Deep Samplable Observation Model
for Global Localization and Kidnapping

Runjian Chen[1], Huan Yin[1], Yanmei Jiao[1], Gamini Dissanayake[2], Yue Wang[1] and Rong Xiong[1]

Abstract— Global localization and kidnapping are two challenging problems in robot localization. The popular method, Monte Carlo Localization (MCL) addresses the problem by iteratively updating a set of particles with a "sampling-weighting" loop. Sampling is decisive to the performance of MCL [1]. However, traditional MCL can only sample from a uniform distribution over the state space. Although variants of MCL propose different sampling models, they fail to provide an accurate distribution or generalize across scenes. To better deal with these problems, we present a distribution proposal model named Deep Samplable Observation Model (DSOM). DSOM takes a map and a 2D laser scan as inputs and outputs a conditional multimodal probability distribution of the pose, making the samples more focusing on the regions with higher likelihood. With such samples, the convergence is expected to be more effective and efficient. Considering that the learning-based sampling model may fail to capture the accurate pose sometimes, we furthermore propose the Adaptive Mixture MCL (AdaM MCL), which deploys a trusty mechanism to adaptively select updating mode for each particle to tolerate this situation. Equipped with DSOM, AdaM MCL can achieve more accurate estimation, faster convergence and better scalability than previous methods in both synthetic and real scenes. Even in real environments with long-term changes, AdaM MCL is able to localize the robot using DSOM trained only by simulation observations from a SLAM map or a blueprint map. Source code for this paper is available **here**.

I. INTRODUCTION

The ability to accurately localize a robot is the fundamental requirement for many autonomous tasks, including motion planning, decision making and control [2], [3], [4], [5]. In this paper, we focus on the global localization and kidnapping problem on 2D scenes with 2D laser observation. Given real-time motion information and 2D laser observation, global localization algorithms aim to estimate the pose of the robot in a map of the environment without any prior about the robot pose. For kidnapping, a more complicated problem where the robot is suddenly taken to some other place without being told, the algorithm should be able to detect this situation and recover from it.

Monte Carlo Localization (MCL) [1] is arguably the most popular and efficient algorithm [6] for these two problems. MCL uses a set of particles with weights to represent the estimated probability distribution of the robot pose and iteratively deploys a "sampling-weighting" routine to update

[1]Runjian Chen, Yue Wang, Huan Yin, Yanmei Jiao and Rong Xiong are the State Key Laboratory of Industrial Control Technology and Institute of Cyber-Systems and Control, Zhejiang University, Zhejiang, China. Yue Wang is the corresponding author wangyue@iipc.zju.edu.cn.
[2]Gamini Dissanayake is with the Center for Autonomous System, University of Technology Sydney.

(a) **Traditional MCL**: only uniform distribution.

(b) **Dual MCL**: sample from limited poses with handcrafted features.

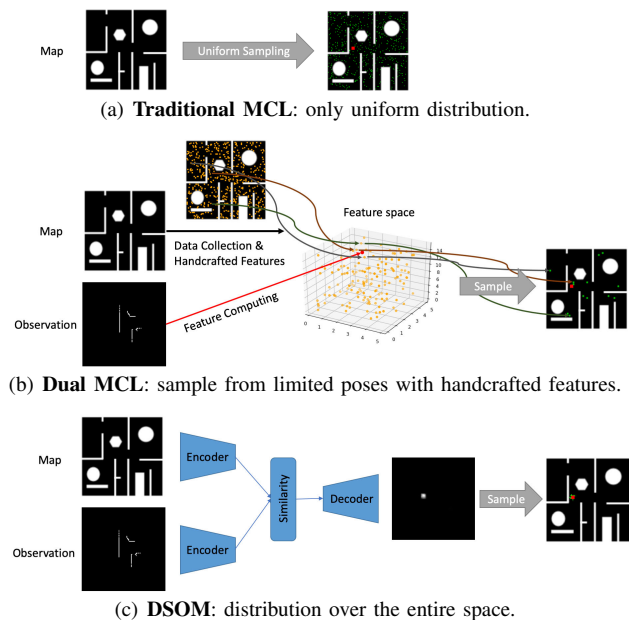(c) **DSOM**: distribution over the entire space.

Fig. 1: Sampling models in MCL, Dual MCL and AdaM MCL.

this set. The essential component of MCL is **sampling** particles from a **proposal distribution** [1].

In global localization and kidnapping problems, traditional MCL can only sample from a uniform distribution over the entire state space. On the one hand, if the particle number is small, there might not be any particle around the accurate pose, resulting in an inaccurate estimation of the robot pose. On the other hand, if the particle number is large enough to cover the whole state space, the computational cost is too high and makes it impossible to finish the update in real-time. To make sampling more effective and efficient, Dual MCL in [1] proposes to sample from observation with three handcrafted features, as shown in Figure 1. Mixture MCL in [1], which combines traditional MCL and Dual MCL by executing the former with probability p or the latter with probability $1 - p$ in an updating iteration, achieves the best performance among variants of MCL [1], [7], [8]. However, these handcrafted features are far from enough to extract meaningful information from observation and it relies on data collection in the same environment where it is deployed, making it unrobust and unable to generalize across scenes.

