VI. Appendix

A. Real-world Experiment

For testing, a real-world experiment is done by combining MTG with a low-level AdaptiveON [7] motion planner for global navigation, as in Figure 1. We choose the generated trajectory with the last waypoint closest to the target as the following trajectory for the local planner. In each step, the robot takes the nearest next waypoint in the trajectory as the next goal. The model runs in the timestep around 0.01s on an onboard machine with an Intel i7 CPU and one Nvidia GTX 1080 GPU.

B. Analysis

Confidence of Trajectories: The trajectories are generated by the embedding Gaussian distributions, which are inputs of the decoder, and we train each distribution to cover each distribution to cover one traversable direction so the standard deviation can tell the confidence of the trajectory. When the standard deviation is large, the confidence is low because the distribution is not certain of the mean value. In our formulation, the variances for each trajectory can be calculated as $\mathbf{v}_c = A\mathbf{v}A^T$, where $\mathbf{v}$ is the variance of the embedding. $\mathbf{v}_c = \{\mathbf{v}_1, ..., \mathbf{v}_k\}$, where $\mathbf{v}_k$ and A are defined in Section III-B. The generated trajectories are shown in Figure 5; the higher the variance, the lower the confidence and the darker the trajectory is. As the following figures show, when the trajectories are near the wall or other non-traversable areas, the confidence is lower. In addition, the last row shows when the person is very close to the robot. Although the model can still ignore the person, the confidence of the trajectories (passing through the person) gets lower.

Generalization: As shown in the above results, our model is well-generalizable to the campus environment. However, for completely different scenarios like cities or woods, we don't expect our method to generalize very well to those out-of-distribution scenarios, which we believe is reasonable for most learning-based approaches. In addition to the out-of-distribution scenarios, because our model is not trained in non-traversable areas, the model cannot perform well in fully non-traversable areas. In Figure 6, we demonstrate that when the robot operates within non-traversable zones, its trajectories exhibit significant randomness. However, as the robot exits these zones, its paths realign within the traversable areas.

C. Traversability Map

Before building 2D maps, we use Lio-sam [15] to build 3D maps by running robots on the campus. Then we remove the noisy points and calculate the surfaces of the area. Then large cliffs, bushes, and buildings are detected and marked as non-traversable areas. Next, we press the 3D maps into 2D maps, creating traversable maps with a resolution of 0.1m. Finally, we manually label some missing non-traversable areas by combining satellite maps.

D. Trajectory Selection

The trajectory length is empirically determined by our robot's speed and the perceptive range of the Lidar sensor. On the one hand, we would like to generate long trajectories to provide the robot with proper guidance for future directions. On the other hand, unnecessarily long trajectories may lead to occluded areas or out of the perception of the Lidar sensor. Our two robots drive around 1-2m/s, and 10-20 meters are enough for robots to drive for 10-20 seconds before the next trajectory generation. Therefore, we chose 15 meters as our trajectory length.

E. Qualitative Evaluation

Figure 7 shows the birds-eye-view of the generated trajectories (purple) and ground truth trajectories (yellow and generated by A* algorithm). Cyan represents Lidar points from the middle channel (8th channel) of a 16-channel Lidar scan for reference. The white areas are all non-traversable areas. We can see the purple trajectories generated by our model are very smooth and cover the traversable areas (black areas), similar to the yellow A* paths.

F. Architectural Details

Perception models in Figure 8: The output of PointCNN has dimension 512, and it concatenates with the velocity embeddings, with dimension 256 as input of the Encoder. The output of the encoder has dimension 512. The function $h_\psi(\mathbf{z})$ contains two linear layers, which output dimension 256. The $A(\cdot)$ and $b(\cdot)$ functions are all single linear layers, and the outputs have the dimension of (512 x trajectory number).

Self-Attention Model: As in Figure 9, the trajectory embedding keeps the dimension 512.

G. Dataset

The dataset is collected on a university campus. As shown in the satellite map 10, the blue trajectories are for the training dataset and the red trajectories are for the testing dataset.

Fig. 5: Trajectory Confidence: The top row shows the camera view of the generated trajectories and the bottom row shows the bird-eye-view of the trajectories. The Cyan color represents the obstacles detected from the middle channel of the 3D Lidar.

