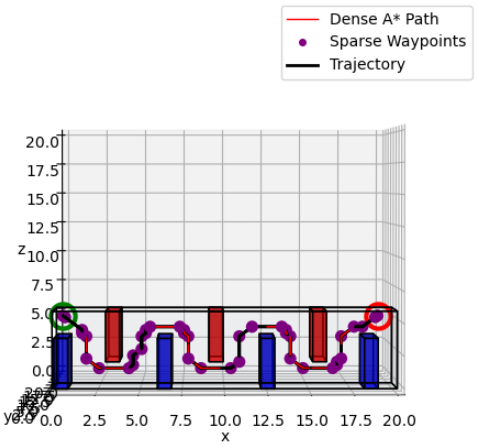
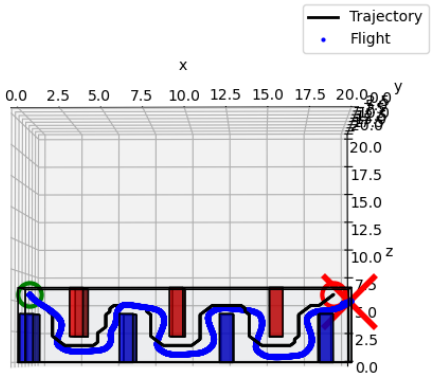
MEAM 620 Project 3

Sahachar Reddy Tippana

Findings: Below are some plots of the trajectories generated and comparison with the performance of my previous world trajectory. For comparison, I'm using Over_Under test case.

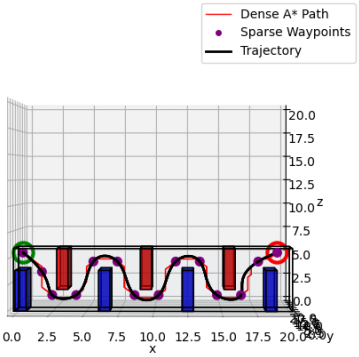
Results before modifications:

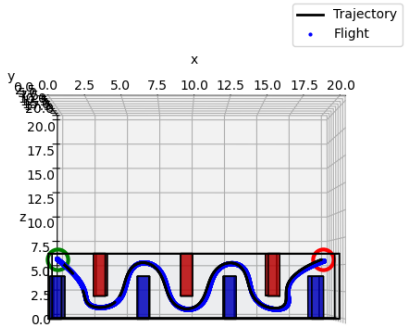


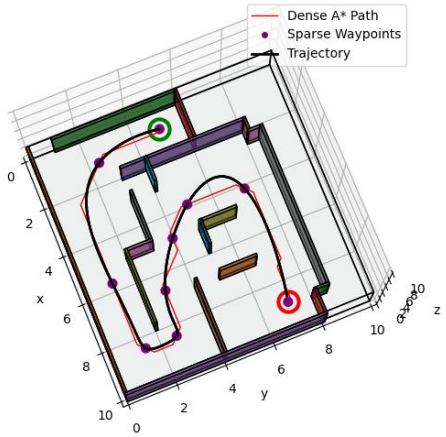


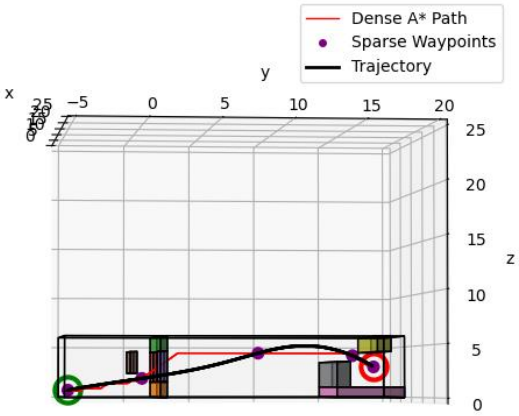
- The trajectory previously generated have their waypoints pruned based on the non collinearity check. The number of waypoints here are large. The generated trajectory is in black an is non differentiable i.e. not at all smooth

Results after Modifications:









- The trajectories generated after the modification (listed below) are very smooth, less waypoints.
- We can observe from the plots of over_under test case that the robot follows (blue) the trajectory (black) generated perfectly
- The trajectory is very smooth after the changes and performance is drastically increased.

Changes and why: Most of the *world_traj.py* file is changed for this project as follows -

1. **Implemented *min_snap()*** function to generate the minimum snap trajectory solution. Considering the poor performance of the constant velocity state trajectory generation code from project 1.3, I've decided to implement minimum snap solution instead. Leveraged the lectures 10 and 11 to define the constant constraint matrices (A, P) with-in this function. A supplementary function called *cont_var_matrix()* is written as a modular component in generating the sparse constant matrix A of size 8mx8m where, m is the number of segments in the trajectory (number of waypoints – 1). The coefficient matrix C is obtained using the *scipy.sparse.linalg.spsolve()* which solves for C from the matrix equation "AC=P". Both A and P are converted into sparse CSC matrix format before using *spsolve()*. This piecewise trajectory generation method allows for a smoother trajectory at high velocities and increases the performance by manifold with proper gains' tuning, trajectory waypoints' pruning, and segment time non-linear scaling.

2. **Wrote *pruner()* function** for pruning the number of waypoints in *world_traj.py* – instead of just deleting the collinear waypoints generated by the *graph_search()* as I've previously done in project 1.3. The *pruner()* function can be used in along with/without the non collinear points removal code i.e. *remove_collinear_prox()* function. The *pruner()* function leverages *path_collisons()* method of the world object being passed into the world_traj.py file. The method is a part of the world.py file in the flightsim.

   The function works iteratively. One iteration looks as follows – it takes in 1st two waypoints and generates some intermediate points using the *numpy.linspace()* which would be the possible trajectory the robot follows. All these intermediate points including the endpoints are passed into the *world.path_collision()* function to check if there is any collision. If a collision does not exist the same process is repeated for waypoints 1 and 3 and so on until there is a collision. If a collision exists in this check for say waypoints 1 and 5, we dump points 2 to 3 and start the whole iteration from waypoint 4. This check is performed throughout the waypoints resulting in a finely pruned collision free waypoints. The resulting waypoints are very sparse so much so that careful time scaling/dilation is required and mainly around the corners of the trajectory.

3. **Non linear scaling of Segment duration/times –** the times of each segment of the trajectory are scaled non lineary using "t_i = t_i * np.sqrt(1.65/t_i)" where t_i is the time taken for the robot to travel segment "i". This eaquation is nothing but $y^2 = a*x$. This allows for a smoother slightly curved trajectory generation.

4. **Slowing around and through the corners -** Using empirical experiments form the local runs on the given maps, I've determined a threshold (corner_thresh = 0.3*6/velocity magnitude) on the time segment (t_i) within which lies a corner in the trajectory. I have increased the time segment through these corners and the adjacent segments of each corner. All of these adjustments are a function of the velocity magnitude used so that the code is robust enough.

5. **Time dilation –** The initial and end segments' times are dilated/increased so that the drift is minimized. The dilate_factor introduced here is also a function of the velocity magnitude and the end segment dilation is a fraction of the original factor. The end segment dilation is also progressive i.e. the segment before it is also dilated a little so that the trajectory does not curve/deviate much to accommodate the dilation.

6. **Control gains' tuning –** The SE3Control gains are tuned further to be compatible with the minimum snap implementation. The gains are set aggressively in comparison with the project 1.3 which yield robust and high performance with the current world trajectory generation.