Recently, many researchers focus on deep learning techniques for providing a proposal distribution [9], [10], [11], [12], [13]. However, the proposal distribution generation in [12] only takes the observation as input and it requires training data collection in the testing environment, which makes it unable to generalize across scenes. [9], [10], [11],

[13] are only able to provide a proposal distribution with a fixed number of probability peaks, which limits their performance because one observation can usually correspond to various numbers of poses where the robot can get similar observations.

To address the problems mentioned above, we propose the Deep Samplable Observation Model (DSOM), which utilizes the feature extraction and generalization ability of the convolutional neural network to efficiently generate a probability map over the entire state space with an adaptive number of probability peaks. Figure 1 describes the difference between sampling models in MCL, Mixture MCL and DSOM. Although the output proposal distribution from DSOM can easily be incorporated in the Mixture MCL framework, the performance is unstable due to the combining mechanism in the original Mixture MCL framework. Thus we further propose Adaptive Mixture MCL (AdaM MCL), which introduces a trusty mechanism to divide the particle set into two parts: highly trusted one and untrusted one, and deploy traditional MCL and dual MCL respectively for these two parts. AdaM MCL is able to generalize across scenes and localize the robot with high accuracy in global localization and kidnapping problems.

II. RELATED WORK

Global localization and kidnapping have long been traditional topics in robotics. From traditional methods built upon probabilistic framework to recent deep learning techniques, we divide algorithms on this topic into the following three categories: (1) traditional techniques (2) deep-sampling-model-aided particle filter (3) differentiable particle filter.

Traditional Techniques. Based on the traditional probabilistic framework, this category is embraced by extended Kalman Filter (EKF) [14], [15] and Monte Carlo Localization (MCL) [1], [16]. EKF depends on a unimodal assumption and is restricted to a specific distribution, i.e. the Gaussian distribution. Thus it is only able to handle the pose tracking problem where the prior about the robot pose (eg. mean and variance) is required. Meanwhile, MCL, also called the particle filter, utilizes a set of particles to represent the multimodal distribution and eliminate the constraint brought by a specific distribution, achieving relatively satisfying results on global localization and kidnapping problems. Among variants of MCL, Mixture MCL [1] is the most successful one. It introduces Dual MCL, which utilizes three handcrafted features (the location of a sensor scan's center of gravity and the average distance measurement) for better sampling particles from observation, as shown in Figure 1. Mixture MCL combines traditional MCL and Dual MCL for better performance. However, the handcrafted features are not informative and data collection in the same scene is required, which is always inapplicable in the real world. Thus Mixture MCL suffers from the lack of generalization ability and unrobustness, especially in highly symmetrical environments.

Deep Sampling Model for Aiding MCL. Works including [9], [10], [11], [17], [18] deploy the deep neural network to provide a proposal distribution, which boosts the performance of MCL. They can be classified into two categories by how many probability peaks they provide. The first one is **unimodal**. Represented by [9], [17], a classifier or a regression network is trained to find the pose with the largest possibility to obtain the observation in a given map, which provides only a unimodal distribution. However, as the environment can be very symmetrical and dynamic, observations at different poses can be very similar and one observation usually corresponds to several poses, for which the unimodal distribution might quickly lose track of the robot pose. The second category is **Top N**. Embraced by [10], [11], [13], [18], this kind of methods train deep models to extract features of the observation and features of templates in the database, with which they are able to propose top N possible regions. This category has two limitations. First, it can only choose the top N regions, where N is a fixed handcrafted parameter. However, in real scenes, one observation might correspond to different numbers of poses and N limits this model's flexibility to approximate the accurate multimodal distribution. Secondly, the output of this kind of model is simply N regions and additional effort should be taken to construct a probability distribution over the state space.

Differentiable Particle Filter. [12], [19], [20] try to make the whole updating process of particle filter differentiable. However, the generation process of proposal distribution in [12] does not take the map as inputs and it is trained on the exact map where they collect testing sequences, making it unable to generalize across different scenes. Also, in [19], [20], the methods do not consider sampling from a deep proposal and are still only able to sample from a uniform distribution at the very beginning. Thus it is doubtful whether these methods are able to cope with kidnapping and there are no experiments on kidnapping in these works.

To briefly summarize, a good sampling model should provide an accurate multimodal probability distribution with an adaptive number of probability peaks. Additionally, generalization ability, low time consumption and scalability over scene sizes are also important requirements. Moreover, as training samples for the sampling model may not cover every possible situation, the predicted probability distribution will occasionally fail to capture the correct pose. The simple combination of traditional MCL and Dual MCL in Mixture MCL may result in losing track of the correct pose if Mixture MCL barely uses the Dual MCL branch when its sampling model fails to capture the correct pose. In this paper, we focus on these two challenges and propose the Deep Sample Observation Model (DSOM) and Adaptive Mixture MCL (AdaM MCL) to address them.

