A Additional Implementation Details

A.1 Robot Pose Prediction

To account for the robot's future ego-motion, we need to predict the future pose of the robot at prediction time step $t+n$. Since the constant velocity motion model is the most widely used motion model for tracking [29], we use it as our robot motion model and assume that the robot keeps constant motion in a relatively short period (*i.e.,* less than 1 second). Note that other more suitable robot motion models can be used to provide better robot pose predictions. Then, we can easily predict the future pose of the robot $\mathbf{x}_{t+n}$ at prediction time step $t+n$ using the robot's current pose $\mathbf{x}_t$ and velocity $\mathbf{u}_t$:

$$\begin{bmatrix} x_{t+n} \\ y_{t+n} \\ \theta_{t+n} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_t \cos(\theta_t) \\ v_t \sin(\theta_t) \\ w_t \end{bmatrix} n\Delta t + \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_\theta \end{bmatrix}, \tag{6}$$

where Δt is the sampling interval, $\sigma_\cdot$ is the Gaussian noise.

A.2 Coordinate Transformation

To compensate for the ego-motion of the robot, we first use a homogeneous transformation matrix to transform the robot poses $\mathbf{x}_{t-\tau:t}$ to the robot's local future coordinate frame R:

$$\begin{bmatrix} x^R_{t-\tau:t} \\ y^R_{t-\tau:t} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_{t+n}) & -\sin(\theta_{t+n}) & x_{t+n} \\ \sin(\theta_{t+n}) & \cos(\theta_{t+n}) & y_{t+n} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{t-\tau:t} \\ y_{t-\tau:t} \\ 1 \end{bmatrix}, \tag{7a}$$

$$\theta^R_{t-\tau:t} = \theta_{t-\tau:t} - \theta_{t+n}. \tag{7b}$$

Then, by adding these ego-motion displacements, we convert the observed lidar measurements $\mathbf{y}_{t-\tau:t}$ from Polar to Cartesian coordinates:

$$\mathbf{y}^R_{t-\tau:t} = \begin{bmatrix} x^z_{t-\tau:t} \\ y^z_{t-\tau:t} \end{bmatrix} = \begin{bmatrix} x^R_{t-\tau:t} \\ y^R_{t-\tau:t} \end{bmatrix} + r_{t-\tau:t} \begin{bmatrix} \cos(b_{t-\tau:t} + \theta^R_{t-\tau:t}) \\ \sin(b_{t-\tau:t} + \theta^R_{t-\tau:t}) \end{bmatrix}. \tag{8}$$

Finally, we implement the transformation function (3d) and obtain a set of observed lidar measurements $\mathbf{y}^R_{t-\tau:t}$ at the robot's local future coordinate frame R. The benefit of ego-motion compensation is that we can treat these lidar measurements from a moving lidar sensor as observations from a stationary lidar sensor at R. This significantly reduces the difficulty of OGM predictions and improves accuracy.

A.3 GPU-accelerated Conversion Function $c(\cdot)$

Algorithm 1 shows the pseudo-code for the GPU-accelerated conversion function $c(\cdot)$ (*i.e.,* (3b)) to convert the lidar measurements to the binary occupancy grid maps.

Algorithm 1: Converting lidar points to OGMs

Input: compensated lidar measurements $\mathbf{y}^R_{t-\tau:t}$
Input: grid cell size s
Input: the physical size of the OGMs S
Input: the lower left corner of the OGMs (x_0, y_0)
Output: OGMs $\mathbf{o}_{t-\tau:t}$

initialize: $\mathbf{o}_{t-\tau:t} = 0$
for all parallel beams $z \in \mathbf{y}^R_{t-\tau:t}$ **do**
  $i = \lfloor (x^z_{t-\tau:t} - x_0)/s \rfloor$
  $j = \lfloor (y^z_{t-\tau:t} - y_0)/s \rfloor$
  if $i, j \in [0, S/s]$ **then**
    $\mathbf{o}_{t-\tau:t}(i, j) = 1$
  end if
end for

A.4 GPU-accelerated OGM Mapping Algorithm $g(\cdot)$

Algorithm 2 shows the pseudo code for the GPU-accelerated and parallelized OGM mapping algorithm g(·) (*i.e.,* (3c)) that parallelizes the independent cell state update operation. Note that l_i is the *log odds* representation of occupancy in the occupancy grid map m [20].

