Question 1: Simple 2D

- Specifications of Computer: 4. 70GHz x 14 Intel Core i7 processor with 16GB RAM and GetForce RTX 3050.

		NMP	NMP-w/o-Dropout	NMP-w/o-LVC
Success Rate		0.97	0.69	0.86
Computation Time	Mean	0.42s	0.81s	0.87s
	Standard Deviation	0.74s	1.32s	2.89s

Table 1: Test Results on 2D Environment

- Comparison using NMP results and given dataset
  Environment 0 on path 4001:

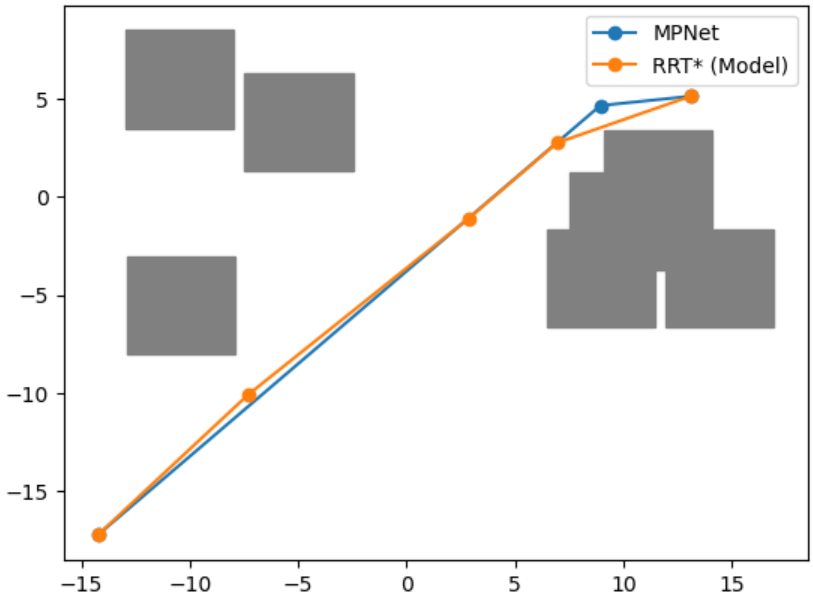

Figure 1: MPNet (blue): cost = 36.1; RRT* (orange): cost = 35.7

Environment 74 on path 4080:

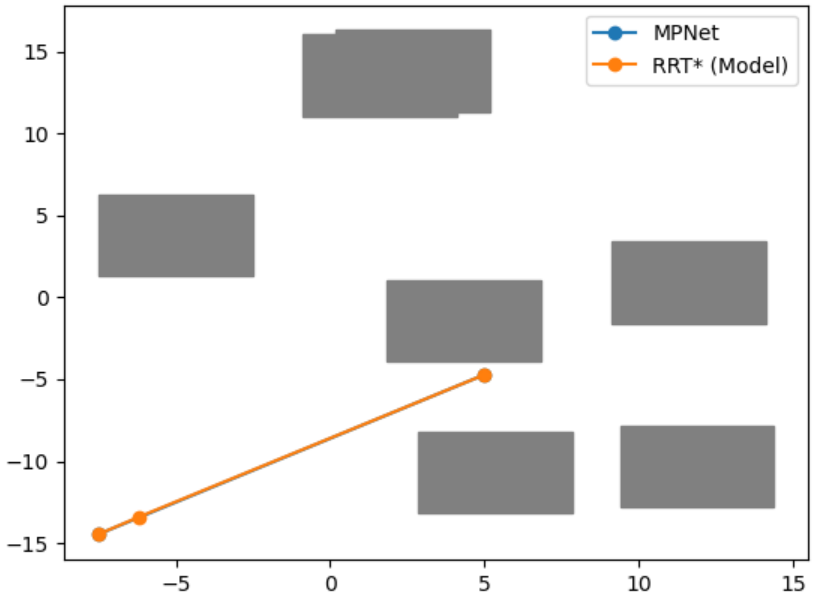

Figure 2: MPNet (blue): cost = 15.8; RRT* (orange): cost = 15.8

Environment 90 on path 4031:

