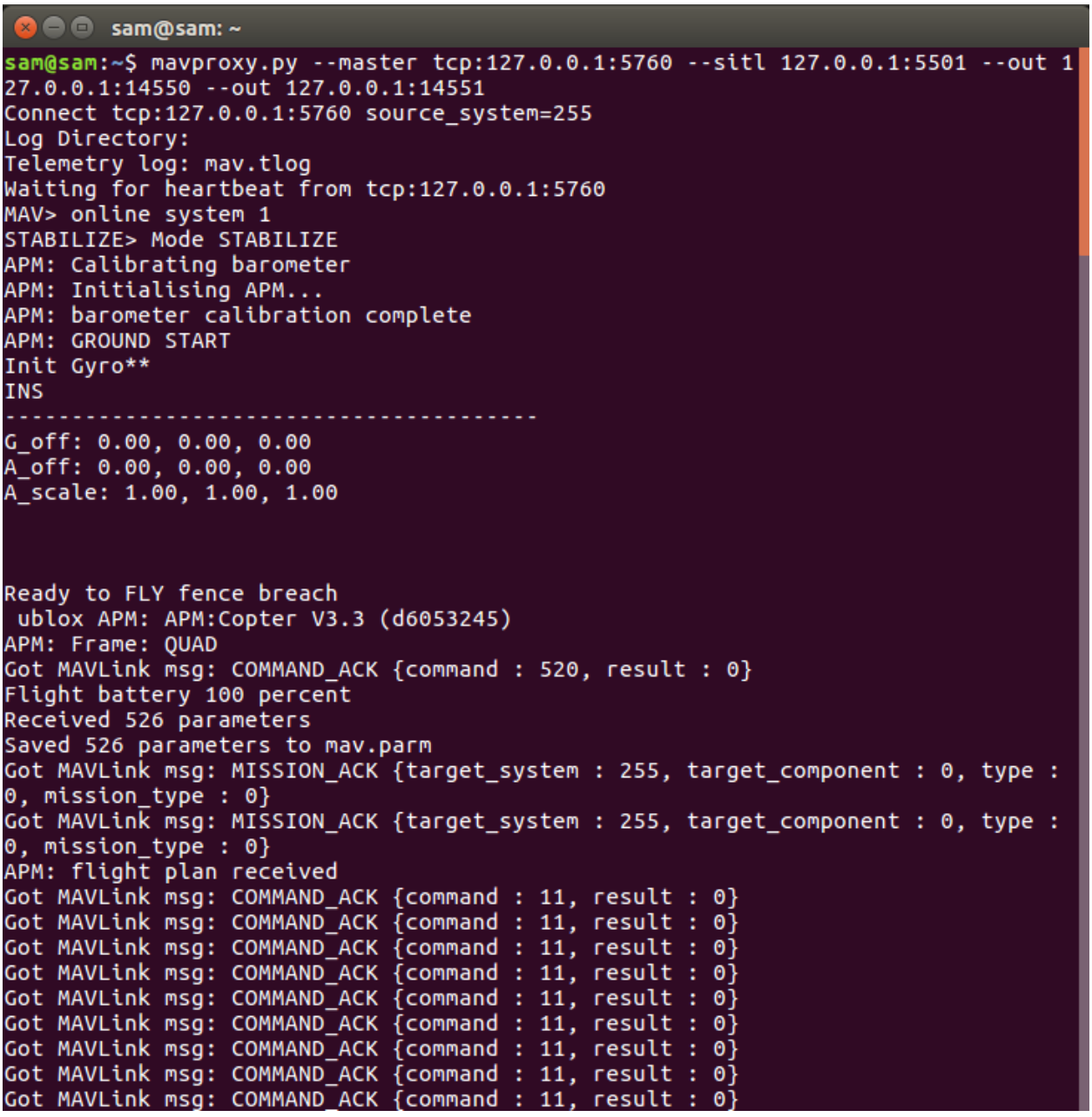


8.1.2 Waypoint square text file

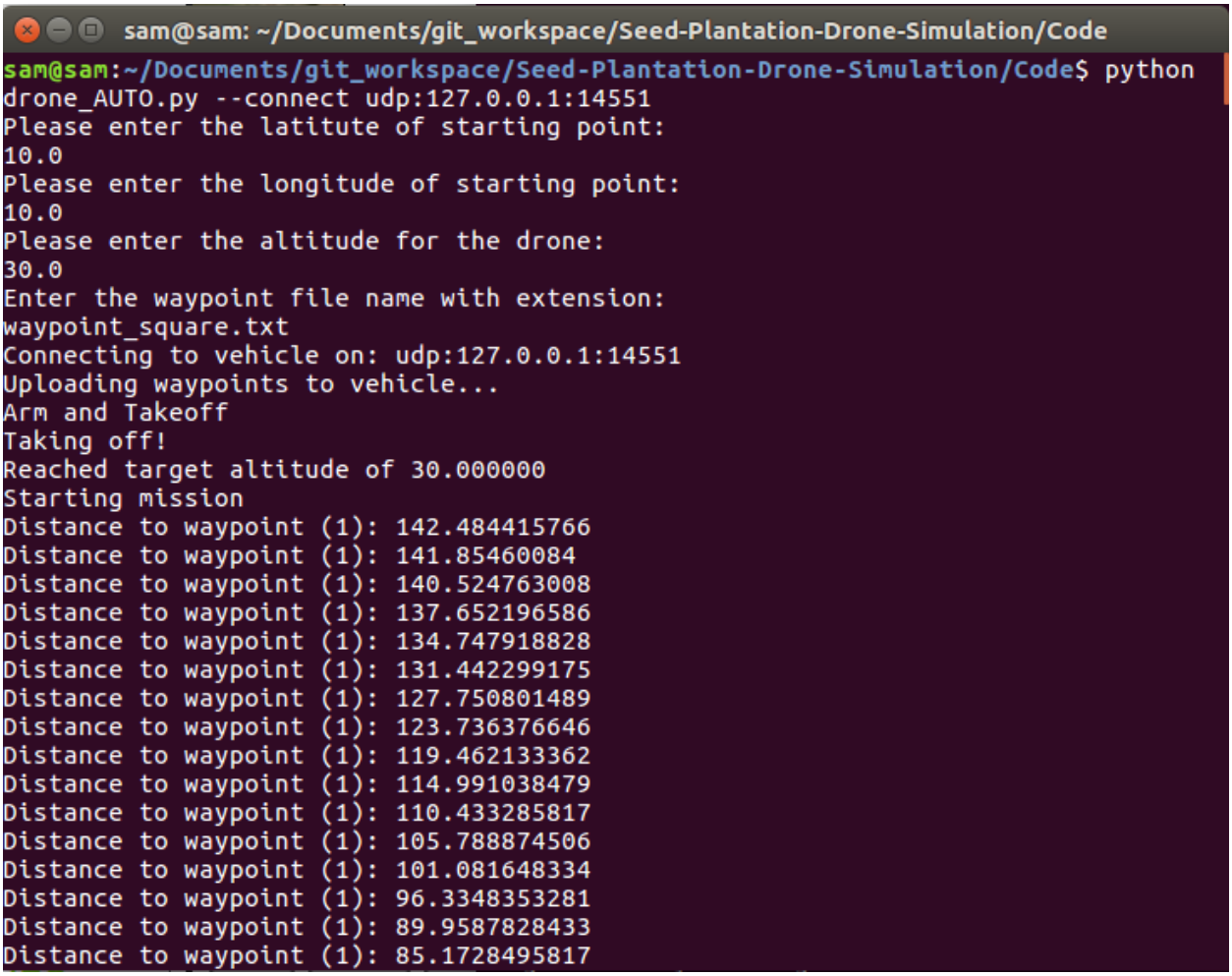
```
9.99910168472,9.99908782675
9.99955084236,9.99908782675
10.0,9.99908782675
10.0004491576,9.99908782675
10.0008983153,9.99908782675
10.0008983153,9.99954391464
10.0004491576,9.99954391464
10.0,9.99954391464
9.99955084236,9.99954391464
9.99910168472,9.99954391464
9.99910168472,10.0
9.99955084236,10.0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

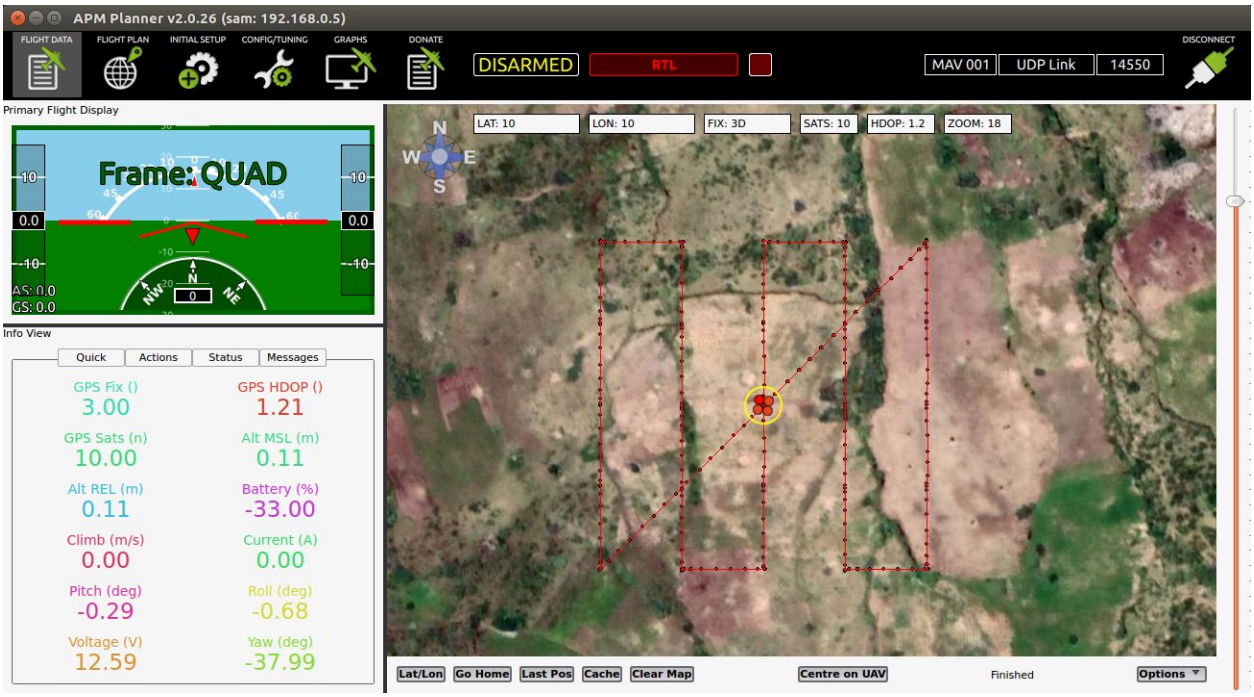



sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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

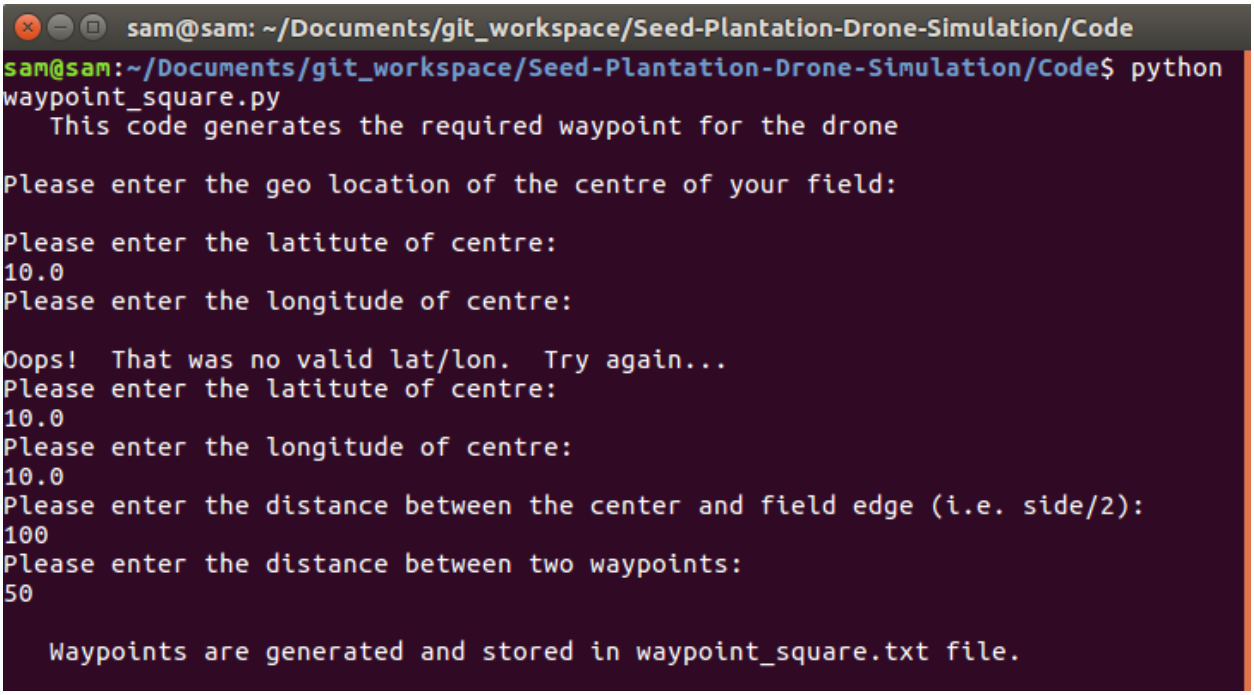
```
DEBUG: drone_AUTO.py: <module>: 141:            USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:            USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:            USER entered altitude value: 30.0
INFO: drone_AUTO.py: <module>: 187:             Connecting 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