Algorithm 2: GPU-accelerated OGM mapping

Input: compensated lidar measurements $\mathbf{y}^R_{t-\tau:t}$
Output: local environment map $\mathbf{m}$

for all time steps n from $t-\tau$ to t **do**
  for all parallel grid cells $\mathbf{m}_i$ in the perceptual field of $\mathbf{y}^R_n$ **do**
    $l_i = l_i + \log \frac{p(\mathbf{m}_i|\mathbf{y}^R_n)}{1-p(\mathbf{m}_i|\mathbf{y}^R_n)} - \log \frac{p(\mathbf{m}_i)}{1-p(\mathbf{m}_i)}$
  end for
end for

A.5 Dataset Collections

A.5.1 OGM-Turtlebot2 Dataset

A simulated Turtlebot2 equipped with a 2D Hokuyo UTM-30LX lidar navigates around an indoor environment with 34 moving pedestrians using random start points and goal points, as shown in Figure 6. The Turtlebot2 uses the dynamic window approach (DWA) planner [28] and has a maximum speed of 0.8 m/s. The Turtlebot2 robot was set up to navigate autonomously in the 3D simulated lobby environment to collect the OGM-Turtlebot2 dataset. The moving pedestrians in the human-robot interaction Gazebo simulator [23] are driven by a microscopic pedestrian crowd simulation library, called the PEDSIM, which uses the social forces model [30, 31] to guide the motion of individual pedestrians:

$$\mathbf{F_p} = \mathbf{F_p^{des}} + \mathbf{F_p^{obs}} + \mathbf{F_p^{per}} + \mathbf{F_p^{rob}}, \tag{9}$$

where $\mathbf{F_p}$ is the resultant force that determines the motion of a pedestrian; $\mathbf{F_p^{des}}$ pulls a pedestrian towards a destination; $\mathbf{F_p^{obs}}$ pushes a pedestrian away from static obstacles; $\mathbf{F_p^{per}}$ models interactions with other pedestrians (*e.g.,* collision avoidance or grouping); and $\mathbf{F_p^{rob}}$ pushes pedestrians away from the robot, modeling the way people would naturally avoid collisions and thereby allowing our control policy to learn this behavior. More details can be found in Xie and Dames [23].

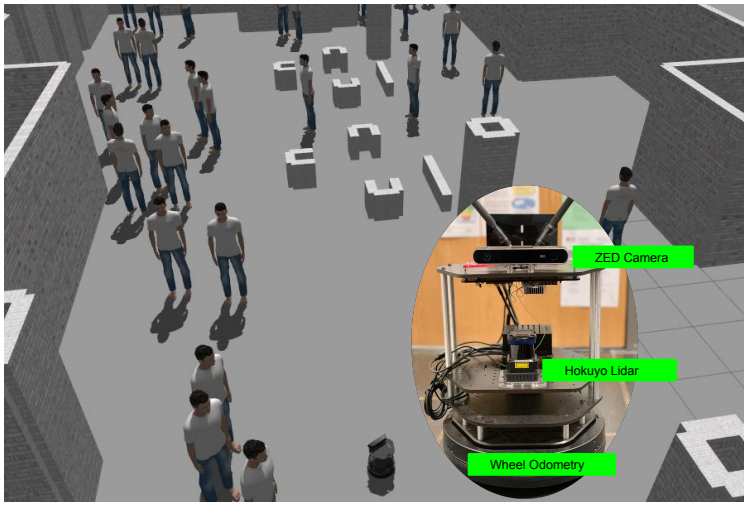

Figure 6: Gazebo simulated environment, where the Turtlebot2 robot was used to collect the OGM-Turtlebot2 dataset.

We collected the robot states $\{\mathbf{x}, \mathbf{u}\}$ and raw lidar measurements $\mathbf{y}$ at a sampling rate of 10 Hz. We collected a total of 94,891 $(\mathbf{x}, \mathbf{u}, \mathbf{y})$ tuples, dividing this into three separate subsets for training (67,000 tuples), validation during training (10,891 tuples), and final testing (17,000 tuples).

