3D motion planning project report

Explain the Starter Code

1. Explain the functionality of what's provided in motion_planning.py and planning_utils.py

motion_planning.py contains the basic event driven structure for using the drone. It subclasses Drone class and registers callbacks for different types of messages that drone will produce. Specifically, there are 3 types of messages which the callbacks are registered: Position messages, Velocity messages and State messages and they are produce when there is change in position, velocity or state of the drone.

Also in this file, the class States is defined which contains the 7 states which drone can be in one of them.

Based on the message that the program receives it calls the corresponding callback. Then the callback decides what the next state would be and what drone needs to do depending on position, velocity, waypoints and the current state. The callbacks in turn call transition functions to change the drone state and send different command like takeoff, land, arm, disarm etc. to the drone.

The function that we should complete is the **plan_path** which is called when the state changes to **ARMING**. This function is responsible for setting the global home position and current drone local position. Then it should load the obstacle map and based on start and gold position of the mission it searches the state space to find the lowest cost path. It then should prune the path and minimize number of waypoints and send the result waypoints to the simulator.

planning_utils.py contains the helper functions which are needed for the motion planning. The 2 important functions in this file are **create_grid** which returns a grid representation of a 2D configuration space based on given obstacle data, drone altitude and safety distance arguments and **a_star** which takes the grid, start, goal and heuristic function as input and finds the lowest cost path between start and goal.

Function **Action** returns next possible actions and **valid_actions** returns valid actions based on obstacle map and map borders.

Implementing Your Path Planning Algorithm

1. Set your global home position

To do so, first I read latitude and longitude from ‘colliders.csv' as instructed and then set the global home position in geodetic coordinates using this command to drone:
self.set_home_position(lon0, lat0, 0)

2. Set your current local position
To get the current local position of the drone, first we should get global position of the drone which is geodetic coordinates from the GPS using:
global_position = [self._longitude, self._latitude, self._altitude]
and then convert this coordinate to NED coordinates in respect to global home using the helper function:
local_p = global_to_local(global_position,self.global_home)

3. Set grid start position from local position

This line sets the start position from the local position considering shifting for offset obtained from **create_grid** function. It gives the grid cell coordinates of the drone on inside the grid:
grid_start = (int(local_p[0]-north_offset), int(local_p[1]-east_offset))

4. Set grid goal position from geodetic coords

To make it easier for testing multiple goals, I defined a global variable **goal_coordinates** outside the **MotionPlanning** class and each time we want to test a different goal, we can set this variable to the goal geodetic coordinates which we can find from the simulator. To cover the goal coordinate to NED we use **global_to_local** function like before:
goal_local_pos = global_to_local([goal_coordinates[0] ,
goal_coordinates[1],self.global_home[2]],self.global_home)

and we convert it to grid cell coordinates too:
grid_goal = (int(goal_local_pos[0]-north_offset), int(goal_local_pos[1]-east_offset))

Now we have **grid_start** and **grid_goal** which are in terms of cells coordinates of the grid. We can feed these coordinates to our search algorithm.

5. Modify A* to include diagonal motion (or replace A* altogether)

A* algorithm is defined in **planning_util** file. The implementation only considers horizontal and vertical moves from a cell. To make it more general and flexible we can add diagonal moves to it. To do so, I added 4 more possible moves to the Action class in addition to existing moves. The added moves are:

NORTHWEST = (-1, -1, np.sqrt(2))
NORTHEAST = (-1, 1, np.sqrt(2))
SOUTHWEST = (1, -1, np.sqrt(2))
SOUTHEAST = (1, 1, np.sqrt(2))

The first 2 numbers are relative move from current cell and sort of 2 is the cost to move to next diagonal cell.

Also to make sure that next move is valid we need to modify **valid_actions** function to check for the obstacles and grid borders and if next move is not valid, remove it from list of possible next actions.

6. Cull waypoints

Function **prune_path** takes the path calculated from search and prunes it. It starts from first point of the path and calculates lines from this point to all other points of the path. It uses Bresenham algorithm to find the cells which fall on the line. For each cell on the line, it checks to see if it is on obstacle or not. If there is no obstacle on any point between 2 points, the points between them will be flagged to be removed. If there is an obstacle anywhere between 2 points, the search for these 2 pints stops and next search starts.

Execute the flight

1. Does it work?

I found several points of the map in the simulator and created points in the code to represent them. They are global variables names coordinates1, coordinates2, coordinates3, coordinates4 and coordinates5. Each time I changed the goal_coordinates to one of these destinations and ran the program and the drone made it to the goal successfully without hitting the obstacles.

Extra Challenges: Real World Planning

1-Probabilistic roadmap

To implement probabilistic roadmap, I created a python file named **probabilistic_roadmap_generator.py** which creates random points in 2D space and then selects the points which are not an obstacle on the grid in 3rd dimension and stores these points in **free_space_random_points** variable.

Then based on the remaining points, we need to find the valid actions which are the actions between points which don't have any obstacles between and also add the cost which is the distance between 2 points. Bresenham algorithm is used again for this stage. The result is **valid_actions_map** which for each points holds delta to other possible points and the cost for getting to them. The combination of the points and actions determine the graph which should be searched for planning. The result is saved to pickle files to be used by planning function.

In **plan_path** function, if **use_probabilistic_roadmap** is True then these pickle files will be loaded and the points and actions get updated considering start and goal points.

For this purpose a modified version of **a_star** function named **probabilistic_a_star** is created which gets **valid_actions_map** as parameter too and finds the lowest cost path from start to goal.

I tried different number of random points and found out for random graphs smaller than size 30*30 there is a higher chance that the path between start and goal can not be found. But for graphs larger than 35*35 the path planning works well and finds the path almost every time although the paths are not optimal as expected.

2-heading

To change the heading based on the source and goal, after calculation the waypoints the following lines were added to calculate the heading for each waypoint:

```
for i , w in enumerate( waypoints[2:-1] ):
    waypoints[i][3] = np.arctan2( (waypoints[i][1]-waypoints[i-1][1]), (waypoints[i][0]-waypoints[i-1][0]) )
```

3-helix

Implemented a simple helix using helix equations:

```
waypoints = []
    for t in np.arange(0,12.6,.1):
        waypoints.append( [int(path[0][0] + north_offset + 15 * math.cos(t) ) , int(path[0][1] + east_offset + 15 * math.sin(t)) ,int( 5*t ) , 0 ] )
```

4-deadbands

Tried few different numbers in **local_position_callback** function to change dead band radius. Looks like the radius 1.00 meter is a good value, meaning once the drone gets into 1 meter of the waypoint, it will start heading for the next one.