III. DEEP SAMPLABLE OBERVATION MODEL

In this section, we propose our Deep Samplable Observation Model (DSOM). As shown in Figure 2, the inputs of DSOM are two images, one for the environment and one for the observation information (here we plot the 2D laser ranges on an image), represented by two matrices $\mathbf{M}_{env} \in \mathbb{R}^{H_{env} \times W_{env}}$ and $\mathbf{M}_{scan} \in \mathbb{R}^{H_{scan} \times W_{scan}}$. **PM,**
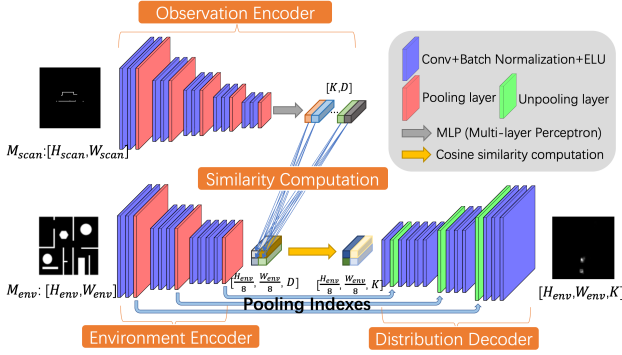
Fig. 2: The architecture of Deep Samplable Observation Model.

the output of DSOM, is the grid approximation of the probability distribution of the pose. We discretize the state space (x, y, θ) by $[H_{env}, W_{env}, K]$. Assuming the environment is x meters in length and y meters in width, the entry (i, j, k) in $\mathbf{PM} \in \mathbb{R}^{H_{env} \times W_{env} \times K}$ stands for the probability that the robot gets the observation $\mathbf{M}_{scan}$ on the exact pose $\left(i \times \frac{x}{H_{env}}, j \times \frac{y}{W_{env}}, k \times \frac{2\pi}{K} \right)$. DSOM consists of four main parts: (1) Observation Encoder (2) Environment Encoder (3) Similarity Computation (4) Distribution Decoder. Section A shows details in these four parts and the loss function. We show the ground truth generation in Section B and in Section C, we explain why DSOM can approximate the multimodal probability distribution.

A. Architecture

Observation Encoder. As described in Equation (1), F_{enc}^{scan} is first deployed to encode $\mathbf{M}_{scan}$ to a feature map and then a Multi-Layer Perceptron (MLP) transforms this feature map to the feature representation $\mathbf{S} \in \mathbb{R}^{K \times D}$. We can see that there are K feature vectors in $\mathbf{S}$, each of which is related to the feature at each discretized rotation angle. As illustrated in Figure 2, F_{enc}^{scan} consists of several encoding blocks, each of which contains convolutional layers, batch normalization layers, non-linear activation layers and pooling layers.

$$\mathbf{S} = MLP \left(F_{enc}^{scan} \left(\mathbf{M}_{scan} \right) \right) \quad (1)$$

Environment Encoder. As Equation (2) shows, F_{enc}^{env}, which is similar to F_{enc}^{scan}, transforms the map of the environment $\mathbf{M}_{env} \in \mathbb{R}^{H_{env} \times W_{env}}$ to its feature map $\mathbf{M} \in \mathbb{R}^{\frac{H_{env}}{8} \times \frac{W_{env}}{8} \times D}$.

$$\mathbf{M} = F_{enc}^{env} \left(\mathbf{M}_{env} \right) \quad (2)$$

Similarity Computation. Cosine similarity is deployed to compute the similarity between $\mathbf{S}$ and $\mathbf{M}$, resulting in the similarity feature map $\mathbf{SIM} \in \mathbb{R}^{\frac{H_{env}}{8} \times \frac{W_{env}}{8} \times K}$. As described in Equation (3), $\mathbf{S}_k \in \mathbb{R}^D$ is the kth feature vector in $\mathbf{S}$, $\mathbf{M}_{i,j} \in \mathbb{R}^D$ is the feature vector on pixel (i, j) on $\mathbf{M}$ and ϵ is a very small number to avoid being divided by zero. $\mathbf{SIM}_{i,j,k}$ indicates the (i, j, k) entry in $\mathbf{SIM}$.

$$\mathbf{SIM}_{i,j,k} = \frac{\mathbf{S}_k{}^T \mathbf{M}_{i,j}}{max \left(\|\mathbf{S}_k\|_2 \cdot \|\mathbf{M}_{i,j}\|_2, \epsilon \right)} \quad (3)$$

