# 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")
```