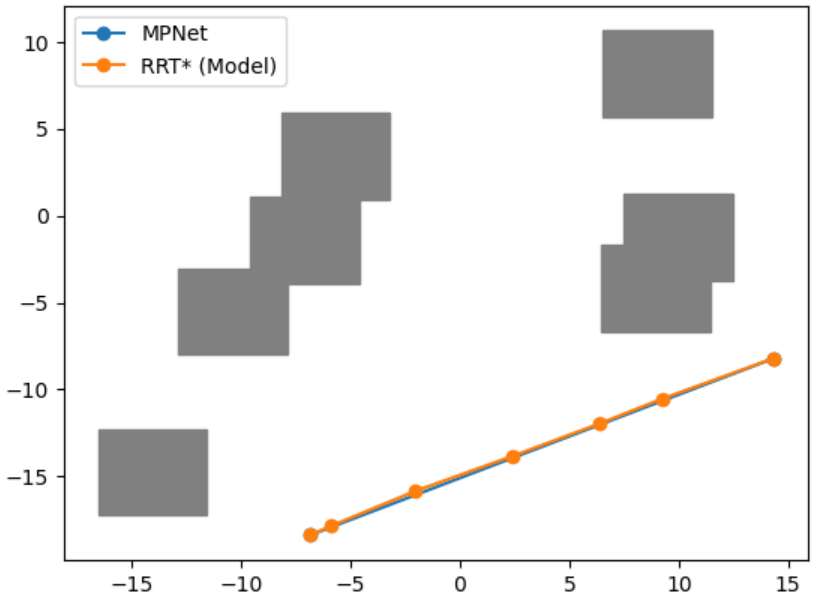

Figure 3: MPNet (blue): cost = 23.51; RRT* (orange): cost = 23.52

- Multiple paths for a fixed planning problem:

  Selected env 0 path 4001:

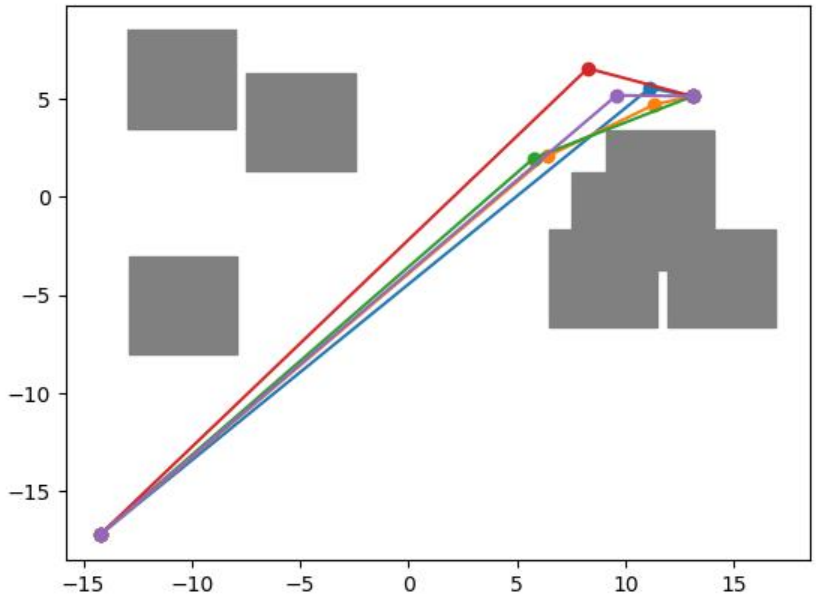

Figure 4: Multiple paths for fixed start and goal

- Describe significance of Dropout and lazy vertex contraction and their role in MPNet (max 100 words)

  “Dropout” selects some neurons randomly and masks them so that they will not affect the model. This is important in MPNet as it adds stochasticity, thus for example it would allow for different paths to be generated for the same start/goal problem. This increases overall performance.

  Lazy vertex contraction removes redundant states by checking the connectivity between non-consecutive states. This role is important as it smooths out the path and increases computational efficiency (as there will be less nodes to process).

Question 2: Complex 3D

- Specifications of Computer: 4. 70GHz x 14 Intel Core i7 processor with 16GB RAM and GetForce RTX 3050.

