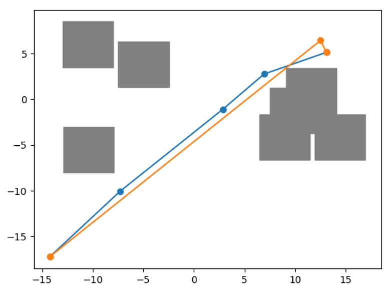
Q1.

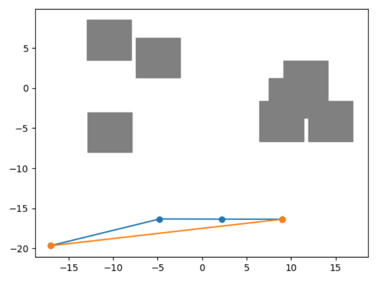
- NMP: Success rate = 0.97, computation time: mean = 0.63, stdev = 1.09
NMP dropout: Success rate = 0.67, computation time: mean = 1.13, stdev = 1.76
NMP lvc: Success rate = 0.86, computation time: mean = 1.58, stdev = 5.74

System Specification:
Purdue Cluster, RAM = 24gb | GPU = 1 core of V100 | CPU = 16 Core

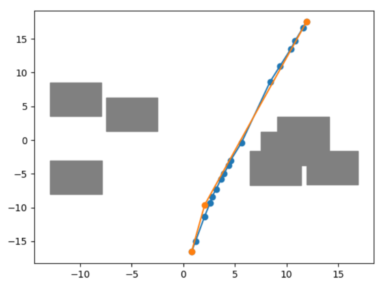
-



35.73646310786616 (RRT, Blue), 37.13409040577743 (mpnet, Orange)

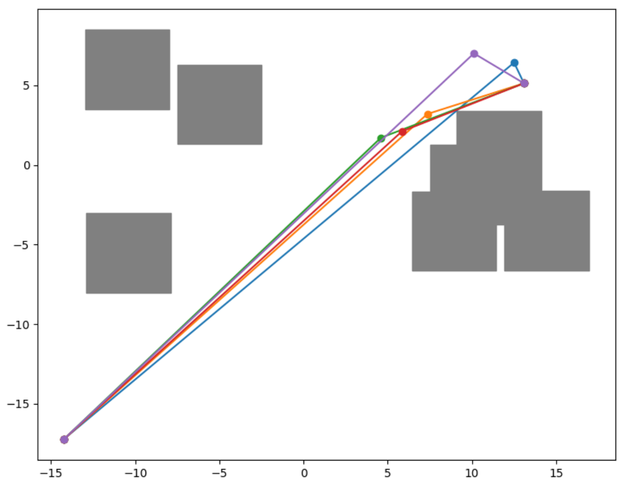


26.483747664921268 (RRT, Blue), 26.24720837936751 (mpnet, Orange)



35.907821533802284 (RRT, Blue), 35.92932872992365 (mpnet, Orange)

•



•

LVC smooths the generated path solution from the MPnet. It removes the intermediate redundant waypoints which can be replaced with direct steering from the previous point. It clearly helps out the computation time as it reduces the nodes to be visited.

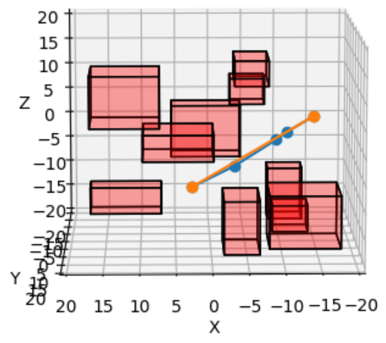
Dropout ultimately adds stochasticity to the model so that the planner would not reproduce the same points over and over again. This is why the visualized output above shows the five different paths.

Q2.

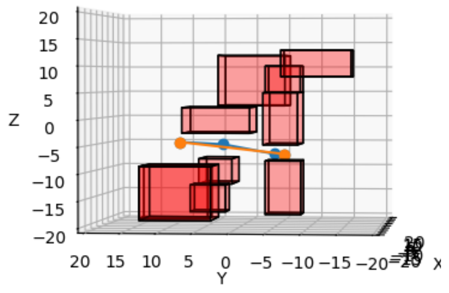
- NMP: Success rate = 0.98, computation time: mean = 0.58, stdev = 0.98
NMP dropout: Success rate = 0.74, computation time: mean = 1.24, stdev = 2.33
NMP lvc: Success rate = 0.87, computation time: mean = 1.58, stdev = 5.74

System Specification:
Purdue Cluster, RAM = 24gb | GPU = 1 core of A30 | CPU = 32 Core

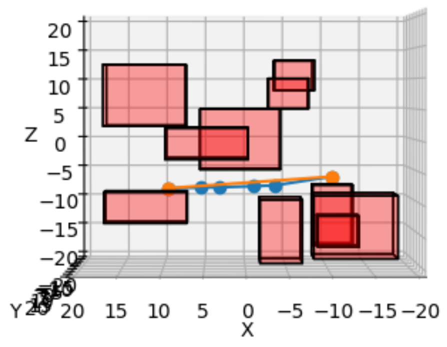
- 21.85631460550719 (RRT, Blue), 21.687727594692763 (mpnet, Orange)



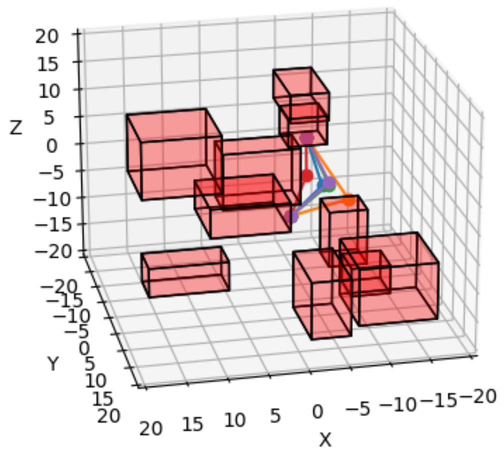
15.164734758085775 (RRT, Blue), 15.090392350699169 (mpnet, Orange)



26.10375895609176 (RRT, Blue), 26.025901528469 (mpnet, Orange)



•


Z
20
15
10
5
0
−5
−10
−15
−20
Y
−20
−15
−10
−5
0
5
10
15
20
X
20
15
10
5
0
−5
−10
−15
−20

A report describing each of your steps:

2d:

To remove dropout, I have created a model_nd.py which passes the probability of dropout of 0. A node will not be dropped when the test function call this model when the corresponding argument is passed.

The lvc function is disabled in the test file when the corresponding argument is passed.

3d:

CAE.3d:

```
self.encoder = nn.Sequential(nn.Linear(6000, 512),nn.PReLU(),nn.Linear(512,
256),nn.PReLU(),nn.Linear(256, 128),nn.PReLU(),nn.Linear(128, 30))
```

The parameter values have been modified to the environment inputs and start-goal dimensions

mpnet test.py, visualizer.py:

For the two files above, following lines have been added to handle the 3d case when the 's3d' is passed for the env parameter. It is mainly to align the env dimension and 3d goal and start point dimension.

```
total_input_size = 6000+6
AE_input_size = 6000
mlp_input_size = 30+6
output_size = 3
```

500 epoch has been chosed to reach the success rate requirement.

plan_s3d.py is created to collision check for 3d cases.

```
for j in range(0,3):
        if abs(obc[i][j] - s[j]) > size[j]/2.0:
           collision = False
           break
```

This handle the 3d cases with size variables with respect to the obstacle numbers.

The visualizer has been modified to render the 3d space using plot3D function from the matplotlib. Some of the codes from matplotlib library and corresponding community has been used and modified for this specific case.

How to run:

Detailed instruction is in the readme included in the submission.