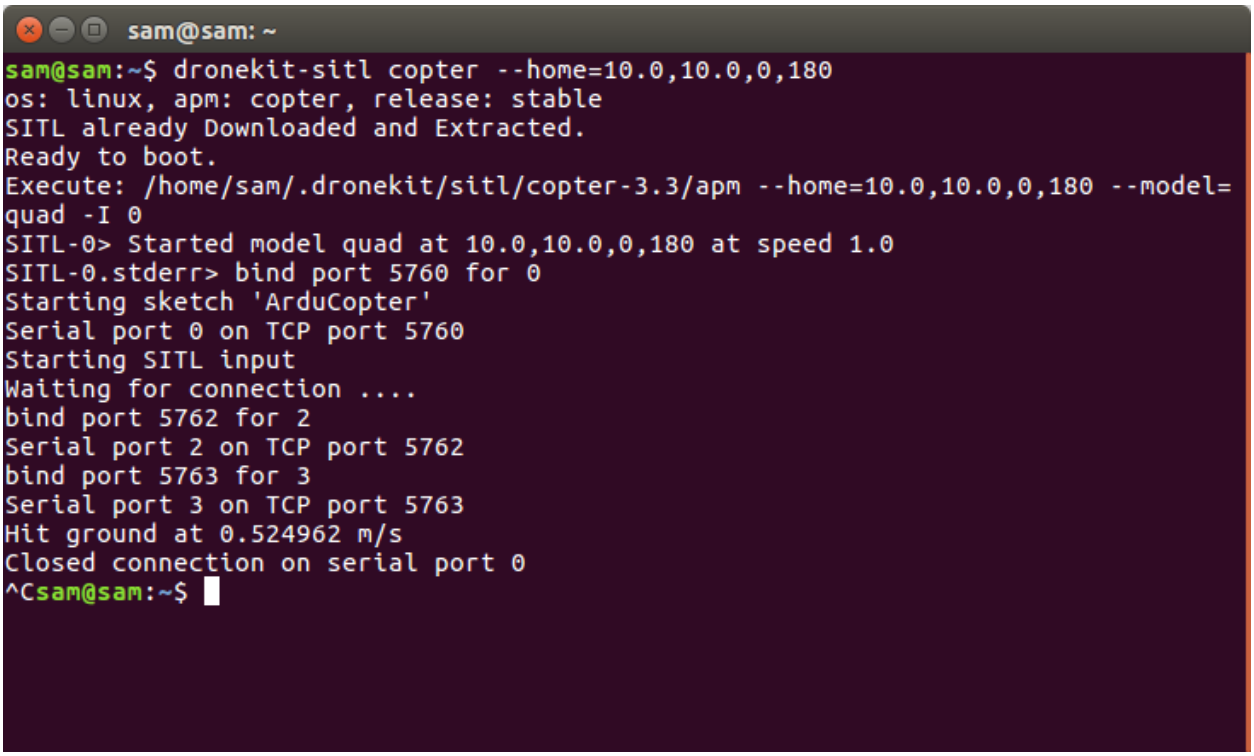
8.2.1 Square waypoint generator terminal