A.5.2 OGM-Jackal and OGM-Spot Datasets

Figure 7 shows the real-world outdoor environment at UT Austin and the Jackal robot used to collect the raw SCAND dataset to construct the OGM-Jackal dataset. Figure 8 shows the real-world indoor environment at UT Austin and the Spot robot used to collect the raw SCAND dataset to construct the OGM-Spot dataset. Note that the SCAND dataset was collected by humans manually operating the Jackal robot and the Spot robot around the indoor/outdoor environments at UT Austin. More details can be found in Karnan et al. [22].

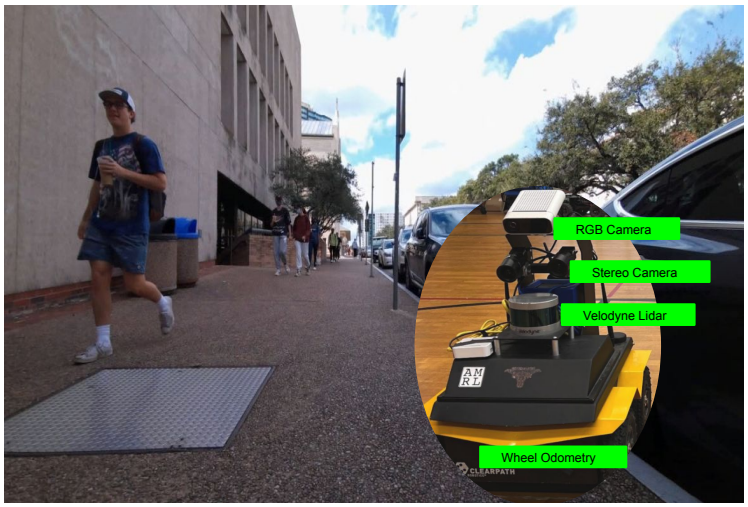

Figure 7: Outdoor environment at UT Austin, where the Jackal robot was used to collect the OGM-Jackal dataset.

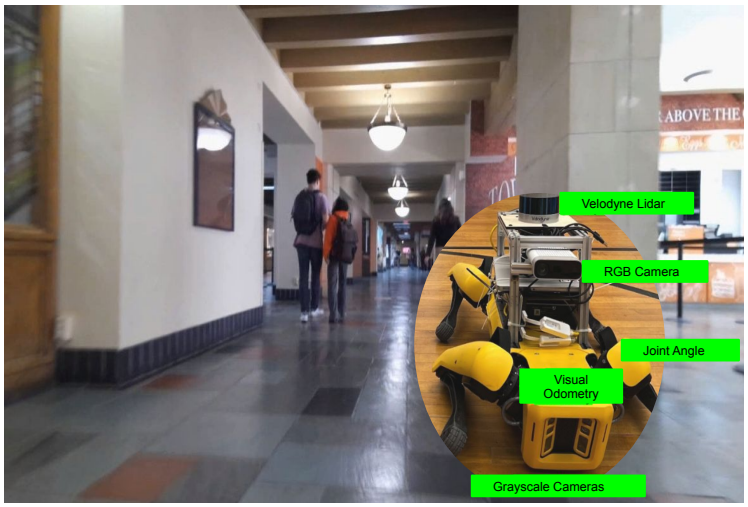

Figure 8: Indoor environment at UT Austin, where the Spot robot was used to collect the OGM-Spot dataset.

A.6 Experiment Details for OGM Prediction

A.6.1 Evaluation Metrics

To comprehensively evaluate the performance of OGM predictors, we define the predicted OGM as $\bar{\mathbf{o}}$ and the ground truth OGM as $\mathbf{o}$, and use the following three metrics:

- **Weighted mean square error (WMSE) [32]**:

$$WMSE = \frac{\sum_{i=1}^{N} w_i \left(\bar{\mathbf{o}}_i - \mathbf{o}_i\right)^2}{\sum_{i=1}^{N} w_i}, \tag{10}$$

  where N is the number of cells in the OGM, and w_i is the weight for the cell i in the OGM, calculated by the median frequency balancing method [33]. This metric is used to evaluate the weighted absolute errors (balancing the imbalance in the percentage of occupied and free cells) between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of single OGM cell.