Distribution Decoder. In Equation (4), F_{dec} and a softmax activation σ lift $\mathbf{SIM}$ to $\mathbf{PM}$, which stands for the grid approximation of probability distribution over the state space. F_{dec} consists of several decoding blocks, each of which contains convolutional layers, batch normalization layers, non-linear activation layers and an unpooling layer. The unpooling indexes are the same as the pooling indexes in the environment encoder.

$$\mathbf{PM} = \sigma \left(F_{dec} \left(\mathbf{SIM} \right) \right) \quad (4)$$

Loss Function. Kullback-Leibler divergence Loss [21] (KLD Loss) is deployed to guide the training process. A unimodal probability distribution at the exact pose where the robot obtains the observation is computed as the ground truth, $\mathbf{GT}$. Equation (5) illustrates how KLD Loss works, where $\mathbf{GT}_{i,j,k}$ and $\mathbf{PM}_{i,j,k}$ respectively indicate the (i, j, k) entry in $\mathbf{GT}$ and $\mathbf{PM}$.

$$Loss = \sum_{i=1}^{H_{env}} \sum_{j=1}^{W_{env}} \sum_{k=1}^{K} \left[\mathbf{GT}_{i,j,k} \log \frac{\mathbf{GT}_{i,j,k}}{\mathbf{PM}_{i,j,k}} \right] \quad (5)$$

B. Ground Truth Generation

To generate $\mathbf{GT} \in \mathbb{R}^{H_{env} \times W_{env} \times K}$ for a given pose (x, y, θ), we begin with an all-zero matrix for $\mathbf{GT}$. Then a GaussianBlur is deployed on a one-hot ground truth matrix of (x, y), resulting in $\mathbf{GT}' \in \mathbb{R}^{H_{env} \times W_{env}}$. After that, we find the nearest two orientation values (θ_1 and θ_2) to θ in the discretized angle space. Then a linear interpolation is deployed on these two values as below, where I_{θ_1} indicates the discretized index for θ_1 and $\mathbf{GT}_{:,:,I_{\theta_1}}$ are all the entries of which third index is I_{θ_1}

$$\mathbf{GT}_{:,:,I_{\theta_1}} = |\frac{\theta_1 - \theta}{\theta_1 - \theta_2}|$$
$$\mathbf{GT}_{:,:,I_{\theta_2}} = 1 - \mathbf{GT}_{:,:,I_{\theta_1}} \quad (6)$$

We aggregate both the position and orientation ground truth as described in Equation (7) and then a normalization is deployed on $\mathbf{GT}$ leading to the final ground truth. $\mathbf{GT}_{i,j,k}$ is the value on the entry (i,j,k) of $\mathbf{GT}$.

$$\mathbf{GT}_{i,j,k} = \mathbf{GT}_{i,j,k} \cdot \mathbf{GT}'_{i,j} \quad (7)$$

C. Multimodal Effect in DSOM

While there might exist a set of similar observations $\{\mathbf{M}_{scan}^l | l = 1, 2, ..., m\}$ obtained at a set of different states $\{s^l | l = 1, 2, ..., m\}$ in the training set, the ground truth PM is only unimodal distribution. Why can the unimodal ground truth guide the network to generate a multimodal probability distribution? We here prove the case in which $\{\mathbf{GT}^l | l = 1, 2, ..., m\}$ are one-hot distribution, as described in Equation (8) where $\mathbf{GT}_{i,j,k}^l$ is the $\{i, j, k\}$ entry in $\mathbf{GT}^l$, i_{gt}^l, j_{gt}^l, k_{gt}^l are respectively the ground truth entry for position and orientation. We further assume that all the $\mathbf{M}_{scan}^l$ are the same for simplicity. The conclusion is that the sum of the KLD Losses of these m samples is minimized only when the output $\mathbf{PM}$ is an even-distributed multimodal distribution on these m poses.

$$\Delta(i,j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{8}$$
$$\mathbf{GT}_{i,j,k}^l = \Delta\left(i, i_{gt}^l\right) \cdot \Delta\left(j, j_{gt}^l\right) \cdot \Delta\left(k, k_{gt}^l\right)$$

Proof. As $\mathbf{M}_{scan}^l$ are the same, all the $\mathbf{S}$ generated by observation encoder are identical while environment encoder obtains the same feature map $\mathbf{M}$. Thus all the $\mathbf{PM}^l$ are the same, denoted as $\mathbf{PM}$. The sum of losses is described in Equation (9), where $\mathbf{PM}_{s^l}$ indicates value on the entry s^l of $\mathbf{PM}$.

$$Loss = \sum_{l=1}^{m} \left\{ \sum_{i=1}^{H_{env}} \sum_{j=1}^{W_{env}} \sum_{k=1}^{K} \left[\mathbf{GT}_{i,j,k}^l \log \frac{\mathbf{GT}_{i,j,k}}{\mathbf{PM}_{i,j,k}} \right] \right\}$$
$$= \sum_{l=1}^{m} - \log \mathbf{PM}_{s^l} \tag{9}$$

