Explain the Starter Code

Motion_planning File

1. On first readthrough, the code is a buildup of the previous project.
2. The difference lies in an extra PLANNING State that has been included in the enum States.
3. Once in this state, we read the grid from ‘colliders.csv file’

Utils File

Our A* Star function is included as an utility/helper function.
This is structured with an Actions class having a valid_actions function to help traverse defined path.

The search algorithm implementation is completely defined here and allows us to import and use the function.

Implementing Your Path Planning Algorithm

1. Setting Position

```
with open('colliders.csv', newline='') as f:
        r = csv.reader(f)
        r1 = next

lat0 = float((r1[0].strip('lat0'))
lon0 = float((r1[1].strip(' lon0'))
```

I’ve used Pythonic code to open the csv file using r/w functions.

The non-numeric headers are removed and the string data is converted to float

2. Set Local Position.

```
global_position = (self._longitude,self._latitude,self._altitude)

local_position = global_to_local(global_position,self.global_home)
```

Standard Pythonic function call.
Global position is derived as a tuple of length 3 (3-Dimensions)

3. Set Grid Start Position

```
grid_start = (int(-north_offset + local_pos[0]), int(-east_offset + local_pos[1]))
```

4. Set grid goal position from geodetic coords

I’ve experimented with a lot of cool ideas but my submission includes one goal:

```
goal_lon = -122.396640
goal_lat =  37.796232
```

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

I used the boiler plate code provided in planning_utils as follows:

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

Checks against diagonals and obstacles

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

5. Cull Waypoints

I used the convenient Coll test to remove extra waypoints.