- **Structural similarity index measure (SSIM)** [34]:

$$SSIM = \frac{\left(2\mu_{\bar{\mathbf{o}}}\mu_{\mathbf{o}} + C_1\right)\left(2\delta_{\bar{\mathbf{o}}\mathbf{o}} + C_2\right)}{\left(\mu_{\bar{\mathbf{o}}}^2 + \mu_{\mathbf{o}}^2 + C_1\right)\left(\delta_{\bar{\mathbf{o}}}^2 + \delta_{\mathbf{o}}^2 + C_2\right)}, \tag{11}$$

  where $\mu_{(\cdot)}$ and $\delta_{(\cdot)}$ denote the mean and variance/covariance, respectively, and $C_{(\cdot)}$ denotes constant parameters to avoid instability. We use $C_1 = 1\text{e}{-4}$ and $C_2 = 9\text{e}{-4}$. This metric is used to evaluate the structural similarity between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of the scene geometry.

- **Optimal subpattern assignment metric (OSPA)** [19]:

$$OSPA = \left(\frac{1}{n}\min_{\pi\in\Pi_n}\sum_{1}^{m} d_c(\bar{\mathbf{o}}_i, \mathbf{o}_{\pi(i)})^p + c^p(n-m)\right)^{\frac{1}{p}}, \tag{12}$$

  where c is the cutoff distance, p is the norm associated to distance, $d_c(\bar{\mathbf{o}}, \mathbf{o}) = \min(c, \|\bar{\mathbf{o}} - \mathbf{o}\|)$, and Π_n is the set of permutations of $\{1, 2, ..., n\}$. We use $c = 10$ (*i.e.,* 1 m) and $p = 1$. This metric is used to evaluate the target tracking accuracy between the predicted OGM and its corresponding ground truth OGM, describing the predicted quality of multi-target localization and assignment.

It is worth noting that while other OGM prediction works [7–11, 15–18] only use the computer vision metrics (*e.g.,* MSE, F1 Score, and SSIM) to evaluate the quality of predicted OGMs, we are the first to evaluate the predicted OGMs from the perspective of multi-target tracking (*i.e.,* OSPA). We believe that since it takes into account multi-target localization error and cardinality error, it can give a more accurate and comprehensive evaluation than only evaluating the image quality.

A.6.2 Evaluation Pipeline for Calculating OSPA Error on OGMs

This evaluation pipeline about how to extract targets from OGMs to calculate their OSPA errors is shown in Figure 9. First, we binarize the predicted OGMs with an occupancy threshold $p_{\text{free}} = 0.3$, which is set by referencing the `occ_thresh` default parameter of 0.25 from the `gmapping` ROS package. Second, we use the density-based spatial clustering of applications with noise (DBSCAN) [35] algorithm to cluster the obstacle points in the OGMs. Finally, we use the mean position of each cluster as the target to calculate the OSPA error (with cutoff distance 10 cells, or 1 m). Note, we get the ground truth target by applying the same process on the ground truth OGMs.

A.7 Experiment Details for Uncertainty Characterization

A.7.1 Evaluation Metrics

To comprehensively characterize the uncertainty information of our SOGMP and SOGMP++ predictors, we define the predicted OGM as $\bar{\mathbf{o}}$ and the ground truth OGM as $\mathbf{o}$ and use the Shannon

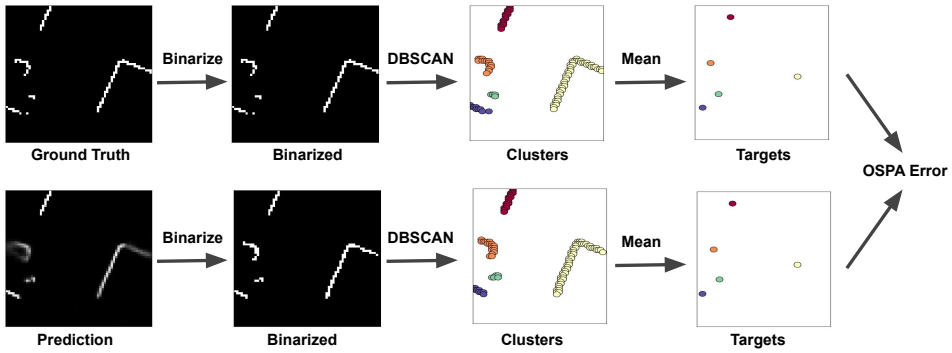

Figure 9: Evaluation pipeline for calculating OSPA error on predicted OGMs.

