Udacity

Flying Car Nanodegree

3D Motion Planning Project

VAN Daro

Explain the Starter Code

In the ***backyard_flyer_solution.py***, the quad flies to the via point. The ***motion_planning.py*** is the modified version. However, there are more function involved. The path planning is given in the pla***n_path().*** In the ***plan_path()***, the colliders.csv is read. Then the ***create_grid()*** in ***planning_utils.py*** is responsible for discretizing the environment into grid. The start and goal are defined. Then A star search algorithm which is written in ***planning_utils.py*** is used to find the path. Lastly, we send the waypoints to the simulation.

Implementing Path Planning Algorithm

- **Modifying the code to read the global home position**
  The drone needs to be able to start planning from anywhere. In this section, this is done from line 123 to line 208. First, load the coordinate of latitude and longitude. We can get this data from the first line of excel file. In this case, the latitude is 37.792480 and longitude is -122.397450. Then set them to home position. This is done in line 127 in *motion_planing.py*.

```
colliders_data = 'colliders.csv'
# TODO: read lat0, lon0 from colliders into floating point values
lat0, lon0 = read_global_home(colliders_data)
# TODO: set home position to (lat0, lon0, 0)
self.set_home_position(lon0, lat0, 0)
```

- **Retrieve current position in geodetic frame and change to current local position**
  This is done from line 129 to line 132 *in motion_planning.py* by using the *global_to_local ()*

```
# TODO: retrieve current global position
local_N, local_E, local_D = global_to_local(self.global_position, self.global_home)
# TODO: convert to current local position using global_to_local()
print('global home {0}, position {1}, local position {2}'.format(self.global_home,
self.global_position,self.local_position))
```

- **Set arbitrary position on the grid given any geodetic coordinates**

In the starter code, the goal position is hardcoded as some location 10 m north and 10 m east of map center. We modify this to be set as some arbitrary position on the grid given any geodetic coordinate.
Set grid start position is done in the *motion_planning.py* from line 139 – line 141.

```
# Define starting point on the grid (this is just grid center)
        grid_SN = int(np.ceil(local_N - north_offset))
        grid_SE = int(np.ceil(local_E - east_offset))
        grid_start = (grid_SN, grid_SE)
```

Set grid goal position from global frame is done in the motion_planning.py form line

```
# Set goal as some arbitrary position on the grid
goal_N, goal_E, goal_alt = global_to_local(self.goal_global_position, self.global_home)
grid_goal = (int(np.ceil(goal_N - north_offset)), int(np.ceil(goal_E - east_offset)))
```

In order to run the code, we use command line with the following command:

```
python motion_planning.py --goal_lat 37.792945 --goal_lon -122.397513 --goal_alt 25
```

- A Star Search Algorithm
  In this section, I modified the a_star() in planning_utils(). The function below is added.
  This was added in the code (SE = South-East, NE = North-East, SW =South-West, NW = North-West)

```
SE = (1, 1, sqrt(2))
NE = (-1, 1, sqrt(2))
SW = (1, -1, sqrt(2))
NW = (-1, -1, sqrt(2))
```

The action is also added. This is done from line 92 to 99

```
if x + 1 > n or y + 1 > m or grid[x + 1, y + 1] == 1:
valid_actions.remove(Action.SE)
if x - 1 < 0 or y + 1 > m or grid[x - 1, y + 1] == 1:
valid_actions.remove(Action.NE)
if x + 1 > n or y - 1 < 0 or grid[x + 1, y - 1] == 1:
```

```
valid_actions.remove(Action.SW)
if x - 1 < 0 or y - 1 < 0 or grid[x - 1, y - 1] == 1:
valid_actions.remove(Action.NW)
```

- Cull waypoints from the path using search

  This is one in *planning_utils()* from line 167 to line190