Explain the Starter Code

Q: Test that *motion_planning.py* is a modified version of backyard_flyer_solution.py for simple path planning. Verify that both scripts work. Then, compare them side by side and describe in words how each of the modifications implemented in motion_planning.py is functioning.

A: Even at first glance it is clear than motion_planning.py is very similar to backyard_flyer_solution.py. By comparing the two files lines by line it is clear that there are a few important differences:

- motion_planning.py has more imports such as planning_utils
- there is a new State called PLANNING
- both include callback function for position, velocity and state updates, but the call back functions are a little difference
    - the local_position_callback function includes a call to calculate_box() that updates the current list of all way points
    - the state_callback function add a PLANNING state between ARMING after the TAKEOFF
- motion_planning.py has a new function called caculate_box -> for dynamic/fine-planning updates
- in motion_planning.py the global position is not set in the arming_transition function so it needs to be set by the plan_path function which is call before the TAKEOFF state
- in motion_planning.py it is assume that the target_position is already set (during planning) as the take of altitude is set by the target_position
- in motion_planning.py the yaw is set by the target_position and not set to 0
- motion_planning.py has a new function called send_waypoint
- motion_planning.py has a new function called plan_path for pre/coarse-planning – this function has a lot of TODO's!!

Implementing Your Path Planning Algorithm

Q: 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())

A: This is done in motion planning.py from line 124-135. The first line the file colliders.csv is the lat0, lon0 so the first line of this file is read and parse and used as the inputs to set the global home position. In subsequent calculation the global home position is used to convert between local and global position.

Q: 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)

A: This is done in motion planning.py on line 138. If the global home is set correctly, then the global_to_local function will convert the global position to local position.

Q: In the starter code, the start point for planning is hardcoded as map center. Change this to be your current local position.

A: This is done in motion planning.py on line 151. If the current position is already calculated us this to choose the closest grid location

Q: 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)

A: This is done in motion planning.py from line 178-179. This involves several steps.

- Take the desired lat & lon and format it as the stand global position tuple
- Use global to local to convert to local position
- Use the grid offsets to calculate the corresponding grid position

Input the lat and lon could be done in many way, I flew the drone around the city and recorded several valid location that did not interfere with obstacles and commented/uncommented these lat/lon pairs to fly the drone and the city.

Q: 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!

A: The basic A* function didn't need to be modified, but the list of possible actions and cost needed to be modified (lines 56-63) and the valid actions functions need to be modified to handle the diagonal moves and determine if they are valid (lines 74-93).

Q: Cull waypoints from the path you determine using search.

A: The result raw path from A* was very slow because the drone tries to stop at every point. The I implements was to combine collinear segment. This was implement in function combine_segments_colinear that I added to planning_utils.py in lines 154-179 and call from motion_planning.py on line 228. This reduces the number of points and stop without changing the path at all. However, if the drone needs to fly at a heading that is not a multiple of 45 degrees this it needs to zig-zag between right angle and diagonal points. In order to convert these zig-zag points into an arbitrary diagonal line, a function based on Bresenham was added to planning_utils.py from lines 181-210 and call from motion_planning.py on line 235.

Executing the flight

Q: This is simply a check on whether it all worked. Send the waypoints and the autopilot should fly you from start to goal!

A: In order to check that it all worked I found a 7 different location around the city location in valid locations that didn't interfere with the obstacles or their 5m buffer zones. I randomly made the drone fly from one location to another by commenting/uncommenting goal location in the code (lines 157-176) and leaving the drone simulator running so that it started at the last goal location.