Thus the problem turns to:

$$\underset{\mathbf{PM}}{\text{minimize}} \quad \sum_{l=1}^{m} - \log \mathbf{PM}_{s^l}$$
$$\text{subject to} \quad \sum_{s} \mathbf{PM}_s = 1 \tag{10}$$

After applying Lagrange multiplier to it, the problem turns to minimize the term below:

$$\underset{\mathbf{PM},\lambda}{\text{minimize}} \quad L = \sum_{l=1}^{m} - \log \mathbf{PM}_{s^l} + \lambda(1 - \sum_{s} \mathbf{PM}_s) \tag{11}$$

Let the derivative of L with respect to every entry in $\mathbf{PM}$ and λ be 0, we get:

$$\frac{1}{\lambda} = \mathbf{PM}_{s^1} = \mathbf{PM}_{s^2} = ... = \mathbf{PM}_{s^m}$$
$$\mathbf{PM}_s = 0, \quad \forall s \notin \{s^l | l = 1, 2, ..., m\} \tag{12}$$

Also with $\sum_{s} \mathbf{PM}_s = 1$, we can get that only when Equation (13) is satisfied, can the sum of these m KLD Losses get minimized. Proof complete. ∎

$$\mathbf{PM}_{s^1} = \mathbf{PM}_{s^2} = ... = \mathbf{PM}_{s^m} = \frac{1}{m} \tag{13}$$

As training samples can cover part of the state space, it is obvious that the KLD Loss can guide the network to approximate the multimodal distribution.

IV. ADAPTIVE MIXTURE MCL

As the training samples cannot cover all the possible cases, $\mathbf{PM}$ sometimes fails to approximate the accurate probability distribution. The combination of traditional MCL and Dual MCL with DSOM provides a solution. However, the combination in [1], which deploys either MCL with probability p or Dual MCL with probability $1 - p$ in one iteration, is too simple. If the proposal distribution fails to capture the correct poses at some iteration and the algorithm happens to deploy Dual MCL, the algorithm will lose track of

the robot pose. To compensate for this problem, we propose a novel filter, Adaptive Mixture MCL (AdaM MCL).

In AdaM MCL, each particle has three elements $\left(s^i, w_{norm}^i, w^i\right)$, where the first one indicates the pose of this particle, w^i is the weight before normalization and w_{norm}^i is the normalized weight. In every updating iteration, we first evaluate the degree we trust a particle by computing $T = \frac{w^i}{w_*}$, where w_* is the output of the weighting model when given zero-difference in every direction as input. Please refer to Section I in Appendix for the detailed equation for w_*. According to T, we divide the whole set into $\mathcal{H}$ and $\mathcal{L}$. As there are obstacles around the robot, w_i is never equal to w_* even when the particle is at the same pose as that of the robot. To compensate for this situation, we introduce a threshold parameter $T_{cut} \in (0, 1)$ and if $T > T_{cut}$, we directly add this particle to $\mathcal{H}$. If not, this particle is added to $\mathcal{H}$ with a probability of T or $\mathcal{L}$ with a probability of $1 - T$. Finally, we deploy traditional MCL update for $\mathcal{H}$ and Dual MCL update with $\mathbf{PM}$ for $\mathcal{L}$. Algorithm 1 in Section IV in the Appendix shows how AdaM MCL works.

V. EXPERIMENTS AND RESULTS

In this section, we conduct experiments in synthetic and real scenes to investigate the ability of DSOM to approximate the multimodal distribution and generalize across scenes. Also, we compare AdaM MCL with previous methods on global localization and kidnapping problems.

A. Experiments Setup

Datasets. We conduct experiments in both synthetic and real environments.

- *Synthetic Environments.* We manually create 20 maps (43 meters in both length and width). We collect training data in 18 of them and testing sequences in the other 2 maps. To train DSOM, we add random obstacles in the environments and simulate laser to get the observation. What we feed into DSOM is the original map without obstacles and the observation. For the testing sequential data, we put unknown obstacles or remove part of the environment along the path.

- *Real Environments.* We conduct experiments on two benchmark real-world datasets: the Royal Alcazar of Seville dataset (UPO) (150 meters in length and 100 meters in width) [22] and the Rawseeds indoor dataset collected in the University di Milano-Bicocca (Bicocca) [23], [24]. For the Bicocca dataset, we divide the whole scene into two smaller parts (40 meters in length and 30 meters in width) for experiments. To train DSOM, we add random obstacles to the map given by these datasets (SLAM-based or floorplan) and collect training data using simulating laser. With these synthetic data, we train DSOM, after which it is applied to real observation in the same scene. A map of the environment where the UPO dataset is collected is shown in Figure 4.

Baselines. There are four baselines in our experiments.

- *MCL* with random sampling [1].