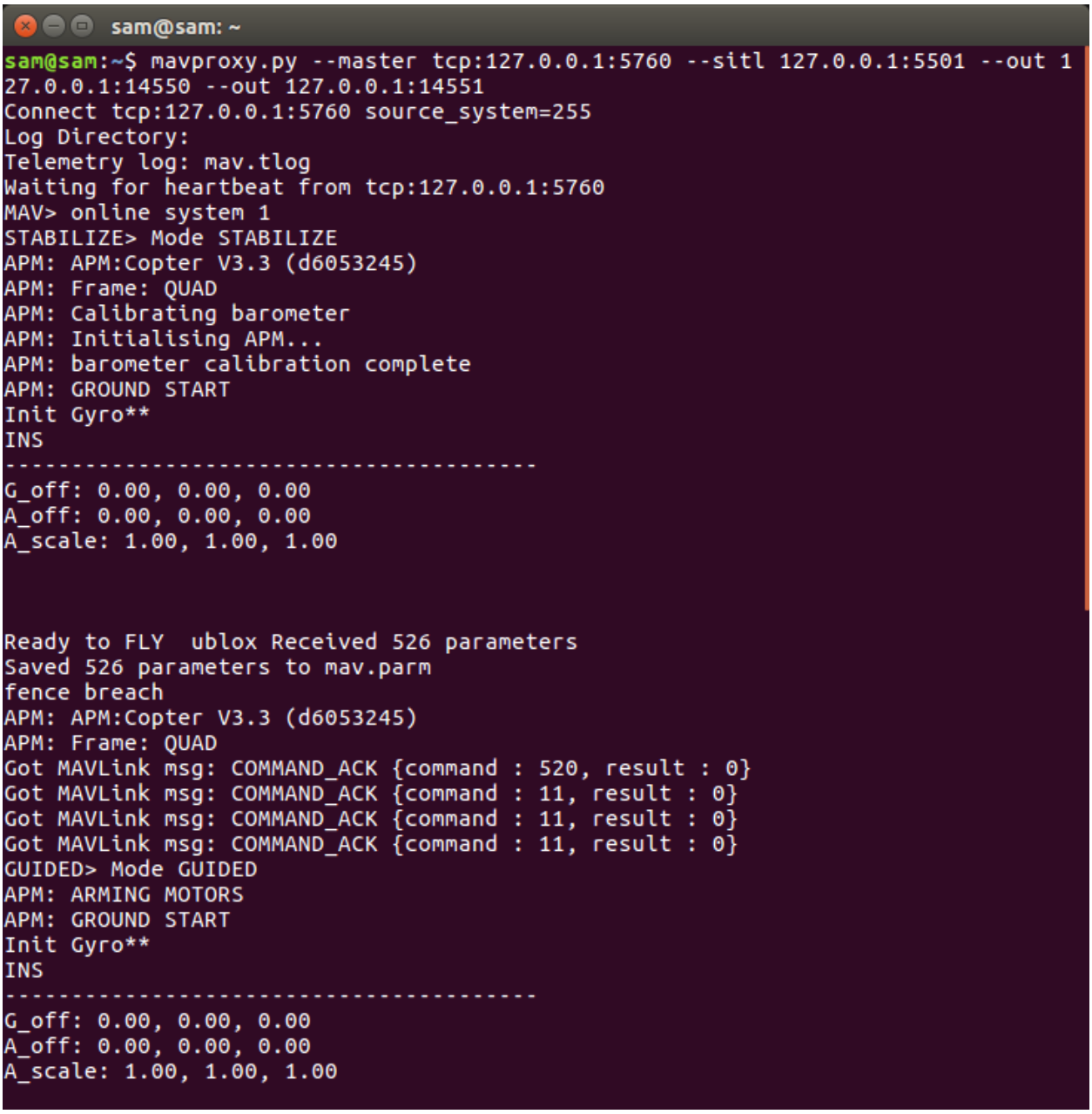


8.2.2 Waypoint square text file

```
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
```

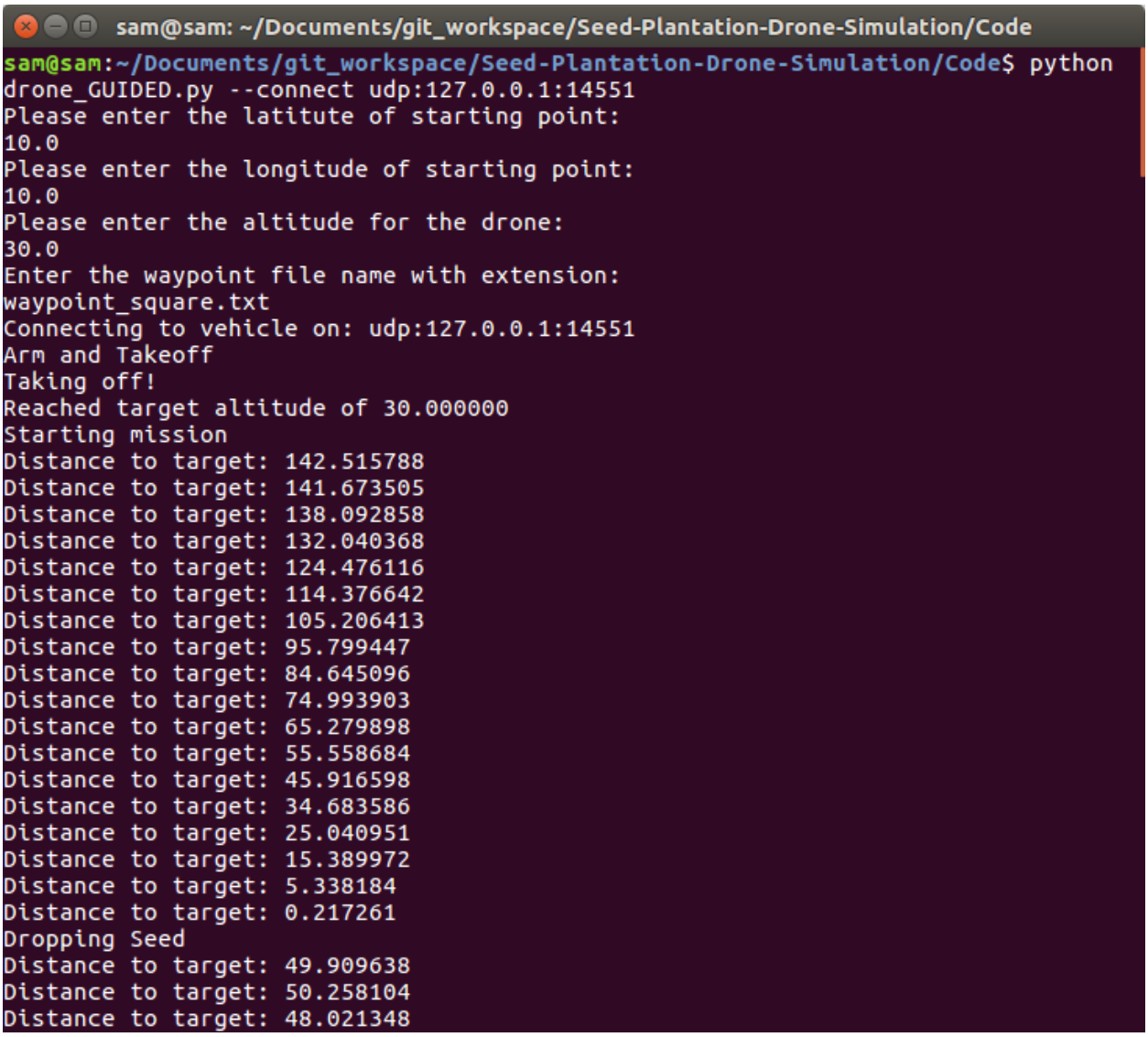
8.2.3 Terminal outputs



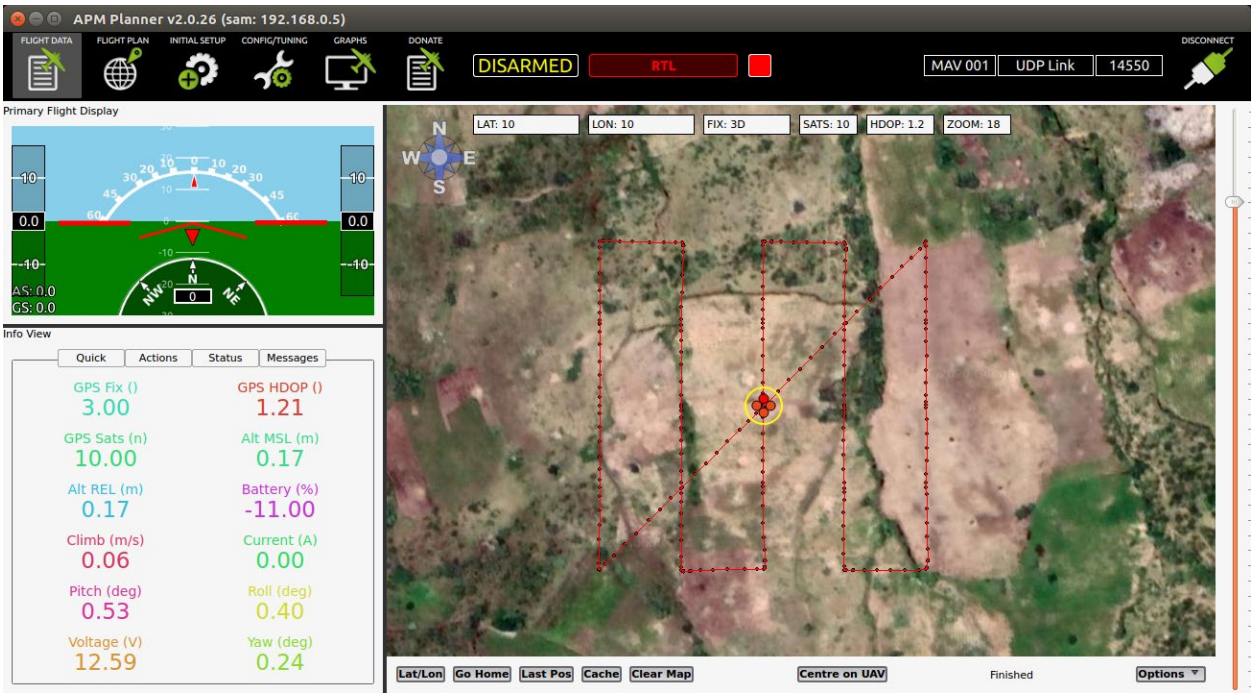

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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

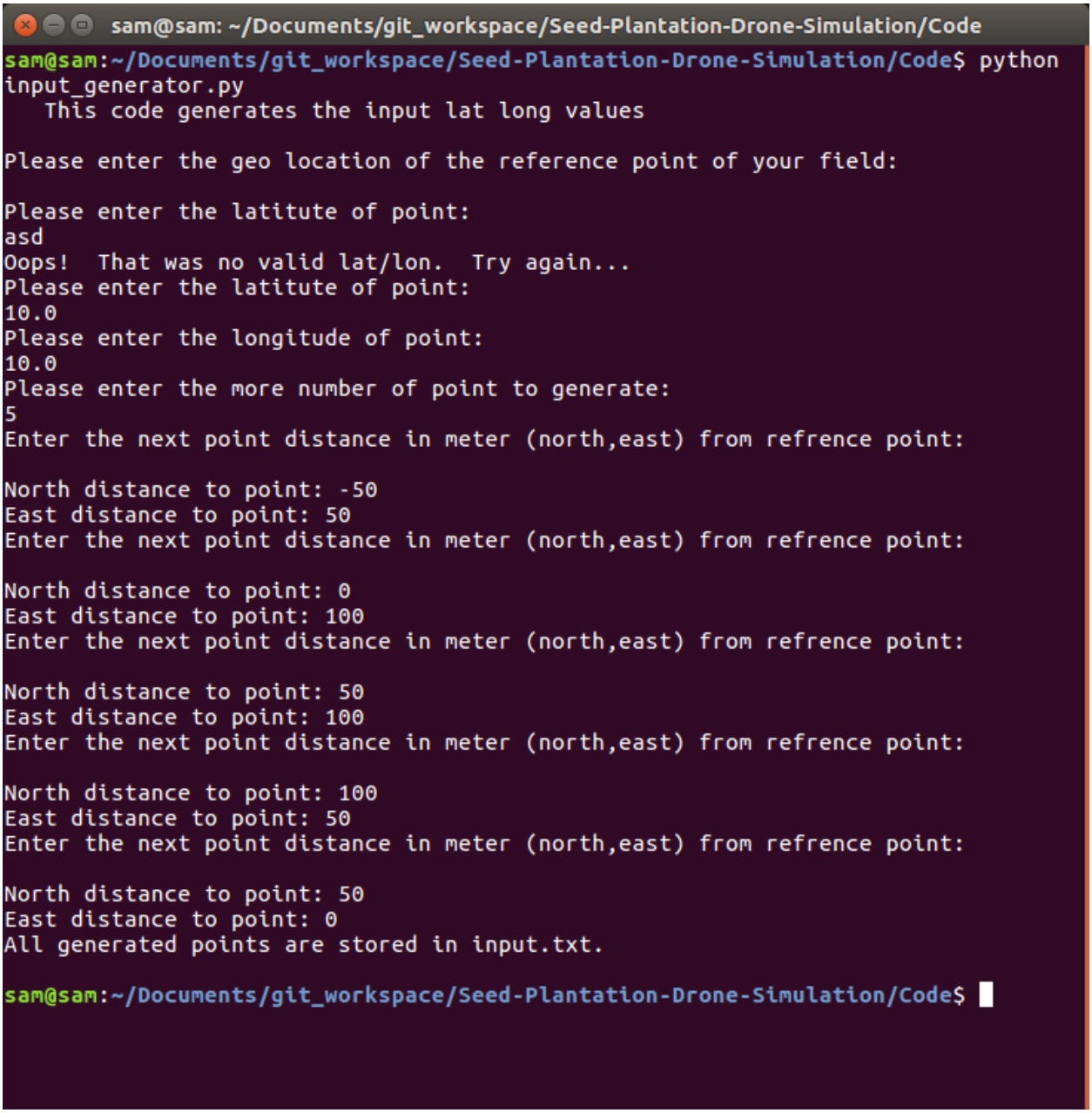
```
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 216:          USER entered latitute value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 221:          USER entered longitude value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 226:          USER entered altitude value: 30.0
INFO: drone_GUIDED.py: 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

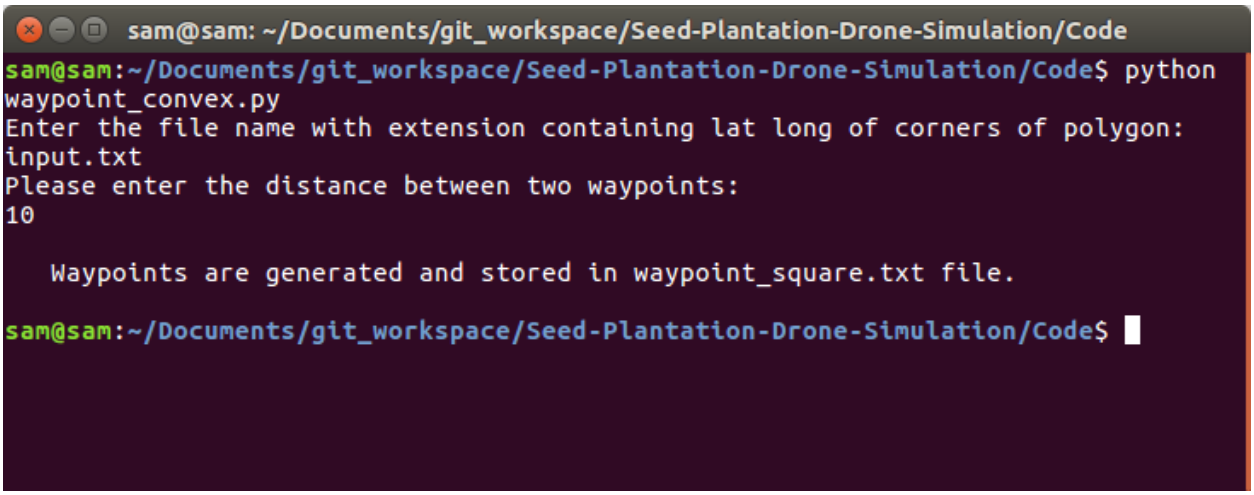


8.3.2 Input Text file