Fig. 6: The out-of-distribution cases: The top figure shows the robot fully in a non-traversable area with no traversable area around it. The bottom figure shows the robot leaving the non-traversable area, where the robot can still generate good trajectories.

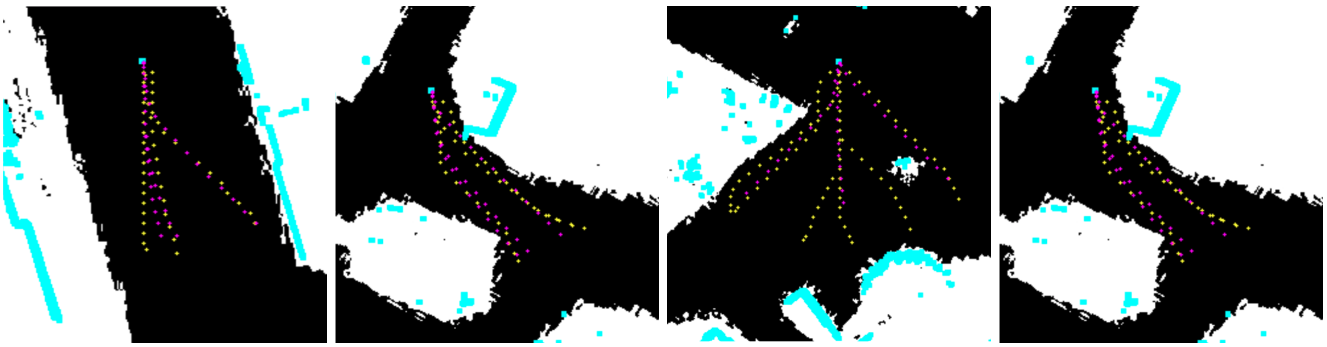

Fig. 7: Evaluation: A* ground truth paths are yellow trajectories and generated paths from MTG are purple trajectories. The white are non-traversable areas and cyan is the obstacle detected by the middle channel of the 16-channel Velodyne Lidar.