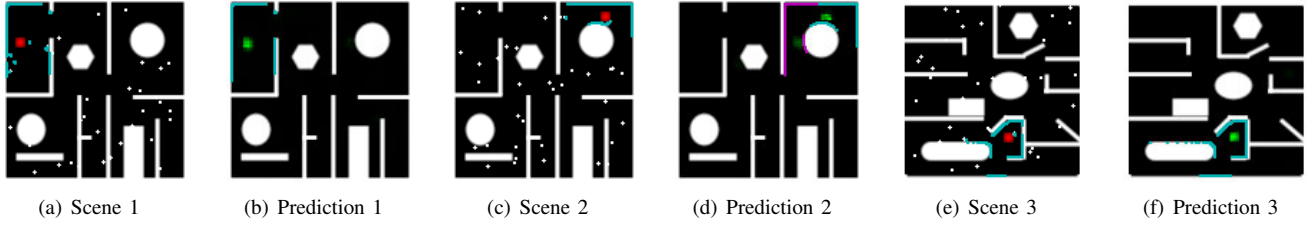| (a) Scene 1 | (b) Prediction 1 | (c) Scene 2 | (d) Prediction 2 | (e) Scene 3 | (f) Prediction 3 |

Fig. 3: Visualization results on the output of DSOM. As shown in (a)(c)(e), observation with obstacles around is indicated by blue lines. Red squares in (a)(c)(e) indicate the ground truth position. The input of DSOM are observation in (a)(c)(e) and a map without obstacles. Green clusters in (b)(d)(f) show the predicted probability peaks. Blue and purple lines in (b)(d)(f) show the observation on the predicted probability peaks.
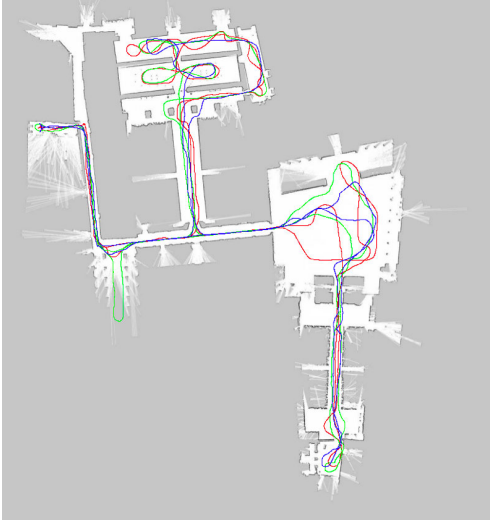

Fig. 4: The map of UPO dataset [22] and multiple trajectories in this dataset. We randomly select three testing clips among them.

- *Mixture MCL* [1] whose sampling model is trained by data collected in the testing scene with random obstacles.
- *Mixture MCL** whose sampling model is trained by data containing the testing sequences except for the exact pose at a given timestamp.
- *PFRNN* [19][1] the most recent and effective deep-learning-based particle filter.
- *PF-Net* [20]: the particle-filter network.

For *MCL* and *Mixture MCL*, we follow the implementation in [25], [26] and also try our best to tune the parameters in these two baselines for better performance than the original parameter set in [25], [26]. For *Mixture MCL**, we note here that the sampling model is "cheating" because an algorithm should not have any prior about the testing sequences. PFRNN and PF-Net are trained by data sequences collected in the same environment where they are tested. The random sampling rate of traditional MCL is set to 0.2. Particle number is set to 200 for all algorithms in synthetic datasets and 500 for all real datasets.

DSOM and AdaM MCL. The network structure in DSOM, as shown in Section II in Appendix, remains the same in different experiments (different environment sizes) to demonstrate its scalability. T_{cut} is set to 0.6 for all synthetic environments and the UPO dataset. For the Bicocca dataset,

[1]We use the implementation from the github repository of the author: https://github.com/Yusufma03/pfrnns

T_{cut} is set to 0.5 because the environment is more dynamic than others.

About the GPU Memory Consumption and Time Consumption: As the maps of the environments grow, the GPU memory consumption becomes larger. To make a fair comparison of the three deep-learning-based methods, we limit the available GPU to only one TITAN GPU with 12 Gigabytes memory. By scaling the resolution of the map, we can train and evaluate the models with limited resources. And typically, inference for one iteration in DSOM requires only 0.02 seconds while PFRNN and PF-Net require only 0.08 seconds, all of which are negligible. Thus, we will use the number of updating iterations before convergence to compare the converging speed of all the methods.

Evaluation Metric. For each testing sequence and method, we run experiments for 100 times. Then the error of the estimated position $(x_{est}, y_{est}, \theta_{est})$ is computed at each updating iteration: $E_{pos} = \sqrt{(x_{est} - x_{gt})^2 + (y_{est} - y_{gt})^2}$ and $E_\theta = \sqrt{(\theta_{est} - \theta_{gt})^2}$. Thus at each updating iteration on each testing sequence, we have 100 error values for each method. To show the converging process of each method, we compute the mean value as well as 95% confidence interval of the 100 error values E_{pos} at each updating step and plot the evolution of estimation error of all methods on the same chart to compare their performance. Because each update iteration in all the methods we use can be finished in real-time, we care more about how long the robot travels before the algorithm converges than the time needed to converge. To evaluate how fast and stable an algorithm converges, we first set a condition for converging to the accurate pose: If E_{pos} is lower than 2 meters and the error in orientation E_θ is lower than 10 degrees for 5 consecutive updating steps, we consider that the algorithm converges to the accurate pose. If an algorithm converges, $STEPS$ is computed as the number of steps the algorithm requires to converge.