		NMP	NMP-w/o-Dropout	NMP-w/o-LVC
Success Rate		0.98	0.73	0.90
Computation Time	Mean	0.46s	0.79s	0.59s
	Standard Deviation	0.83s	1.45s	1.31s

Table 2: Test Results on 3D Environment

- Comparison using NMP results and given dataset:

  Two different views were given for each path shown to aid in visibility (e.g.: path is not colliding with obstacles)

  Environment 5 on path 2012

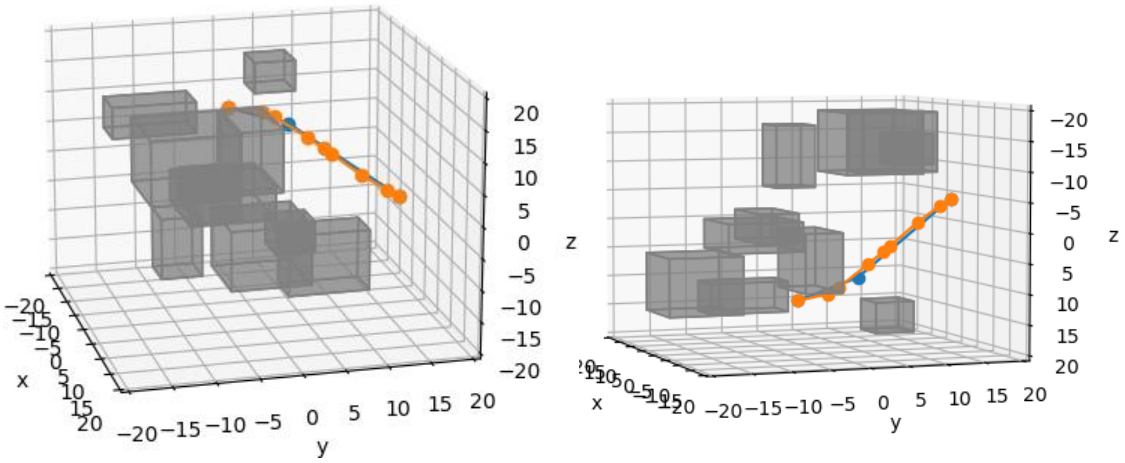

Figure 5: MPNet (blue): cost = 27.1; RRT* (orange): cost = 27.2

Environment 0 on path 2070:

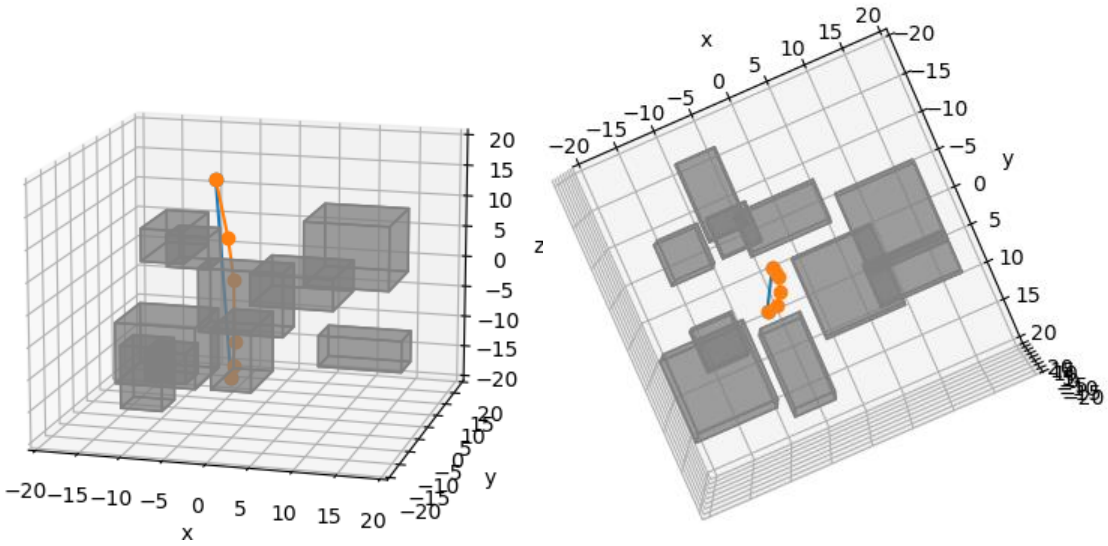

Figure 6: MPNet (blue): cost = 30.7; RRT* (orange): cost = 30.9

Environment 3 on path 2019:

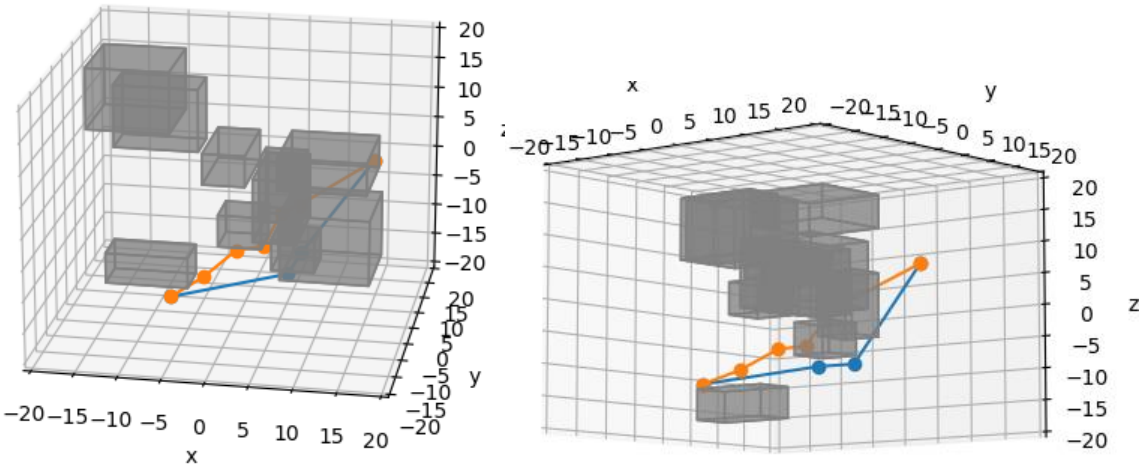

Figure 7: MPNet (blue): cost = 36.9; RRT* (orange): cost = 32.5

- Multiple paths for a fixed planning problem:

  Selected env 4 path 2008:

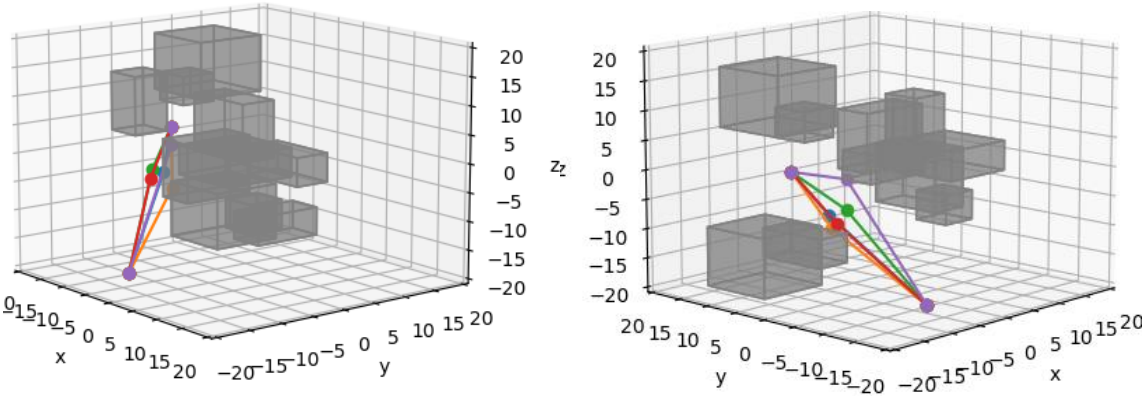

Figure 8: Multiple paths for fixed start and goal

- Changes done to program to work with 3D complex data:
  1. **Training the model**: I changed the encoder size and I/O values so that it would match the new environment and setup in the mpnet_train3D.py and CAE.py

     In the 2D case, the encoder size was 2800 which came from = 7 obstacles * 2 positions (x,y) * 100 (number of paths/env) * 2. With this formula, we can come up with the size of an encoder that would work for the 3D case: 10 obstacles * 3 positions (x,y,z) * 100 (number of paths/env) * 2 = 6000.

     The latent encoder dimension was left at 28 as training and testing proved to be a good number.

     Finally, instead of having an input/output values of 2+2 (for x,y position), we have 3+3 (for x,y,z position).

     The training was done with 500 epochs to the entire 10 environments using the 2100 paths.

  2. **Testing the model:** Same encoder size and I/o values were changed as explained above. Moreover, the collision algorithm was edited to handle 3D cases (the same logic was used).

  3. **Visualizing the model:** Several things were implemented here:

- Plot was changed from 2D axis to 3D axis using the tag = 3D projection when creating a figure
- Point cloud was adjusted so that it would use all three dimensions (x,y,z): reshape matrix into a x by 3 instead of the x by 2.
- Poly3DCollection library was used to plot the 10 obstacles. This was done by calculating the vertices position and generating the six surfaces needed to form a rectangular prism.
- Finally, path matrix was reshaped (from x by 2 to x by 3) to be able to get the 3D positions.