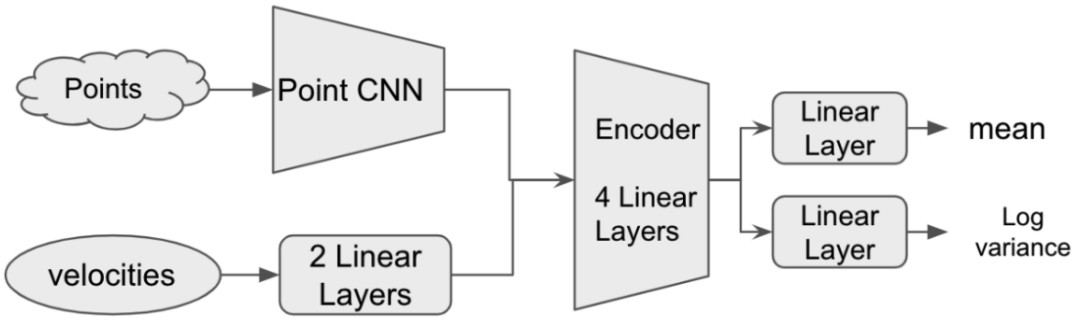

Fig. 8: Perception Models

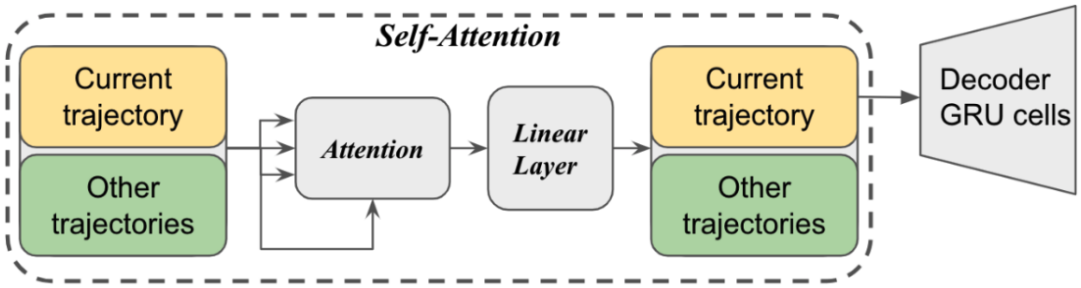

Fig. 9: Self-Attention Model

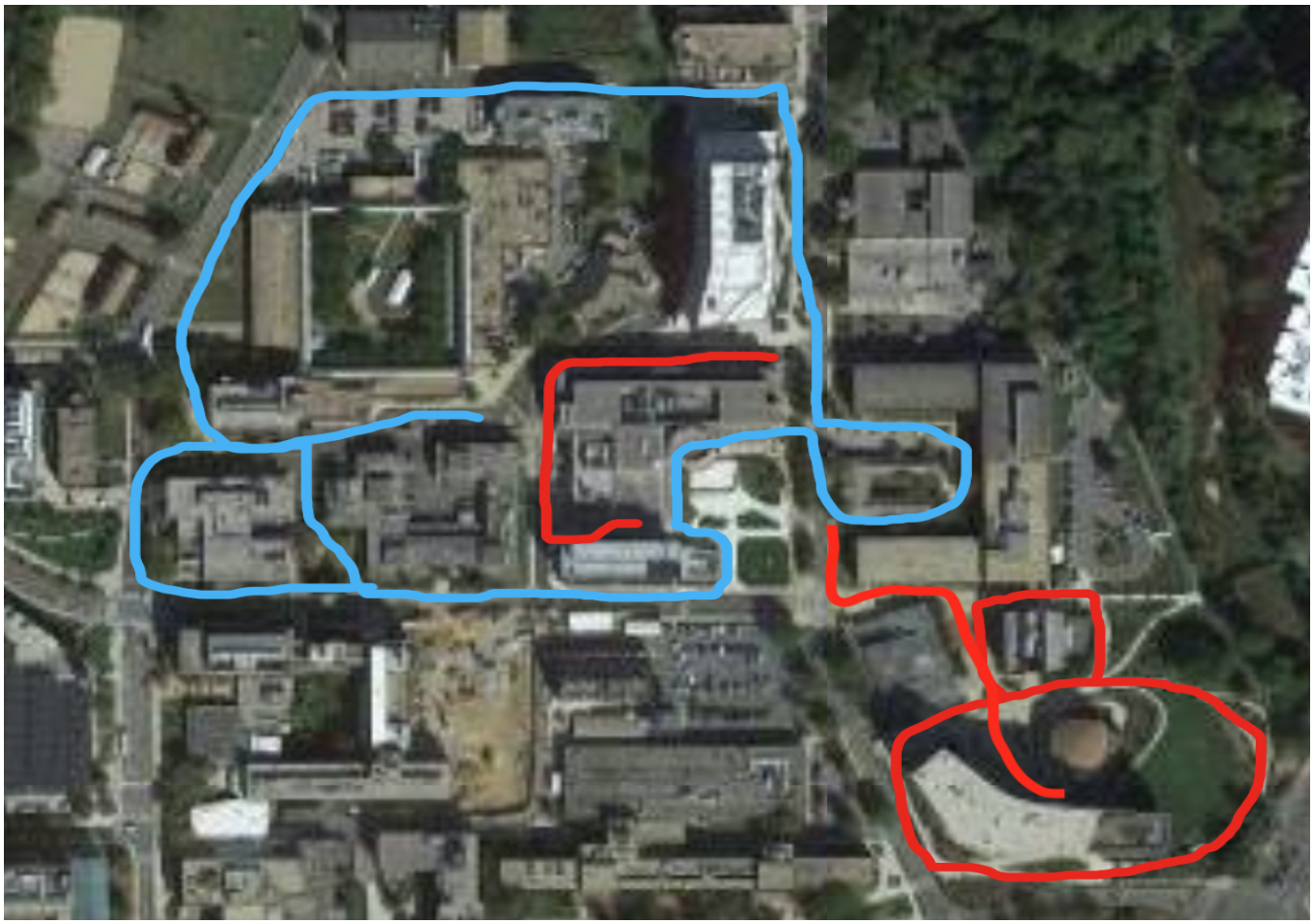

Fig. 10: The data is collected at a university campus. The blue trajectories are for training and red are for testing.