B. Model Validation

Case Study of the multimodal effect in DSOM. In Figure 3, it can be found that DSOM is able to take the observation and map without obstacles as inputs and generate various numbers of probability peaks, including the ground truth, which demonstrates DSOM as a good sampling model. A visualization of the output of DSOM on Bicocca dataset can be found in Appendix III.
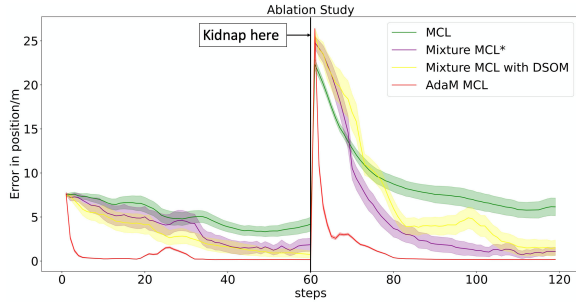
Fig. 5: Results on the ablation study experiment.

TABLE I: Parameter sensitivity experiment on T_{cut}

| T_{cut} | Global Localization | | Kidnapping | |
	E_{pos}/m	E_θ/rad	E_{pos}/m	E_θ/rad
0.1	0.299	0.017	0.188	0.031
0.3	0.190	0.019	0.150	0.021
0.5	0.163	0.018	0.169	0.018
0.7	0.155	0.015	0.286	0.053
0.9	0.725	0.023	1.133	0.547

TABLE II: Parameter sensitivity experiment on particle number.

| Particle Number | Global Localization | | Kidnapping | |
	E_{pos}/m	E_θ/rad	E_{pos}/m	E_θ/rad
200	0.153	0.022	0.128	0.022
500	0.088	0.017	0.089	0.016
1000	0.060	0.016	0.072	0.007

Ablation Study. We compare four methods in a testing synthetic environment: MCL, Mixture MCL*, Mixture MCL with DSOM as the sampling model and AdaM MCL. The results for global localization and kidnapping with these four methods are shown in Figure 5. It can be found that Mixture MCL with DSOM achieves slightly better performance in the global localization problem while Mixture MCL* outperforms a bit in the kidnapping problem. The only difference between them lies in the sampling model and as discussed previously, the sampling model in Mixture MCL* is trained by testing data while DSOM is trained by data collected in other environments. This demonstrates that DSOM can generalize across synthetic environments and provide an accurate probability distribution over the state space for sampling (similar performance to a "cheating" sampling model). Also, it is found that AdaM MCL converges faster and achieves a more accurate estimation than Mixture MCL with DSOM, demonstrating the effectiveness of AdaM MCL.

Parameter Sensitivity Experiment. We conduct parameter sensitivity experiments on two important parameters in AdaM MCL: T_{cut} and the number of particles while other parameters are fixed. Results are shown in Table I (the number of particles is set to 200) and II (T_{cut} is set to 0.6). It can be found that the estimation accuracy is relatively stable when the two parameters vary, which demonstrates that AdaM MCL is not sensitive to parameter settings.

C. Comparisons to Baselines in Global Localization and Kidnapping

Train in some synthetic maps and test in "unseen" synthetic maps. Six sequences of synthetic data are col-
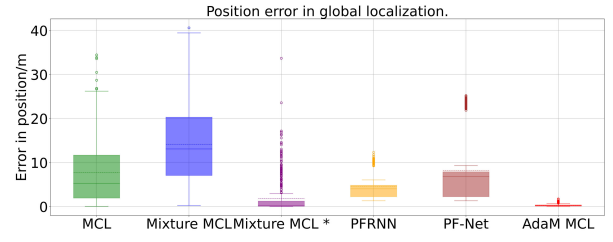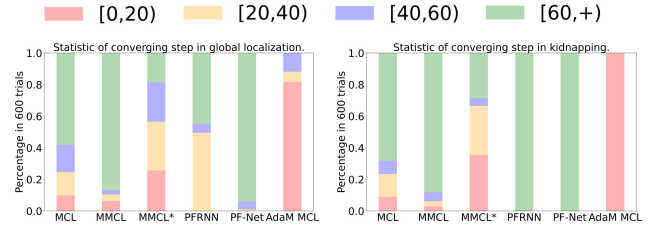

(a) Global Localization.
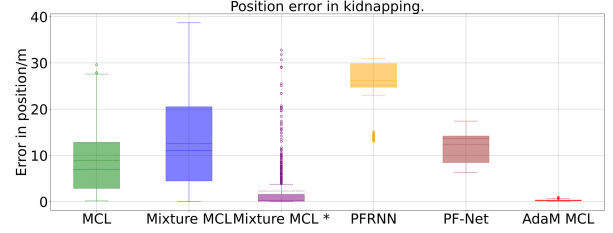
(b) Kidnapping

Fig. 6: Results of E_{pos} in synthetic environments.

