Explain the Starter Code:

The ***backyard_flyer_solution.py*** is a simple script to fly a drone in a predefined trajectory. Whereas, ***motion_planning.py*** script takes help of the ***planning_util.py*** script to come up with its trajectory. The ***motion_planning.py*** has a method called **plan_path()** which as the name says does the planning for the drone. It sets the drone's home position, creates a grid of the environment and comes up with a path to traverse through the grid using the relevant helper methods from ***planning_util.py***.

Implementing your Path Planning Algorithm:

| | |
|---|---|
| In the starter code, we assume that the home position is where the drone first initializes, but in reality you need to be able to start planning from anywhere. Modify your code to read the global home location from the first line of the `colliders.csv` file and set that position as global home (`self.set_home_position()`) | Line # 130 - Line # 147<br>Pretty straight forward code. Opened the file using regular Python IO, read the first line. Did some string parsing to extract the floating point values. |
| In the starter code, we assume the drone takes off from map center, but you'll need to be able to takeoff from anywhere. Retrieve your current position in geodetic coordinates from `self._latitude`, `self._longitude` and `self._altitude`. Then use the utility function `global_to_local()`to convert to local position (using `self.global_home` as well, which you just set) | Line # 150 - Line # 153<br>Simply grab the global position from the Drone class' attributed list. Then call global_to_local method defined in frame_util.py script using global position and home position as the arguments. This method simply converts geodetic frame data to ECEF frame. |
| In the starter code, the `start` point for planning is hardcoded as map center. Change this to be your current local position. | Line # 153, Line # 166 - Line # 169<br>At 153, the drone's current local position is assigned to a variable. Later, this local position is subtracted from the offset to get a global start position. |

| | |
|---|---|
| In the starter code, the goal position is hardcoded as some location 10 m north and 10 m east of map center. Modify this to be set as some arbitrary position on the grid given any geodetic coordinates (latitude, longitude) | Line # 174 - Line # 181<br>Arbitrary coordinates (-122.401902, 37.794409) are chosen and passed to the global_to_local helper method to convert it into an ECEF frame. Later, the converted position is checked for collision on the grid. If it is then the location is dropped. |
| Write your search algorithm. Minimum requirement here is to add diagonal motions to the A* implementation provided, and assign them a cost of sqrt(2). However, you're encouraged to get creative and try other methods from the lessons and beyond! | Class Action in planning_util.py, line # 58 - line # 61 defines the diagonal motions and the cost associated with any such action.<br>In the same class, line # 91 - line # 98 checks if a given diagonal action is outside the grid or if the diagonal grid cell is an obstacle. If true, then the action is marked as invalid (by removing it). |
| Cull waypoints from the path you determine using search. | Line # 190 - Line # 205<br>The loop picks three neighboring points and checks for collinearity. A helper function called is_collinear is defined at line # 109 which returns true if collinear else false. If the points are collinear then the second and third node is removed from the path array. In case of an exception, the loop terminates. |
| | |