# 3D Motion Planning

## Udacity – Flying Car and Autonomous Flight Engineer Program

## Introduction

The challenge here is to program a drone to be able to navigate autonomously between any two points in a 2.5D simulation of a real city, namely San Francisco. The simulation of the city and the drone is in the Unity environment, and the project will be written in Python.

Image of the city in Unity environment - **TODO**

## Starter Code

The starter code consists of 2 files – motion\_planning.py and planning\_utls.py. The starter code works!

### Motion\_planning.py

This code is like backyard\_flyer\_solution.py. However, unlike the backyard flyer, this code does not navigate a predetermined trajectory. It navigates the drone from one point to another in a grid setup using the A\* path-planning algorithm. The code is encapsulated in the MotionPlanning class with the following methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| start | Start the connection to receive events from the drone |
| arming\_transition  disarming\_transition | Event handlers to arm (turn on the rotors) and disarm (turn off the rotors) the drone |
| takeoff\_transition  landing\_transition | Event handlers to takeoff (given an altitude) and land |
| manual\_transition | Event handler to switch the drone to manual control |
| waypoint\_transition | Event handler to move the drone to the next waypoint in the path |
| send\_waypoints | This method send the waypoints to the simulator for visualization purposes |
| plan\_path | This method is the brains of the motion planning. It builds the path to navigate the drone to its destination via a series of waypoints, avoiding obstructions along the way.  Basically, it builds a collection of ordered waypoints and sends them to the simulator |

Highlighted methods are new as compared to backyard\_flyer.py

### Planning\_utils.py

This is a set of utility functions provided to facilitate the project. We will have to use/modify these utility functions in MotionPlanning->plan\_path() as we flesh out our new path planning code. The following functions are provided:

|  |  |
| --- | --- |
| **Method** | **Description** |
| create\_grid | Returns a grid representation of a 2D configuration space based on given obstacle data, drone altitude and safety distance |
| valid\_actions | Returns a list of valid actions given a grid and current node. Each action is an object of the class Action that has an associated cost and delta (tuple describing relative movement) |
| a\_star | Basic A\* path planning algorithm to compute a set of waypoints given a grid, start and goal |
| heuristic | Heuristic function for the A\* algorithm |

### Pseudocode for MotionPlanning->plan\_path()

1. Initialize target altitude and safety distance
2. Read in the obstacle map
3. Create the grid based on the target altitude and safety distance
4. Define arbitrary starting node and goal node
5. Run the A\* method to get back a path from start to goal, given the grid of obstacles and a heuristic function
6. Convert the path to a waypoint’s matrix in the SIM world
7. Display the waypoints in the simulator
8. Transition the drone to takeoff and navigate the planned path of waypoints to the goal

Image of backyard\_flyer\_solution drone flying in the sim – **TODO**

Image of motion\_planning drone flying in the sim - **TODO**