entropy [36] as the metric:

$$H(\bar{\mathbf{o}}) = \frac{1}{N}\sum_{i=1}^{N}[\bar{\mathbf{o}}_\mathbf{i}\log\bar{\mathbf{o}}_\mathbf{i} + (1-\bar{\mathbf{o}}_\mathbf{i})\log(1-\bar{\mathbf{o}}_\mathbf{i})], \tag{13}$$

where N is the number of cells in the predicted OGM $\bar{\mathbf{o}}$, and $\bar{\mathbf{o}}_\mathbf{i}$ is the value of the cell i in the predicted OGM $\bar{\mathbf{o}}$.

A.7.2 Experiment Setup

Since our SOGMP/SOGMP++ network predicts a bunch of binary OGM samples (*i.e.,* cell value is 0 or 1) rather than a probabilistic OGM, we first combine these binary OGM samples to create a single probabilistic OGM and then use Shannon entropy to characterize its uncertainty. Note that the number of samples we draw can be scaled according to the robot's available computational resources. Before we conduct our uncertainty experiments, we first compute the number of objects in each input sequence of the OGM-Turtlebot test dataset at the 5th prediction time step using the evaluation pipeline shown in A.6.2, where we classify these input sequences into 12 categories according to the number of objects in them. Then, we randomly select 20 input sequences for each number of objects (*i.e.,* from 1 to 12) and generate a total of 1,024 OGM samples for each input sequence at the 5th prediction time step.

To analyze the relationship between the entropy of the predicted OGM and its sample size, we use all selected test sequences and calculate the average entropy over the number of samples growing as an exponential power of 2. To analyze the relationship between the entropy of the predicted OGM and the number of objects in it, we first use 1,024 OGM samples for each input sequence to generate the final probabilistic OGM and then calculate the average entropy over the number of objects from 1 to 12.

A.8 Experiment Details for Robot Navigation

A.8.1 Evaluation Metrics

To comprehensively evaluate the performance of navigation control policies, we use the following four metrics from [23, 26]:

- **Success rate**: the fraction of collision-free trials.
- **Average time**: the average travel time of trials.
- **Average length**: the average trajectory length of trials.
- **Average speed**: the average speed during trials.

A.8.2 Experiment Setup

For the robot navigation experiments, we use the Turtlebot2 robot with a maximum speed of 0.5 m/s, equipped with a Hokuyo UTM-30LX lidar and an NVIDIA Jetson AVG Xavier embedded computer. Considering the computational resources of the Turtlebot2 robot, we use the SOGMP predictor to generate 8 predicted OGM samples at the 6th prediction time step (*i.e.,* 0.6 s). Based on these 8 predicted OGM samples, we generate a prediction map (mean) and an uncertainty map (standard deviation), which are used to generate the prediction costmap layer and uncertainty costmap layer respectively for our predictive uncertainty-aware planner (*i.e.,* DWA-PU), as shown in Figure 5. Note that each costmap grid cell has an initial constant cost, and we map each occupied grid cell of prediction costmap and uncertainty costmap to a Gaussian obstacle value rather than a "lethal" obstacle value. This is because the predicted obstacles and uncertainty regions are not real obstacle spaces.

B Additional Results

B.1 Inference Speed Results

Before we focus on quantitative results on the quality of these OGM predictions, we first talk about the inference speed and model size of these predictors. This is because robots are resource-limited, and smaller model sizes and faster inference speeds mean robots have a faster reaction time to face and handle dangerous situations in complex dynamic scenarios.

Table 2 summarizes the inference speed and model size of six predictors tested on an NVIDIA Jetson TX2 embedded computer. We can see that although DeepTracking [6] has the smallest model size, our proposed SOGMP models are about 1.4 times smaller than the ConvLSTM [12] and 4 times smaller than the PhyDNet [14], and their inference speed is the fastest (up to 24 FPS).

Table 2: Inference Speed and Model Size

Models	ConvLSTM [12]	PhyDNet [14]	DeepTracking [6]	SOGMP_NEMC	SOGMP	SOGMP++
FPS	2.95	4.66	5.32	**24.83**	23.29	10.68
# of Params	12.44 M	37.17 M	**0.95 M**	8.84 M	8.84 M	8.85 M

B.2 Qualitative Prediction Results

Figure 10, Figure 11, Figure 12, and the accompanying Multimedia illustrate the future OGM predictions generated by our proposed predictors and the baselines. We observe two interesting phenomena. First, the image-based baselines, especially the PhyDNet, generate blurry future predictions after 5 time steps, with only blurred shapes of static objects (*i.e.,* walls) and missing dynamic objects (*i.e.,* pedestrians). We believe that this is because these two baselines are deterministic models that only treat time series OGMs as images/video and ignore the kinematics and dynamics of the robot itself, dynamic objects, and static objects. Second, the SOGMP++ with a local environment map has a sharper and more accurate surrounding scene geometry (*i.e.,* right walls) than the SOGMP without a local environment map. This difference indicates that the local environment map for static objects is beneficial and plays a key role in predicting surrounding scene geometry.

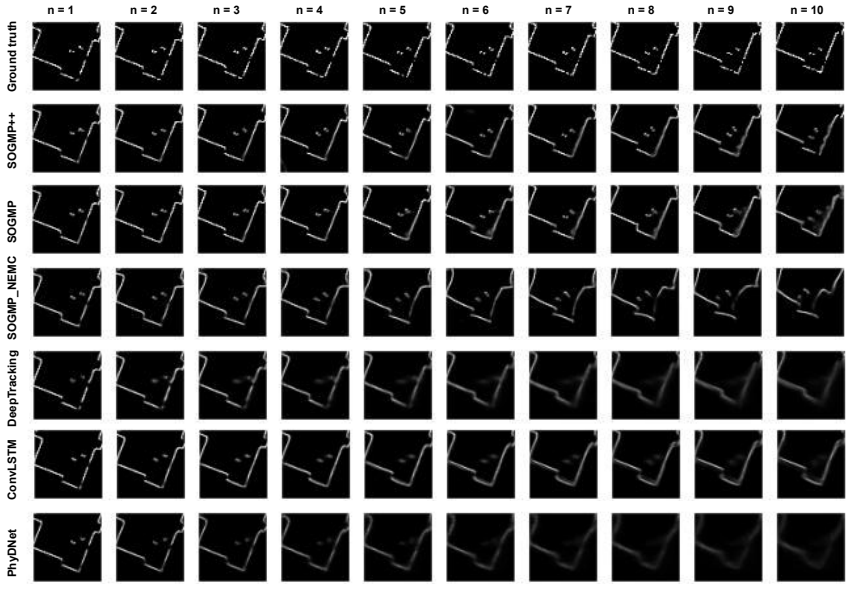

Figure 10: A prediction showcase of the four predictors tested on the OGM-Turtlebot2 dataset over the prediction horizon. The black area is the free space and the white area is the occupied space.

