Based on Rubrics : https://review.udacity.com/#!/rubrics/1534/view

Explaining points step by step:

Explain the starter code :

In plan_path function :

1. Setting target altitude (TARGET_ALTITUDE) and safety distance (SAFTY_DISTANCE) from obstacles.
2. Its loading the colliders.csv file in float format, starting from 3rd row.
3. `data = np.loadtxt('colliders.csv', delimiter=',', dtype='Float64', skiprows=2)`

4. Then on basis of returned data from colliders.csv , a grid is being formed on basis of TARGET_ALTITUDE and SAFTY_DISTANCE. Setting obstacles as 1 and free path as zero.

5. `grid, north_offset, east_offset = create_grid(data, TARGET_ALTITUDE, SAFETY_DISTANCE)`

6. This create_grid function is defined in "planning_utils" file.
7. Setting grid start and grid end as center point of grid . Then calling A star algorithms with existing grid and heuristic and grid start and end position for extracting waypoints from grid start to grid end.
8. A* is returning waypoints based on four actions only in four directions(N,E,W,S). Then waypoints are being sent to simulator.
9. **Planning.utils.py :** contains functions create_grid which is returning free and obstacles space. It also has implementation of A* algorithm with defined heuristics.

Implementing Your Path Planning Algorithm:

As provided in rubrics I have added line number 129 to 131 . But it will always set home position always to lat, long which is provided in colliders.csv.

```
124         print('1-global home {0}, position {1}, local position {2}'.format(self.global_home, self.global_position,self.local_position))
125         # TODO: set home position to (lon0, lat0, 0)
126         # TODO: retrieve current global position
127         # TODO: convert to current local position using global_to_local()
128         #Assignment - START
129         #with open("colliders.csv") as f:
130         #    content = f.readlines()
131         #self.set_home_position(content[0][22:34],content[0][5:14],0)
132
133         self.set_home_position(self.global_position[0],self.global_position[1],self.global_position[2])
134         new_local_position = global_to_local(self.global_position, self.global_home)
135         #Assignment - END
136
```

So I have added line 133 which will set home position to current global_position. So that next time it will take off it , that will its home position.

In line 134 : getting new local position by calling global_to_local position.

In Line 145 and 146 , I am converting local NED to relative grid points and assigning into grid_start.

```
143         # TODO: convert start position to current position rather than map center
144         #Assignment -START
145         new_local_grid_point_x,new_local_grid_point_y = local_to_grid(self.local_position,north_offset,east_offset)
146         grid_start = (new_local_grid_point_x, new_local_grid_point_y)
147         # TODO: adapt to set goal as latitude / longitude position and convert
148
149         #new_local_position_tar = global_to_local([-122.3974446,37.7928291,0], self.global_home) # grid center to x+20 position GPS
150         #new_local_position_tar = global_to_local([-122.3974429,37.792477,0], self.global_home) # grid center  GPS
151         #new_local_position_tar = global_to_local([-122.395725,37.792247,93.115], self.global_home) # one building rooftop  GPS ,have to char
152         new_local_position_tar = global_to_local([-122.397221,37.792661,0], self.global_home) # diagonal (20,20)  GPS
153
154         new_local_grid_point_x_tar,new_local_grid_point_y_tar = local_to_grid(new_local_position_tar,north_offset,east_offset)
155         print("new_local_grid_point_x_tar,new_local_grid_point_y_tar",new_local_grid_point_x_tar,new_local_grid_point_y_tar)
156         grid_goal = (new_local_grid_point_x_tar, new_local_grid_point_y_tar)
157
158         # Run A* to find a path from start to goal
159         # TODO: add diagonal motions with a cost of sqrt(2) to your A* implementation
160         #Assignment - Diagonal motion has been implemented in my_planning_utils.py file
161         path, _ = a_star(grid, heuristic, grid_start, grid_goal)
162
```

In line 152 to 156 , I am giving any GPS co-ordinates and converting to grid positions and assigning to grid_goal.

In line 160, calling a_star algorithm which has implementation in my_planning_utils.py . This a_star algorithm has implementation to go to total 8 possible directions. (N,E,W,S, NE,NW,SE,SW). Below two screen shots from my_planning_utils.py file.

In below screen shot line number 59 to 63 : diagonal step and its associated cost. In next screen shot : line 94 to 101 has all valid actions to go diagonal as well.

```
56  NORTH = (-1, 0, 1)
57  SOUTH = (1, 0, 1)
58  #Assignment-START
59  NORTH_EAST = (-1,-1,1.4)
60  SOUTH_EAST = (1,-1,1.4)
61  NORTH_WEST = (-1,1,1.4)
62  SOUTH_WEST = (1,1,1.4)
63  #Assignment-END
64  @property
```

my_planning_utils.py

```
80
81  # check if the node is off the grid or
82  # it's an obstacle
83
84  if x - 1 < 0 or grid[x - 1, y] == 1:
85      valid_actions.remove(Action.NORTH)
86  if x + 1 > n or grid[x + 1, y] == 1:
87      valid_actions.remove(Action.SOUTH)
88  if y - 1 < 0 or grid[x, y - 1] == 1:
89      valid_actions.remove(Action.WEST)
90  if y + 1 > m or grid[x, y + 1] == 1:
91      valid_actions.remove(Action.EAST)
92
93  #Assignment-START
94  if (x-1 < 0 or y-1 < 0) or grid[x-1,y-1] == 1:
95      valid_actions.remove(Action.NORTH_EAST)
96  if (x + 1 > n or y - 1 < 0) or grid[x+1,y-1] == 1:
97      valid_actions.remove(Action.SOUTH_EAST)
98  if (x-1 < 0 or y + 1 > m) or grid[x-1,y+1] == 1:
99      valid_actions.remove(Action.NORTH_WEST)
100 if (x+1 > n or y + 1 > m) or grid[x+1,y+1] == 1:
101     valid_actions.remove(Action.SOUTH_WEST)
102 #Assignment-END
103 return valid_actions
```

In line 163 to 165 in below screen shot , I have added new heuristics functions for sq. root distance between two points.

```
159 #def heuristic(position, goal_position):
160 #    return np.linalg.norm(np.array(position) - np.array(goal_position))
161
162 #Assignment-START
163 def heuristic(position, goal_position):
164     h = np.sqrt((position[0] - goal_position[0])**2 + (position[1] - goal_position[1])**2)
165     return h
166 #Assignment-END
```

'my_planning_utils.py' saved