```
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

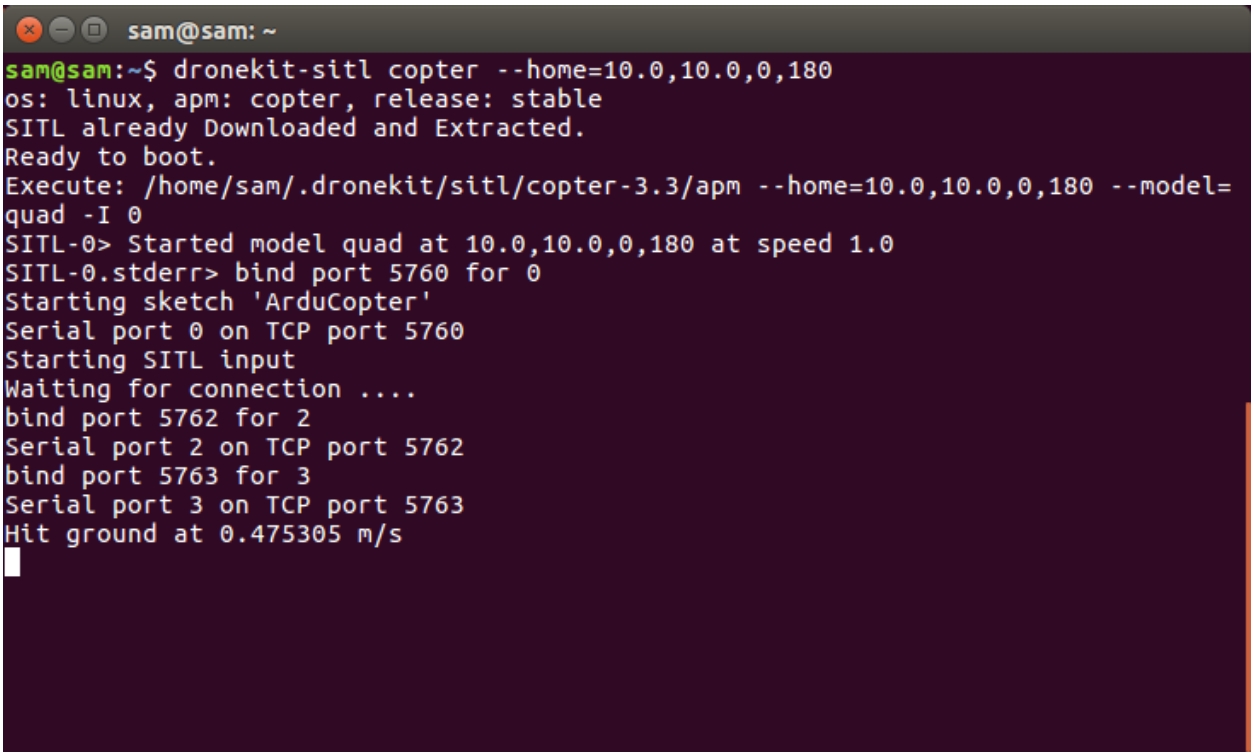


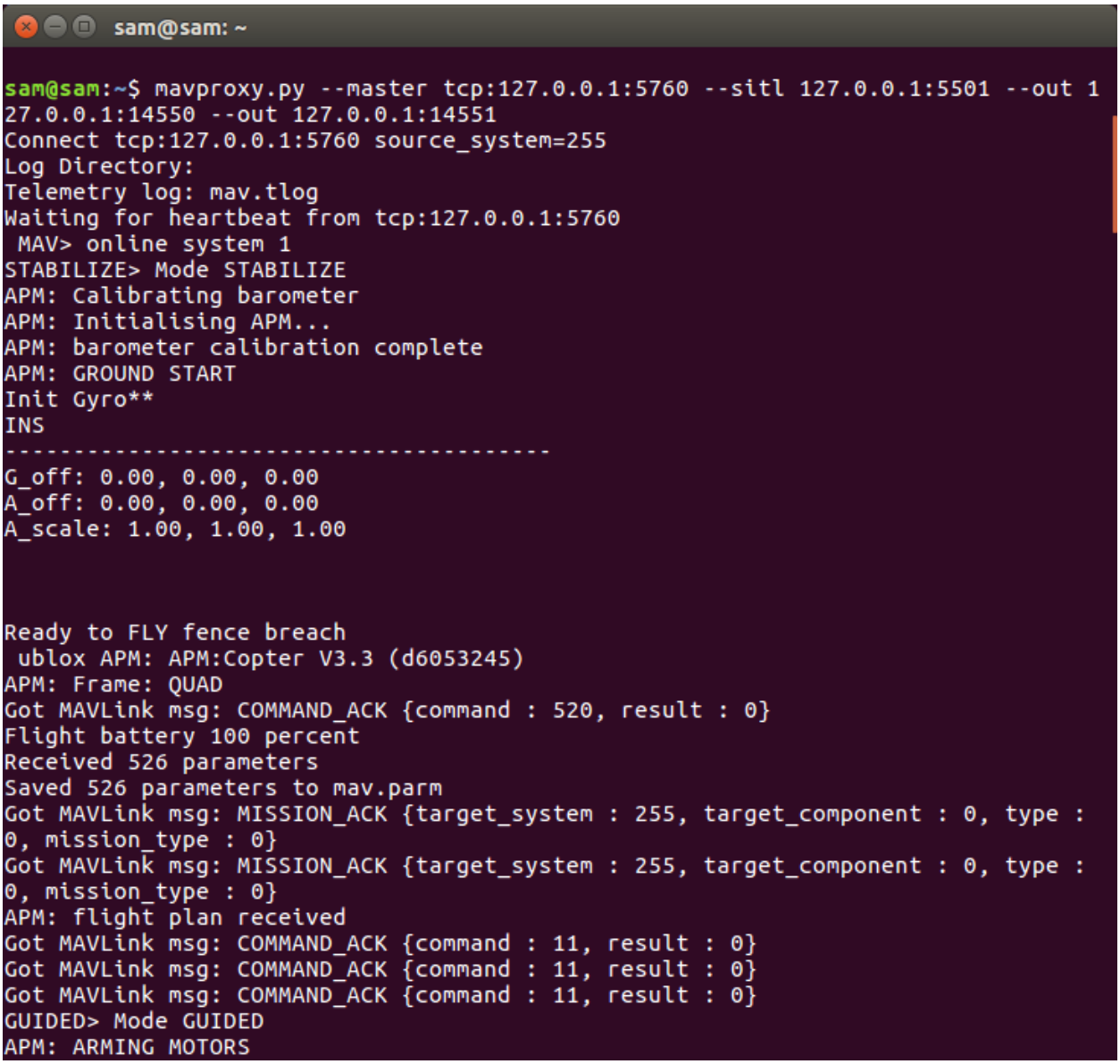
8.3.5 Waypoint hexagon text file

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



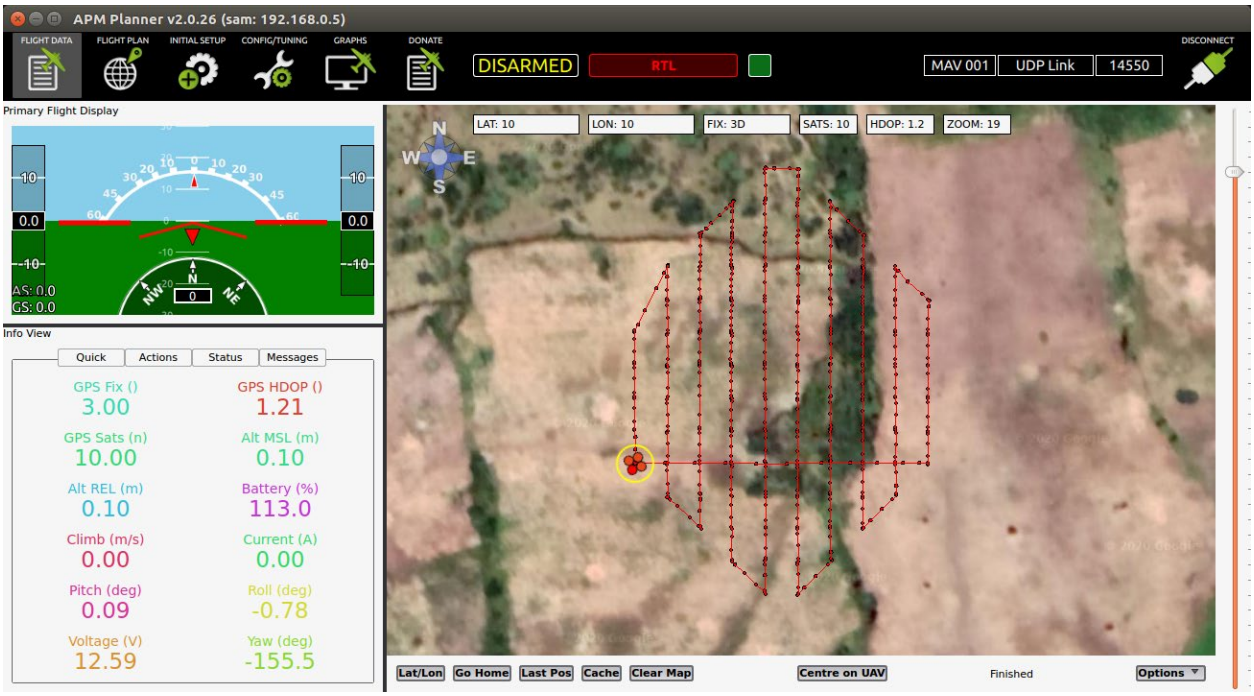

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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

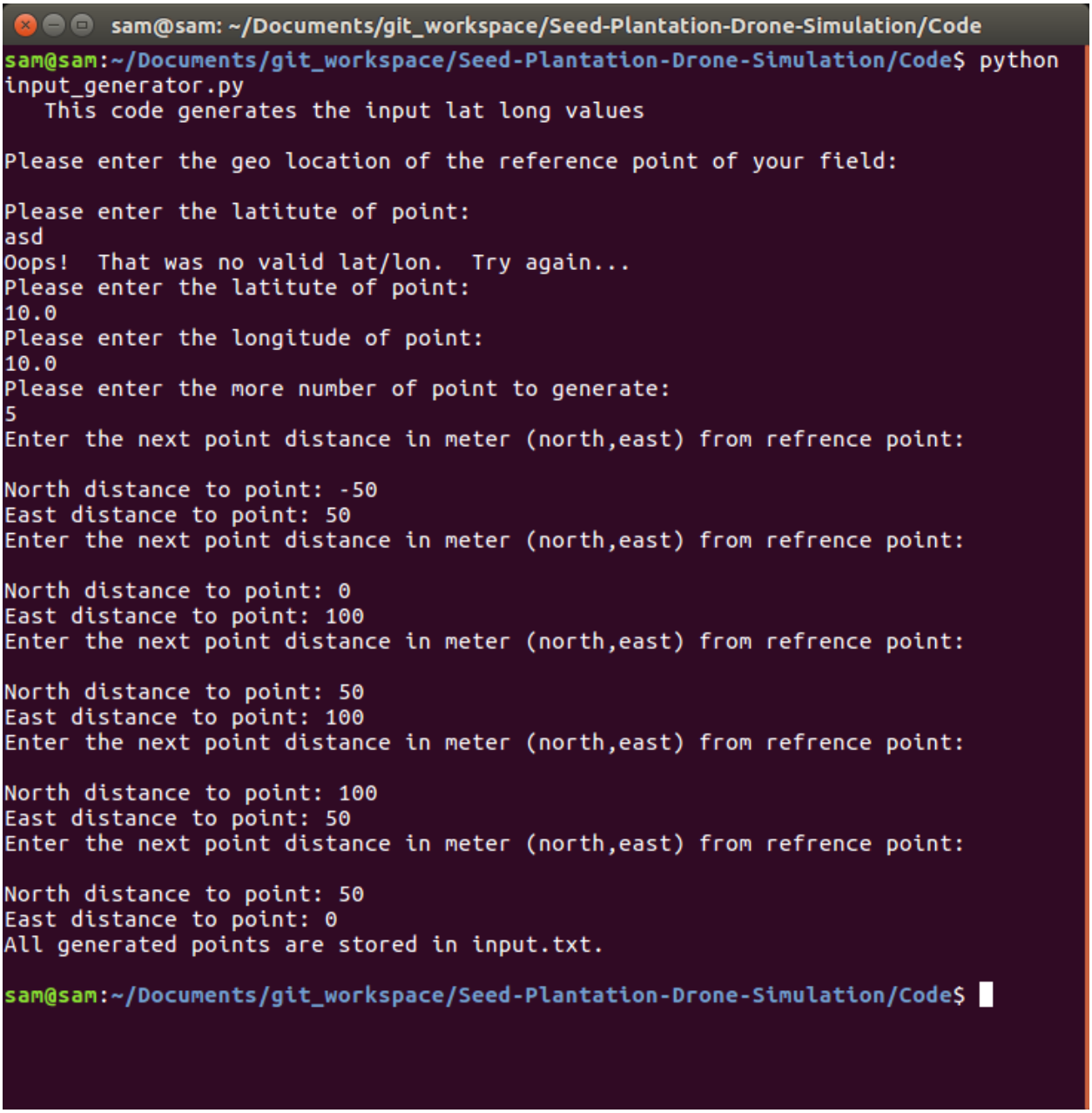
```
DEBUG: drone_AUTO.py: <module>: 141:            USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:            USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:            USER entered altitude value: 15.0
INFO: drone_AUTO.py: <module>: 187:             Connecting 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

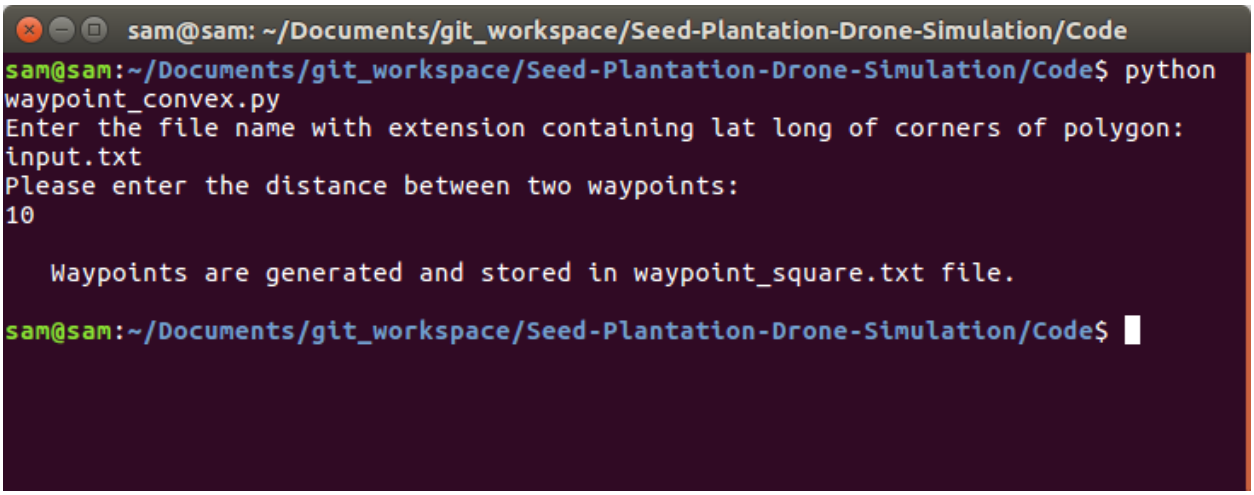


8.4.2 Input Text file