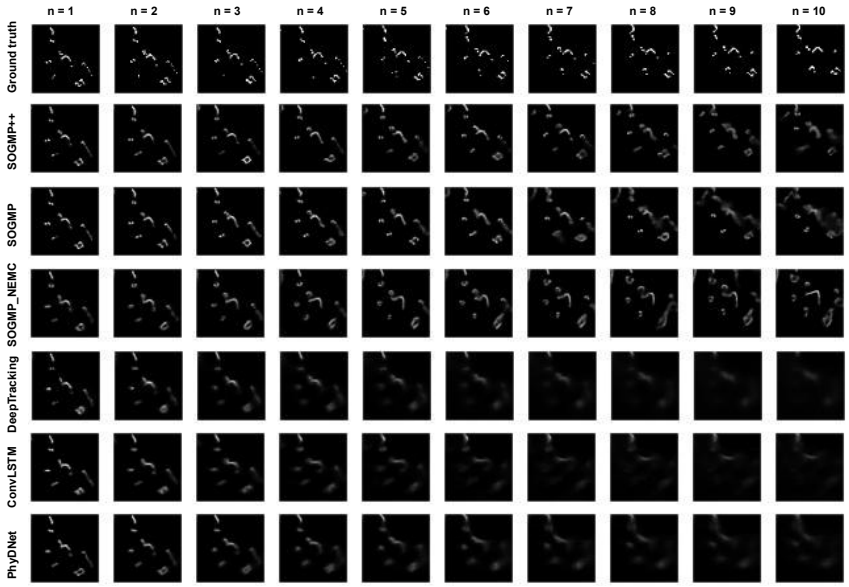

Figure 11: A prediction showcase of the four predictors tested on the OGM-Turtlebot2 dataset over the prediction horizon. The black area is the free space and the white area is the occupied space.

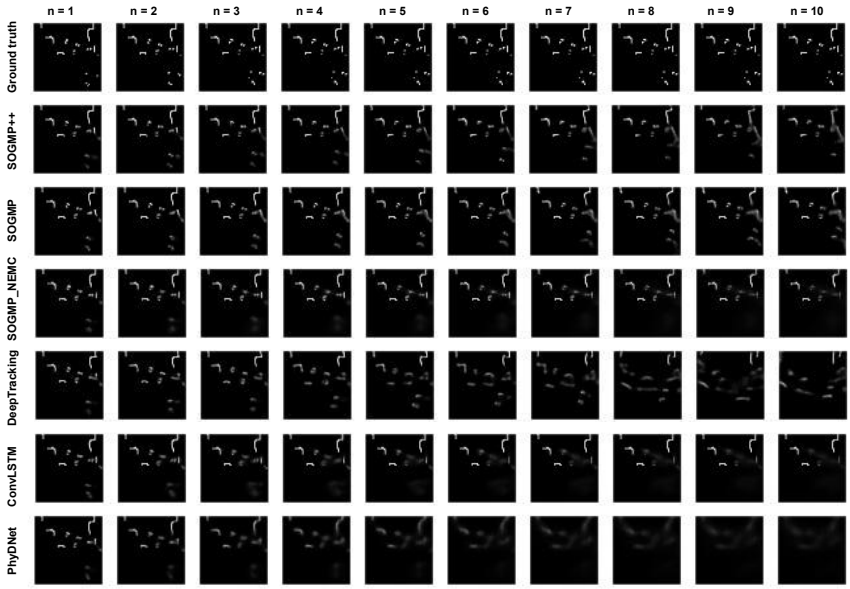

Figure 12: A prediction showcase of the four predictors tested on the OGM-Turtlebot2 dataset over the prediction horizon. The black area is the free space and the white area is the occupied space.

B.3 Qualitative Navigation Results

Figure 13 shows the difference of nominal paths and costmaps generated by three different planners in the simulated lobby environment. The default DWA planner [28] only cares about the current state of the environment and generates a costmap based on the perceived obstacles. The predictive DWA planner (*i.e.,* DWA-P) using the prediction map of our proposed SOGMP predictor can generate a costmap with predicted obstacles. The predictive uncertainty-aware DWA planner (*i.e.,* DWA-PU) using both the prediction map and uncertainty map of our proposed SOGMP predictor can generate a safer costmap with predicted obstacles and uncertainty regions. These additional predicted obstacles and uncertainty regions of our proposed DWA-PU planner enable the robot to follow safer nominal

paths and reduce collisions with obstacles, especially moving pedestrians. See the accompanying Multimedia for a detailed navigation demonstration.

(a) DWA [28] (b) DWA-P (c) DWA-PU

Figure 13: Robot reactions and their corresponding costmaps generated by different control policies in the simulated lobby environment. The robot (black disk) is avoiding pedestrians (colorful square boxes) and reaching the goal (red disk) according to the nominal path (green line) planned by the costmap (square white map).