Udacity Nanodegree

Sumedh V Joshi

May 2020

1 Starter Code - Explanation

In the backyard flyer program the waypoints for the drone were explicitly defined. The basic requirement here was that the drone traverses through all these waypoints without any consideration for optimal path from start to goal. The motion planner differs from the backyard flyer in the following ways:

1. The map for the motion planner is vastly complex compared the that for the backyard flyer. The backyard flyer didn't have any obstacles once the drone had risen to the defined altitude level. Hence the waypoints could be easily defined. On the other hand the map for motion planner is a real world map where the drone has to navigate very carefully.

2. In motion planner, the plan_path method defines the start and goal locations for the drone. The create_graph method in utils then renders the 3D map as a 2D grid. The A_star path planning algorithm searches for an optimal path from the start to goal locations. Each node along this path is then fed as a waypoint to the simulator. If needed path pruning algorithms could be used to further optimise the path.

2 Motion Planner - Home location and global to local

I have uses pandas to set the home position as described below. One can also open the csv file and use split and strip to read the lat and lon values.

```
home = pd.read_csv("colliders.csv",header=None,nrows=1,sep=' |,', engine='python')
lat0 = home[0][0]
lon0 = home[3][0]
# TODO: set home position to (lon0, lat0, 0)
self.set_home_position(lon0,lat0,0)
# TODO: retrieve current global position
current_global_pos = [self._longitude, self._latitude, self._altitude]
# TODO: convert to current local position using global_to_local()
current_local_pos = global_to_local(current_global_pos,self.global_home)
```

Figure 1: Set home location

3 Motion Planner - Goal location

For defining the goal location, I calculated the maximum and minimum range of latitude and longitude values for the given map. The goal_location method then chooses any random (lat,lon) position and checks whether this location coincides with an obstacle or not. In the former case, the goal_location method chooses a new random goal.

```
def goal_location(lat_min,lat_max,lon_min,lon_max):
    goal_lon, goal_lan = uniform(lon_min,lon_max),uniform(lat_min,lat_max)
    goal_alt = 0
    goal_lla = (goal_lon,goal_lan,goal_alt)
    goal_ned = global_to_local(goal_lla,self.global_home)
    goal_pos = (int(goal_ned[0] - north_offset), int(goal_ned[1] - east_offset))

    return goal_pos
```

Figure 2: Set goal location

```
lat_min = 37.789658
lat_max = 37.797905
lon_max = -122.391999
lon_min = -122.402527
grid_goal = goal_location(lat_min,lat_max,lon_min,lon_max)
while grid[grid_goal[0],grid_goal[1]] == 1:
    grid_goal = goal_location(lat_min,lat_max,lon_min,lon_max)
```

Figure 3: Set goal location - obstacle check

4 Search algorithm

I have used two approaches here.

1. Modified A_star search - Additional motion primitives in the diagonal direction have been added. Each diagonal motion has a cost of $\sqrt{2}$ associated with it.

2. Probabilistic random search - The obstacles are represented as polygons. Using the sampler class discussed in the lectures, a graph is plotted wherein the nodes and edges of the graph represent all possible paths for the drone. A modified version of A_star is implemented to find the optimal path from start to goal location.

5 Path pruning algorithm

I have implemented the collinearity check algorithm here. If three points lie along the same path one of the points is culled to reduce the number of waypoints and optimise the path.