```
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

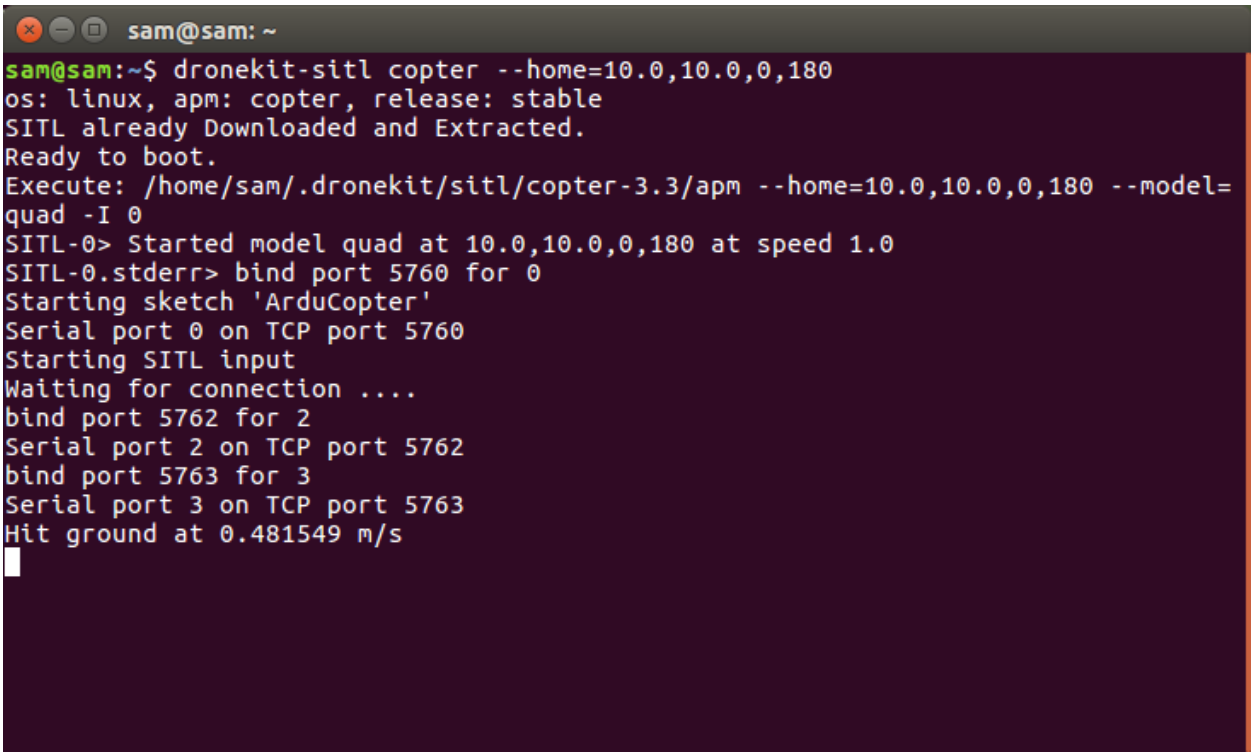


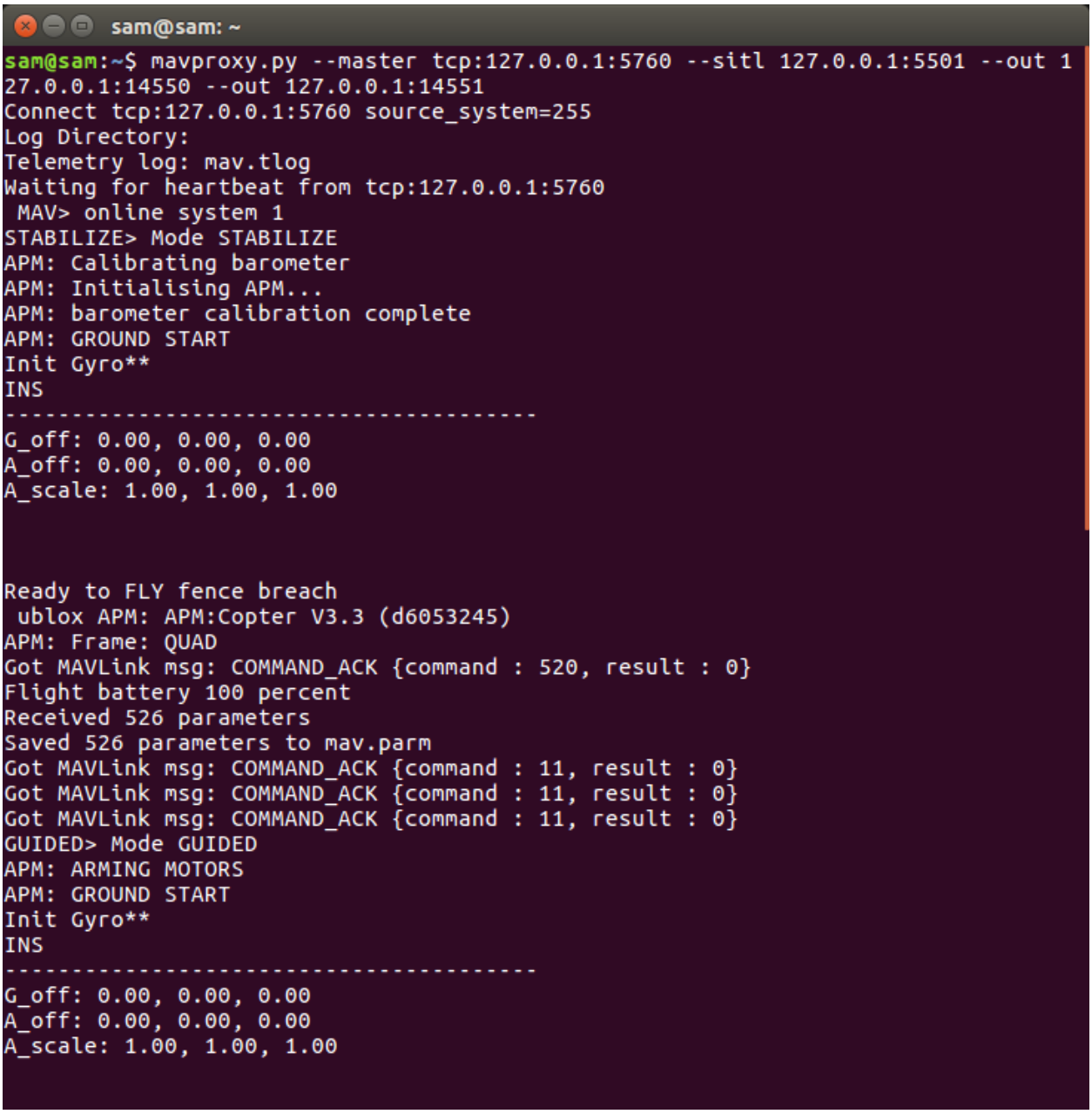
8.4.5 Waypoint hexagon text file

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
```

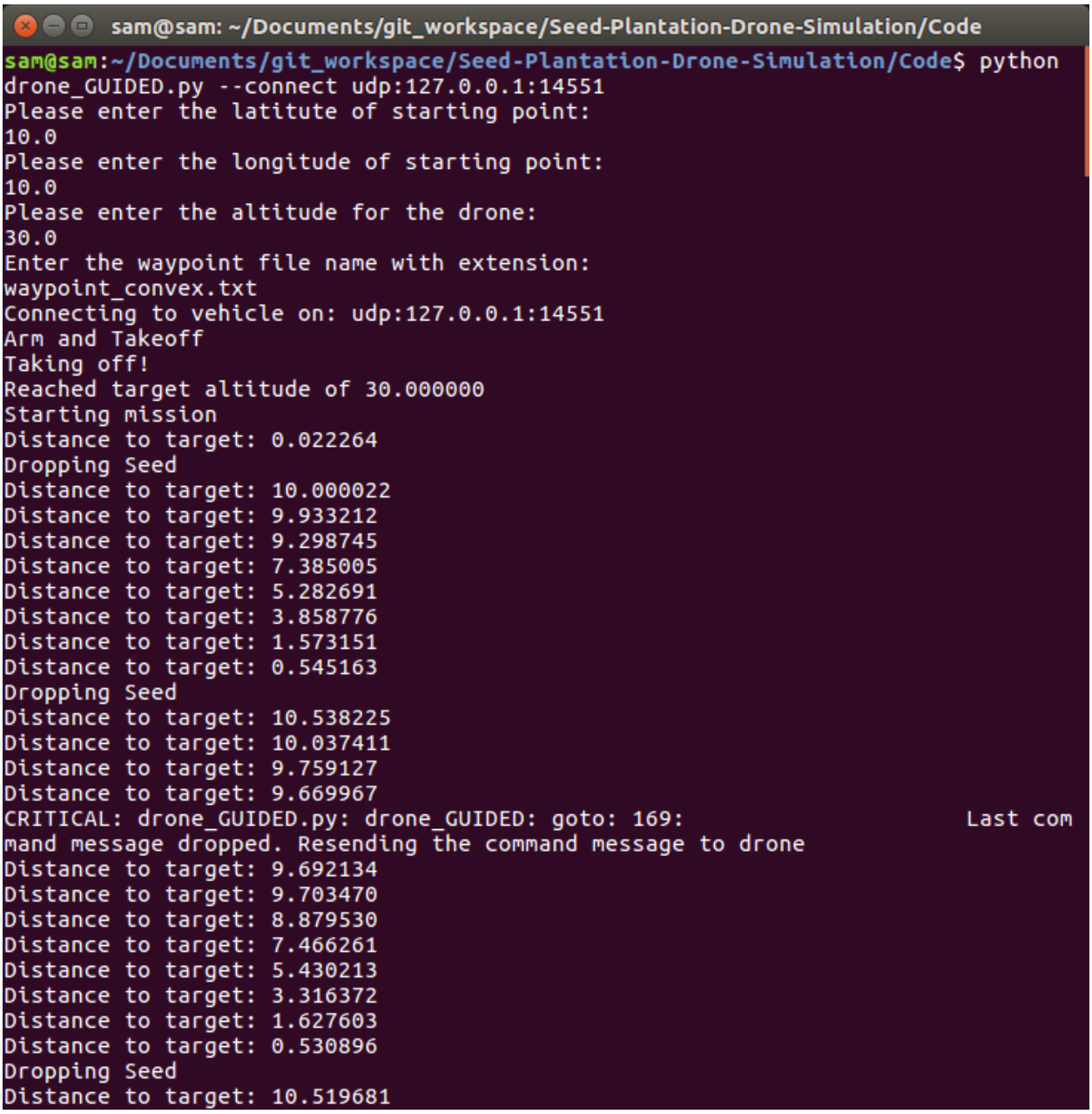
8.4.6 Terminal outputs



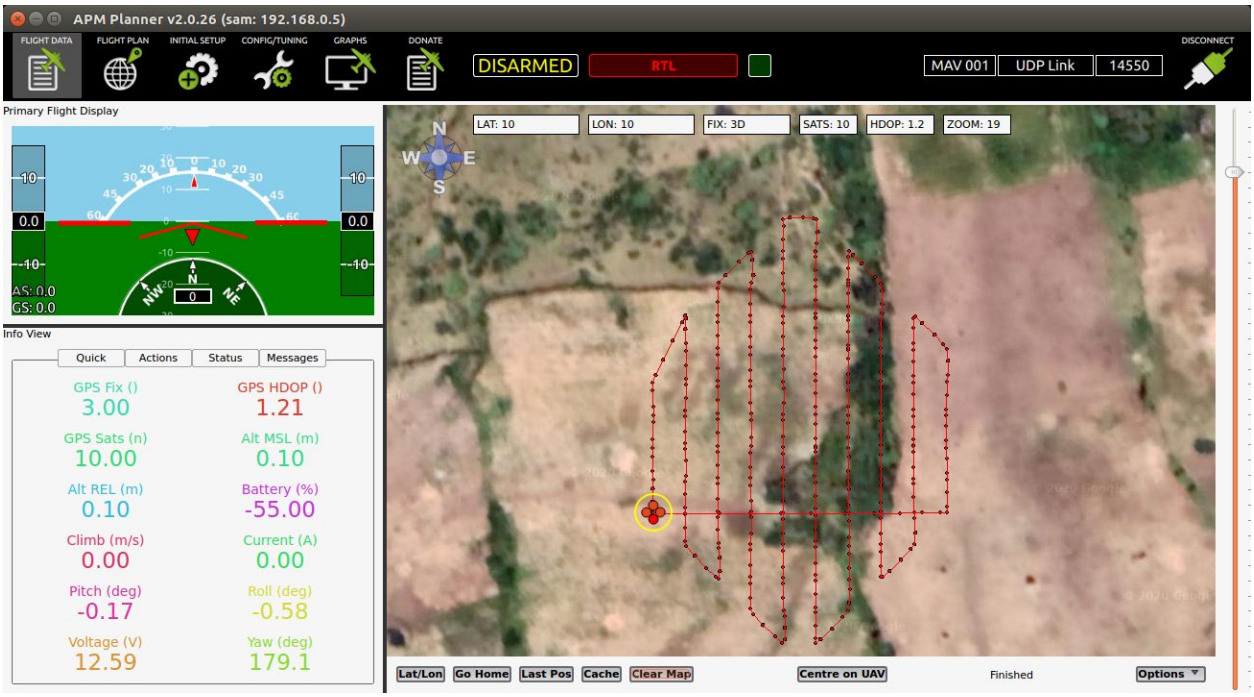

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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

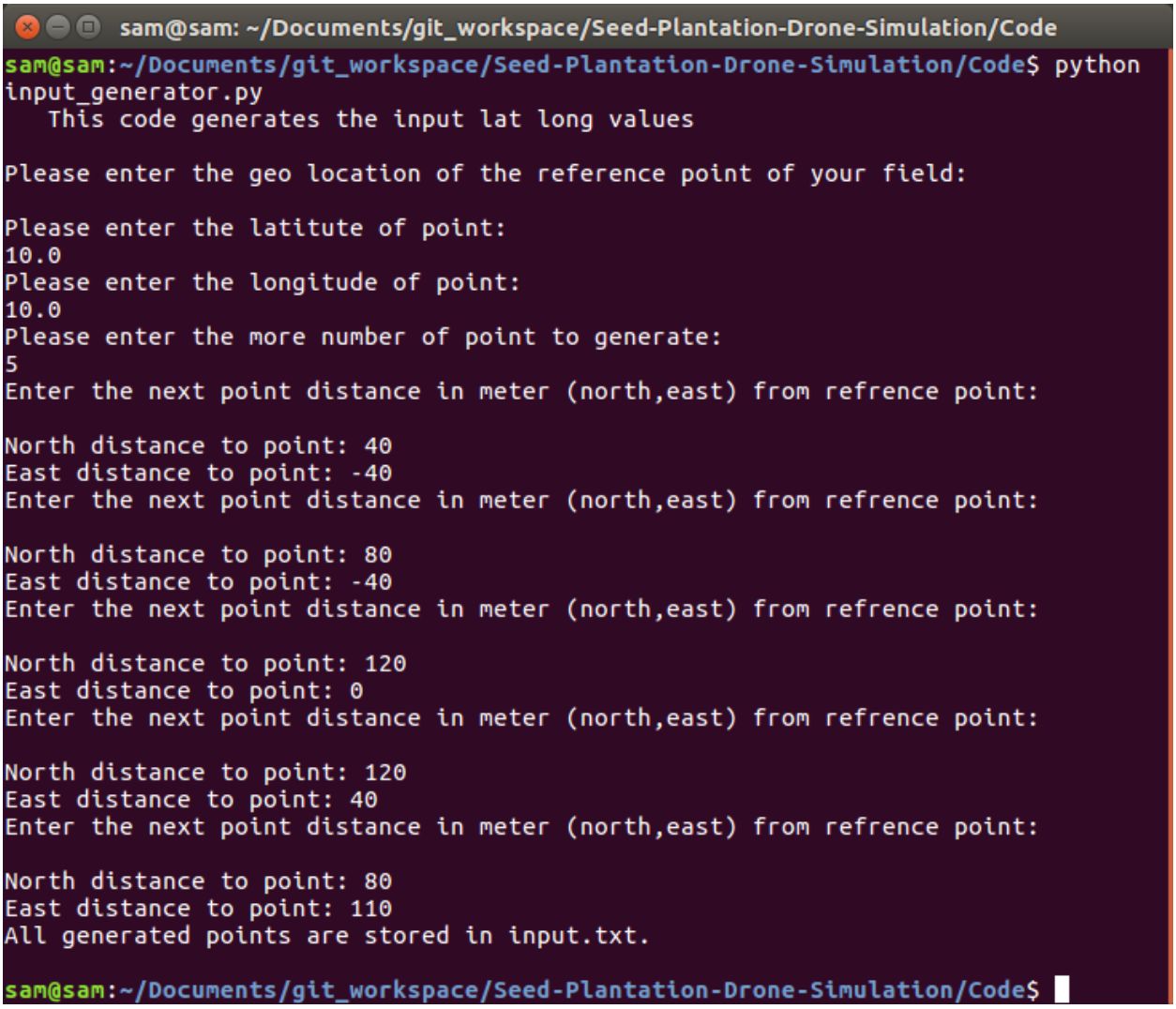
```
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 245:		USER entered latitute value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 250:		USER entered longitude value: 10.0
DEBUG: drone_GUIDED.py: drone_GUIDED: <module>: 255:		USER entered altitude value: 30.0
INFO: drone_GUIDED.py: drone_GUIDED: <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