(b) Global Localization.

(c) Kidnapping.

Fig. 7: Results for converging steps among 600 synthetic experiments. Red indicates the percentage that an algorithm converges in 0 to 20 steps among 600 experiments and it is similar for other colors. MMCL stands for Mixture MCL.

lected in two synthetic maps to evaluate the performance of the three methods, leading to 600 experiments for each method. For each sequence of testing data, we first start from global localization and then kidnap the robot for one time. Results for converging accuracy, i.e. E_{pos}, are "boxplotted" in Figure 6. For converging steps, we divide the statistic into 4 intervals and compute the percentage that each method converges within the respective interval. A normalized histogram about converging steps is shown in Figure 7.

It can be found that AdaM MCL achieves significantly better estimation results than all the baselines. Also, AdaM MCL achieves the highest converging rate and requires the least steps to converge. Note that DSOM is only trained by data collected in other environments while the sampling model in Mixture MCL* is trained by testing data and PFRNN as well as PF-Net is trained by data collected in the testing environments. These demonstrate that DSOM is able to generalize across different synthetic environments and AdaM MCL can accurately localize the robot in "unseen" synthetic scenes.

Train with simulation observation on SLAM-based or floorplan maps and test on real observation. As shown in

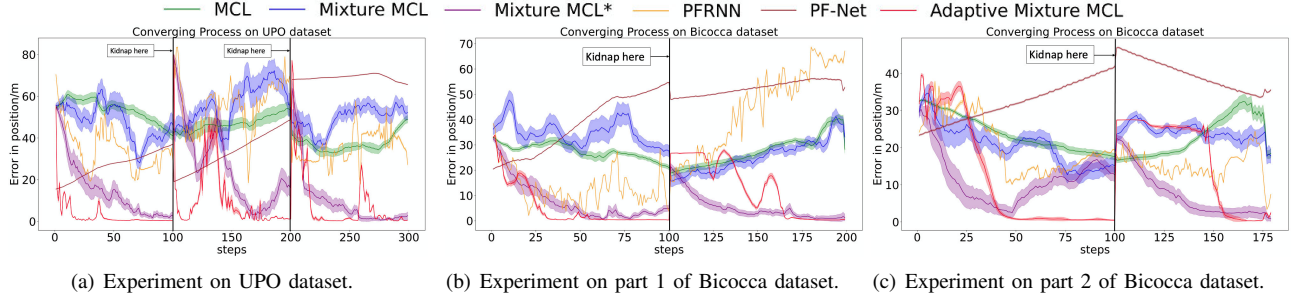| (a) Experiment on UPO dataset. | (b) Experiment on part 1 of Bicocca dataset. | (c) Experiment on part 2 of Bicocca dataset. |

Fig. 8: Results on real datasets.

Figure 8, it can be found that in real scenes, AdaM MCL with DSOM, which is trained by synthetic laser range data in the given blueprint or SLAM-based map, outperforms all the baseline on estimation accuracy, even Mixture MCL* whose sampling model is trained by testing data. These results indicate that DSOM can learn from synthetic laser data in the map of a real environment (SLAM-based or blueprint) and generalize to real observation, which helps AdaM MCL attain accurate estimation of the robot pose in real environments. This is very promising because with a SLAM-based or blueprint map, which is always available, we can quickly generate simulation observations to train DSOM and then deploy AdaM MCL with DSOM to accurately localize the robot in the real scenes.

VI. CONCLUSION

In this paper, we propose a samplable deep multimodal observation model, DSOM, as well as AdaM MCL, which is a novel localization filter for global localization and kidnapping problems. Our extensive experiments demonstrate that DSOM is able to generalize across different synthetic environments. Moreover, it can generalize to real environments when trained on only simulated observations from SLAM-based or even blueprint maps. Experiment results in both synthetic and real environments show that AdaM MCL equipped with DSOM achieves superior performance than previous methods on both synthetic and real datasets in global localization and kidnapping problems.

In the future, there are two interesting directions that might stem from this work. The first is to transfer models trained in synthetic environments to real environments, whose difficulties stem from the considerable difference between synthetic and real environments. This can be achieved by several possible methods: retraining models on real datasets, which might accelerate the training process, or finding a suitable way to augment synthetic data to make it possible to directly transfer models trained on synthetic data to real data. The second direction is to extend the work to localization in 3D space with 3D observation information like point clouds or RGBD images. Recent deep learning techniques for processing 3D information such as [27] can be incorporated into DSOM to extract robust features from both 3D maps and 3D observations. