Project 2, FCND Program

Roswitha Remling, 2018-12-29

Step 6:
Explain what's going on in `motion_planning.py` and `planning_utils.py`

Task 1a:
What is different about `motion_planning.py` from `backyard_flyer_solution.py`

- Several new imports, such as (from) argparse, msgpack, planning_utils, udacidrone.frame_utils
- However, WebSocketConnection is no longer imported
- In `class States(Enum)`
  - A new State: PLANNING
  - Also states no longer are associated with integers (0,1, …) but now all are `auto()`
- `class BackyardFlyer(Drone)` is renamed to `class MotionPlanning(Drone)`
- `self.all_waypoints` became `self.waypoints`
- `calculate_box()` method was removed and is no longer called
- transition methods now set new state first
- `state_callback(self)` method now also
  - calls `self.plan_path()` once armed
  - and transitions to takeoff, `if self.flight_state == States.Planning`
- Target Altitude is no longer hardwired, but a variable contained in `self.target_position[2]`
- In method `waypoint_transition(self)` Heading is no longer set to 0, but expressed in a variable `self.target_position[3]`
- `Send_waypoints(self)` and `plan_path(self)` are two new methods, latter with several TODOs (see Step 8 write up for details)
- In `start(self)`, `super().start()` was replaced with `self().connection.start()`
- The `___main___` program now includes several lines using `argparse` methods

Task 1b: How do functions provided in `planning_utils.py` work?

`create_grid`
- Returns
  - an array of feasible and infeasible cells for the drone's altitude and safety distance based on provided data on obstacles
  - as well as the north (x) and east (y) offset of the array 0,0 point from the center of the grid
- if the obstacle is lower than the drone altitude + safety distance it is considered feasible

`class Action`: provides possible moves as well as their costs (x and y only, not diagonal)

`valid_actions`: eliminates actions (moves) if they are no longer on the grid or encounter an obstacle at the provided drone altitude (= removes infeasible actions)

`a_star`
- returns the lowest cost path and the cost of that path, based on start, goal and heuristic

- or prints “Failed to find a path!” and returns and empty path array

`heuristic`

- returns the cost (absolute distance) from a given position to goal
- is consistent and admissible

Step 7: Write Your Planner

Updated `motion_planning.py` and `planning_utils.py`

These two commands worked for me (though solution was not elegant)

```
python motion_planning.py --goal_lon -122.398450 --goal_lat 37.793680 --goal_alt -0.147

python motion_planning.py --goal_lon -122.397450 --goal_lat 37.792680 --goal_alt -0.147
```

See also QuickTime Screen Recording `run04many.mov`

Step 8: Write it up

Please note, all line numbers refer to motion_planning.py unless specifically stated otherwise

```
# TODO: read lat0, lon0 from colliders into floating point values
```
lines 128-133

- read only the first line of colliders.csv
- strip new line
- split by spaces
- strip comma (,)
- assign float values to lat0 and lon0
- this only will work if the format of the first line is consistent, if the order of lat and lon are changed it will fail

```
# TODO: set home position to (lon0, lat0, 0)
```
line 136

- the method to use was provided
- key point to pay attention to was the order of longitude and latitude

```
# TODO: retrieve current global position
```
lines 139-141

set `global_position` from methods provided for latitude, longitude and altitude

`# TODO: convert to current local position using global_to_local()`
lines 144-147

- determine local position using `global_to_local` providing `global_position` and `global_home`
- I also set NEDs

`# TODO: convert start position to current position rather than map center`
lines 160-162

- Calculate grid x and y by adding `self._north` and `self._east` to the grid offsets
- assign `grid_start` variable

`# TODO: adapt to set goal as latitude / longitude position and convert`
lines 168+169

- Calculate local NED from passed in global goal position using `global_to__local` and global home
- Assign `grid_goal` variable from Goal NEDs (including north and east offset)
- This got a bit more complicated when I learned on Dec 29 that the goal should be passed from command line
- The following additional lines were also required (I hope I got them all here, for source please see line 2 in `motion_planning.py`)
    - Line 31 to update `Motion_Planning` initializer signature
    - Line 41 to include attribute in initializer
    - Lines 209-211 to define the arguments to be parsed
    - Line 216 sets `goal_global_position` for arguments passed
    - Line 217 updated calling `MotionPlanning` with passed in goal positions

`# TODO: add diagonal motions with a cost of sqrt(2) to your A* implementation – done`
`planning_utils.py` lines 61-65 and 96-104

- Added 4 actions to class Action (NW, NE, SW, SE)
- Added 4 additional conditions to eliminate actions if infeasible to method `valid_actions`

`# TODO: prune path to minimize number of waypoints`
lines 177-184 and `planning_utils.py` lines 108-134

- Renamed original path
- Called `prune_path` on the original path to create `pruned_path`
- Added 3 methods to `planning_utils.py` to remove waypoints that are collinear

Additional Comments:

1. Provided the time I had available, I decided to submit the minimum required, since I want to stay on schedule with this course. I admit that this solution is not elegant, and plan to spend more time on it after the course ends and post that on github as a project (the only silver lining I see is that it will re-enforce the learning)

2. On Dec 29, I saw from the student hub discussions that the goal location should be passed as arguments. I found the approach to a solution on line and referenced the source in line 2 of motion_planning.py (as well as referenced to line 2 where-ever changes were required). The rest of the solution was either based on lecture material (referenced) or my own

3. Pruning the path is functional, but the example commands provided still have sequences of small steps. This may be due to diagonal steps being included. This would clearly be very uncomfortable as a passenger. Also I reduced the maximum speed in the simulator to 5 and increased the SAFTETY_DISTANCE to 7 to avoid bouncing off a building.

4. As mentioned before, this solution is only the best I can do with the current time restrictions (I work full time and have other obligations as well). Clearly the techniques introduced in Part 2 Lesson 5 would allow much better solutions, as would extracting some of the steps into separate methods. (I am looking forward to spending more time on this in future)