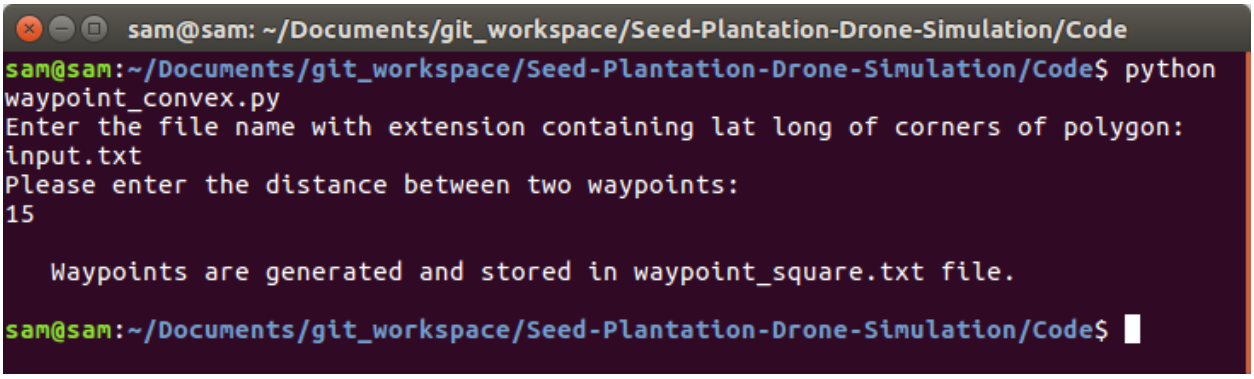
8.5.1 Input Generator file



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

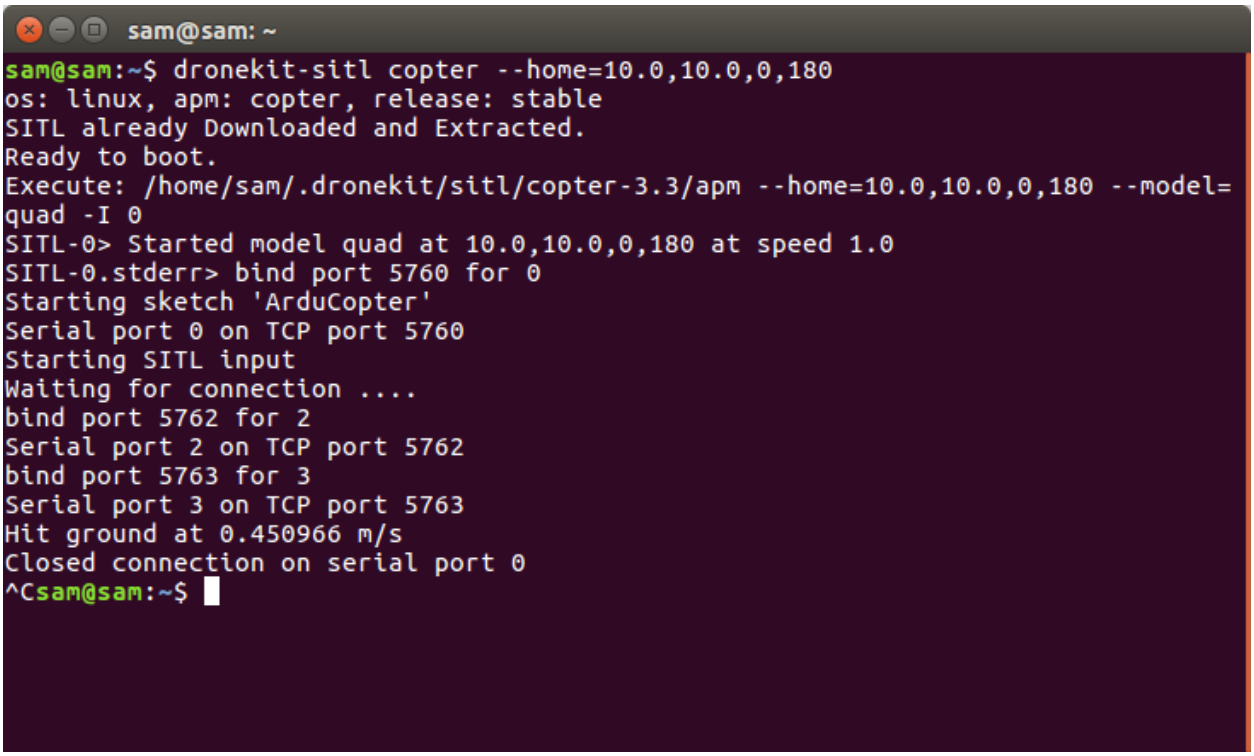


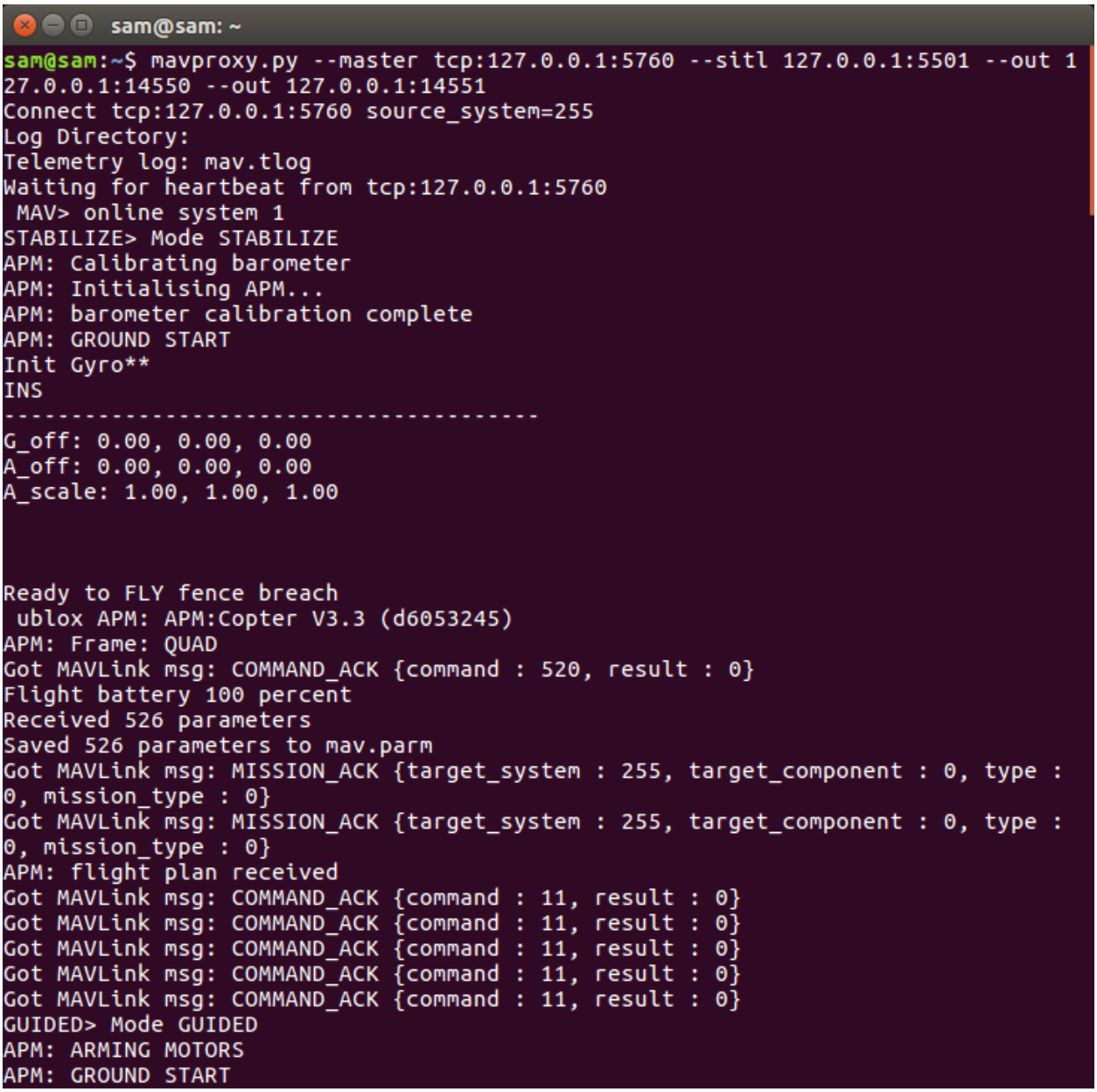
8.5.4 Waypoint convex text file

48 lines (48 sloc) | 1.28 KB

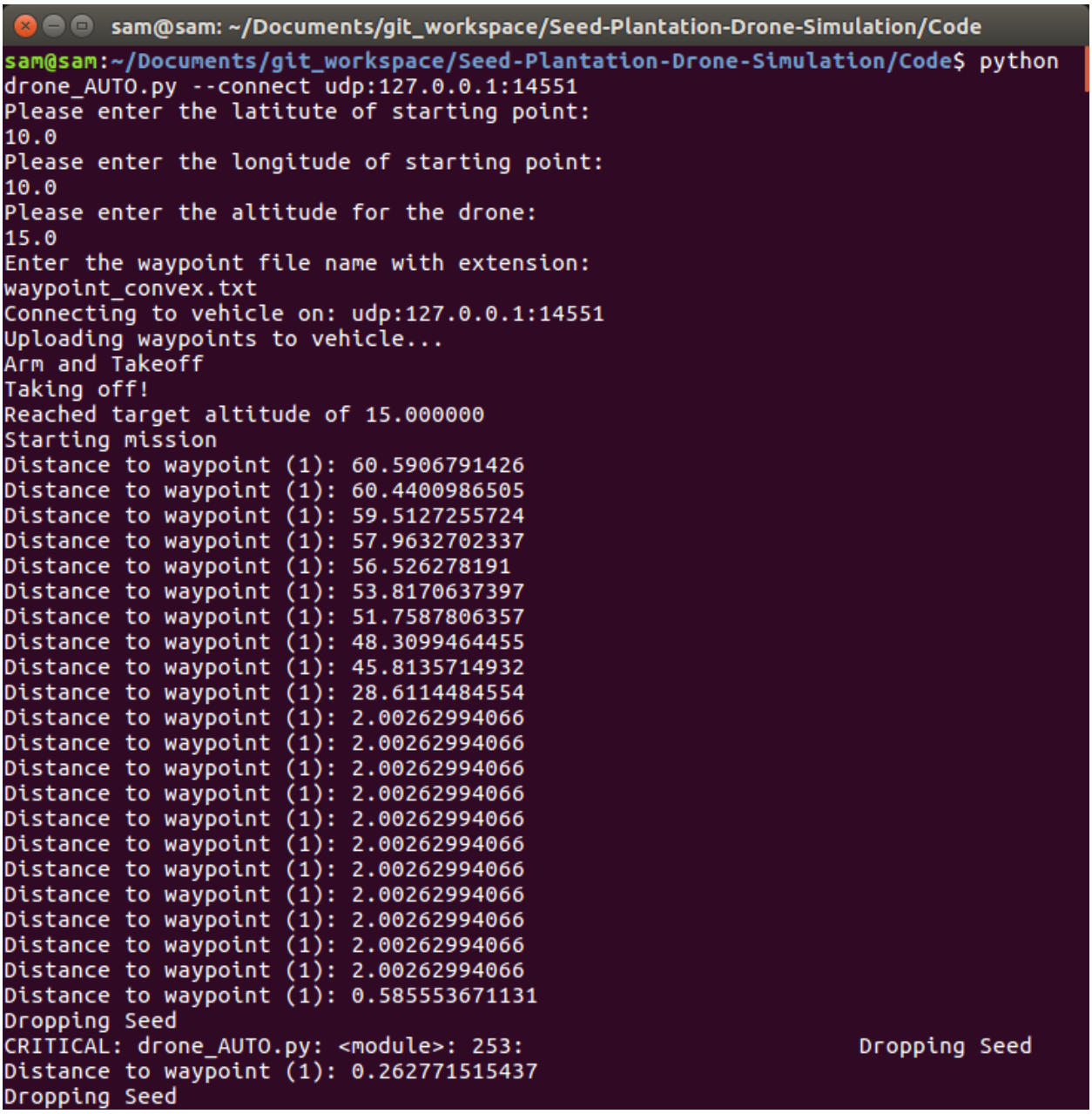
```
10.0004042419,9.9996351307
10.0005389892,9.9996351307
10.0006737365,9.9996351307
10.0008084838,9.99977195737
10.0006737365,9.99977195737
10.0005389892,9.99977195737
10.0004042419,9.99977195737
10.0002694946,9.99977195737
10.0001347473,9.99990878335
10.0002694946,9.99990878335
10.0004042419,9.99990878335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

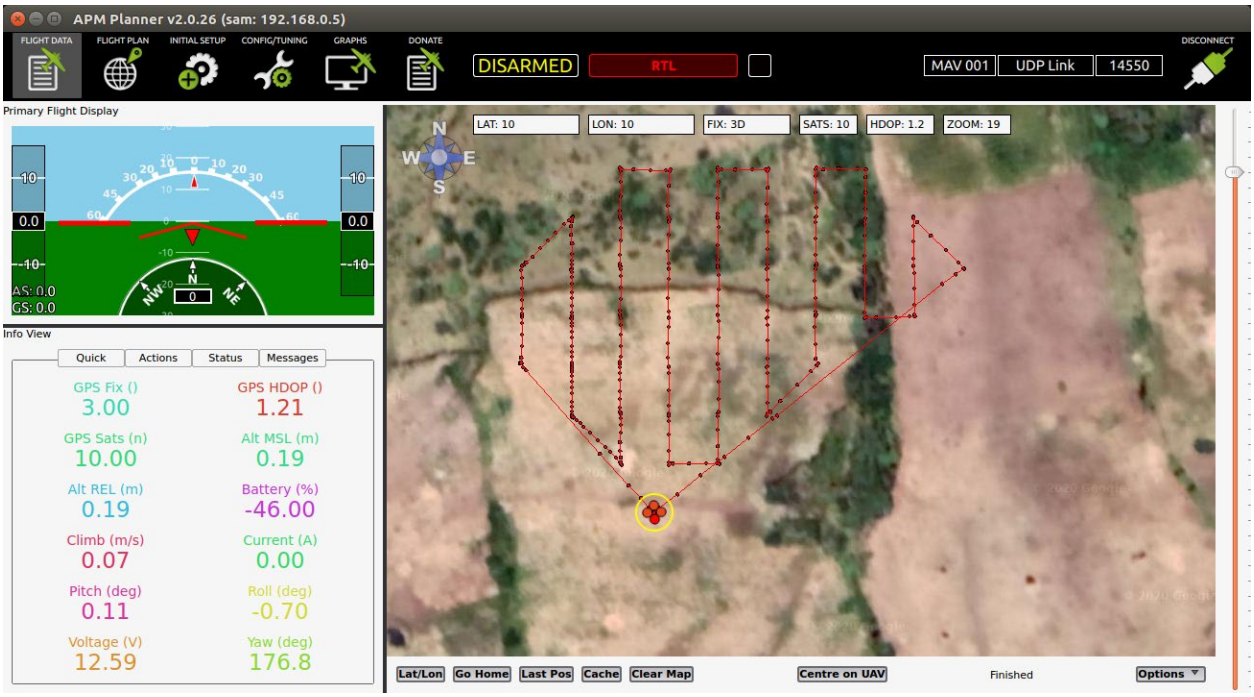



sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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

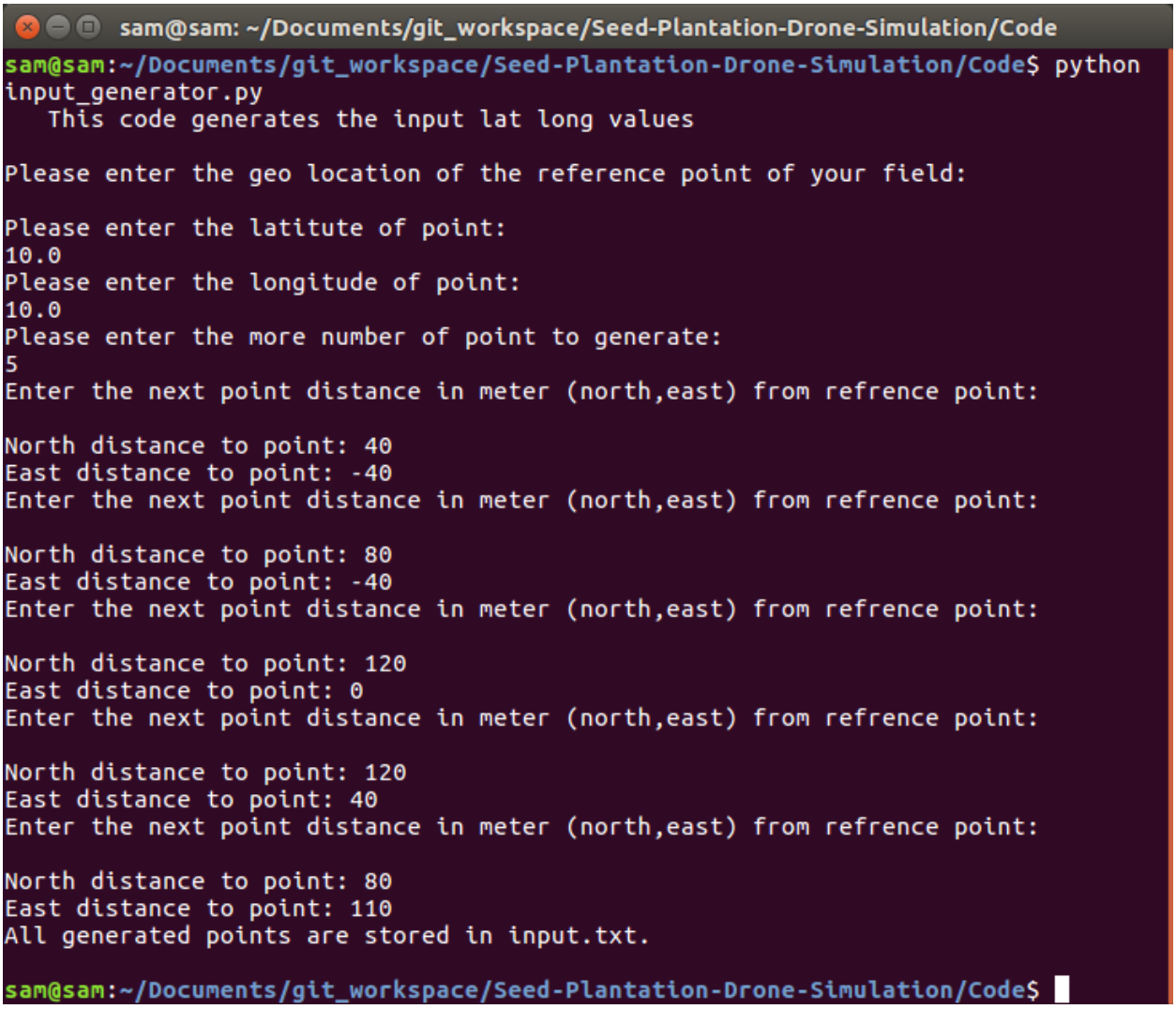
```
DEBUG: drone_AUTO.py: <module>: 141:                USER entered latitute value: 10.0
DEBUG: drone_AUTO.py: <module>: 146:                USER entered longitude value: 10.0
DEBUG: drone_AUTO.py: <module>: 151:                USER entered altitude value: 15.0
INFO: drone_AUTO.py: <module>: 187:                 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

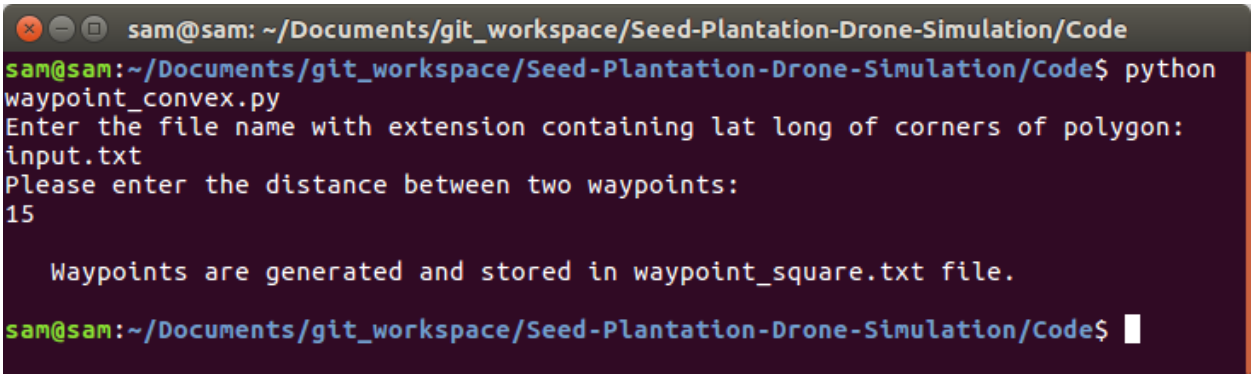
8.6.1 Input Generator file



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

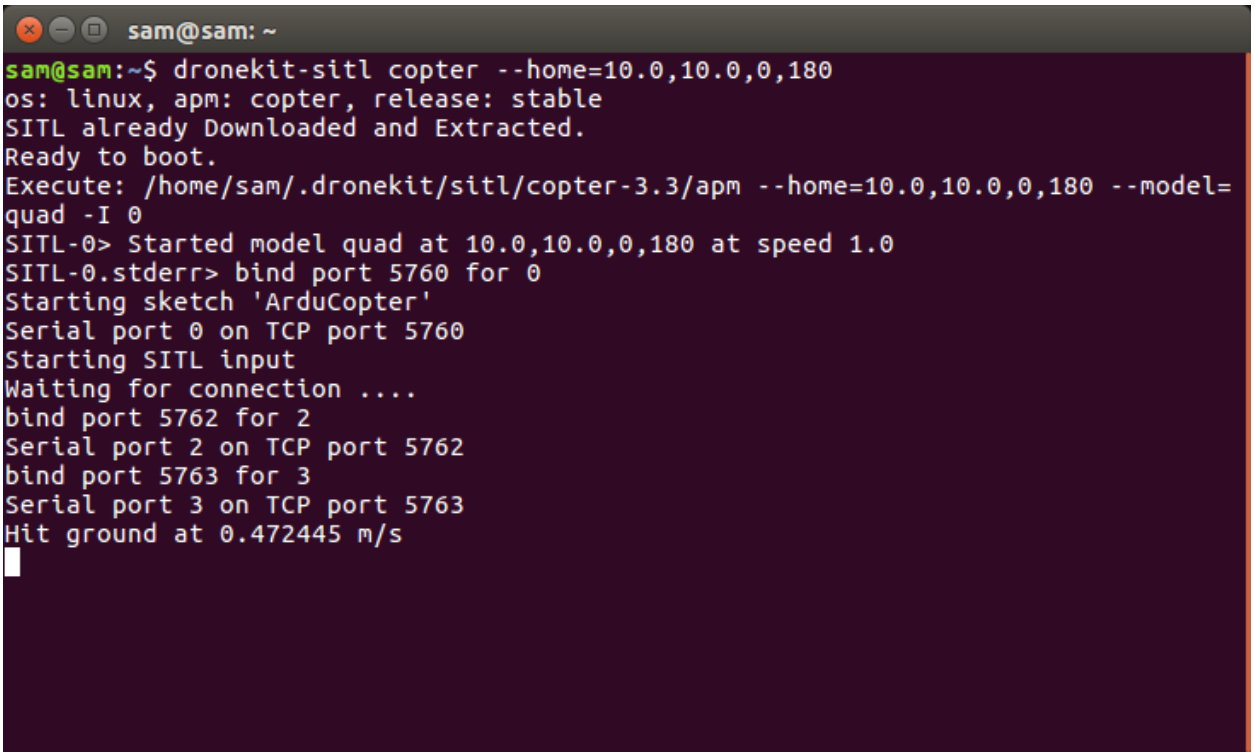


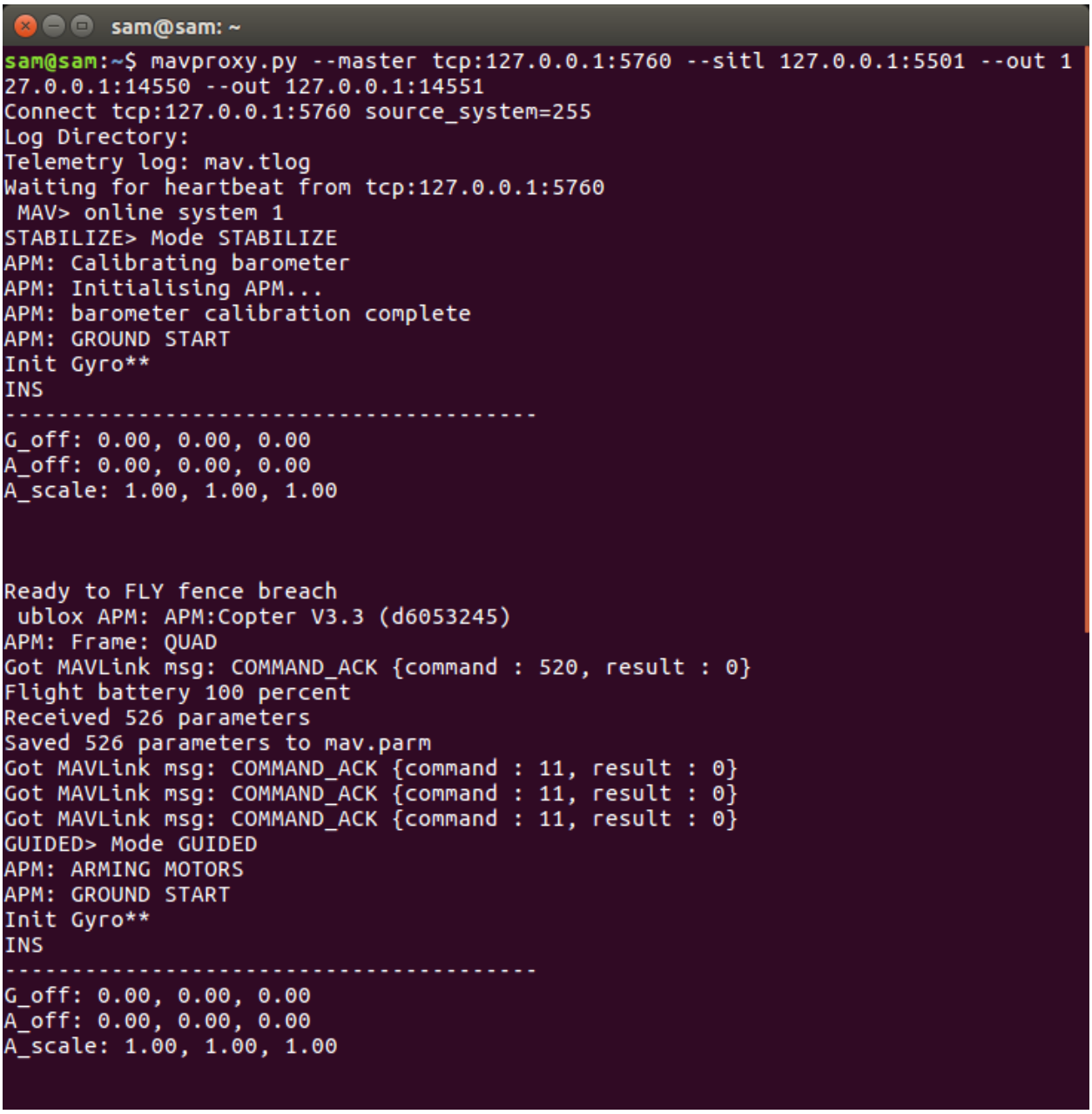
8.6.4 Waypoint convex text file

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
```

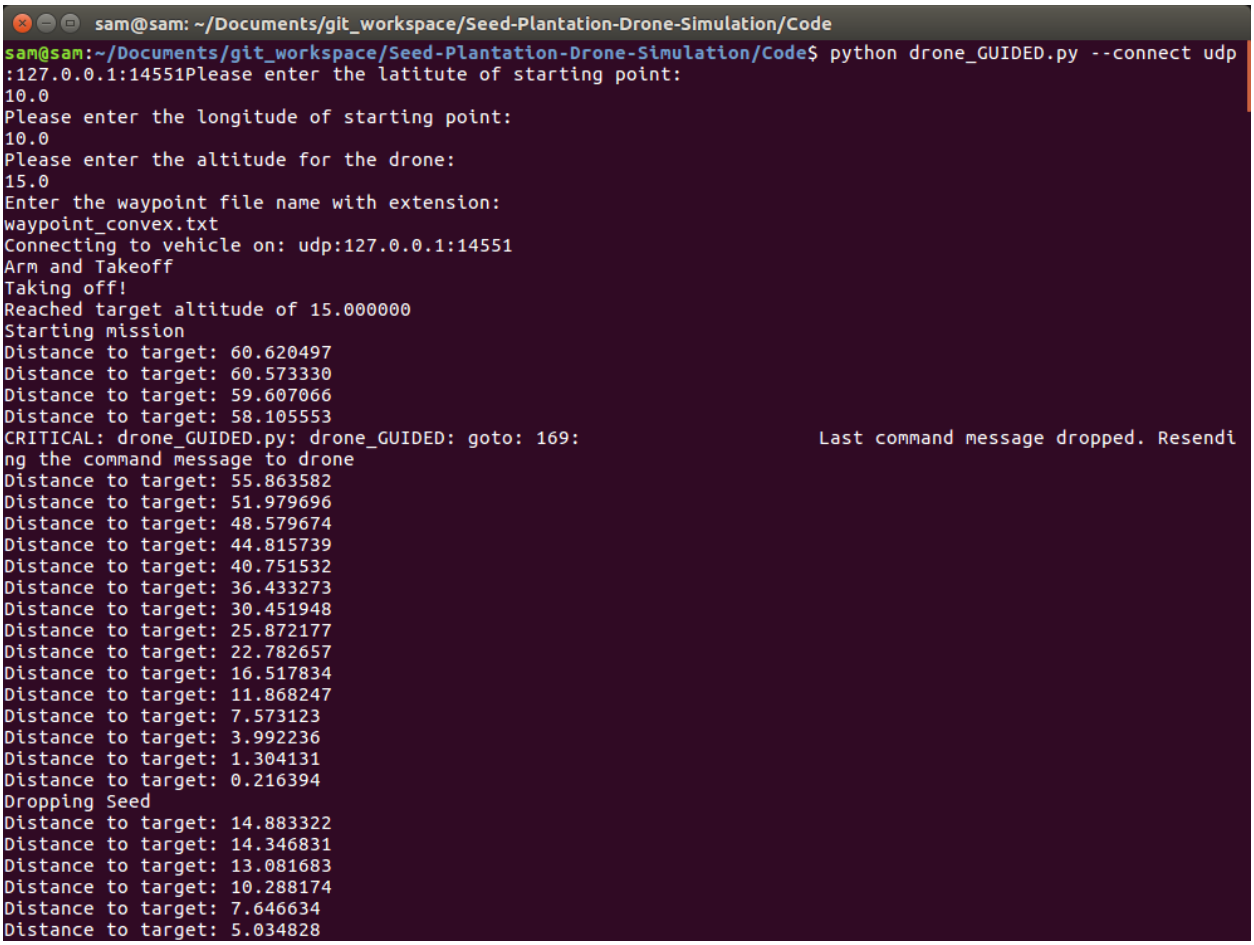
8.6.5 Terminal outputs



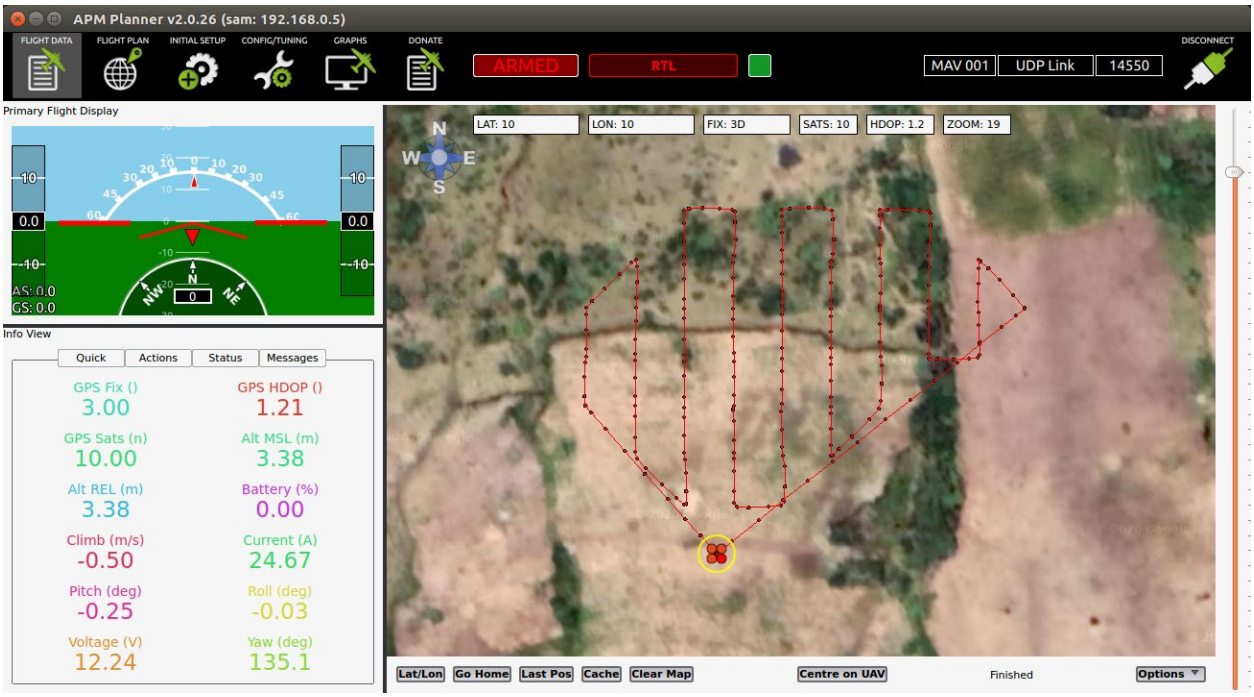

sam@sam: ~
sam@sam:~$ mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 1
27.0.0.1:14550 --out 127.0.0.1:14551
Connect tcp:127.0.0.1:5760 source_system=255
Log Directory:
Telemetry log: mav.tlog
Waiting for heartbeat from tcp:127.0.0.1:5760
MAV> online system 1
STABILIZE> Mode STABILIZE
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