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")
```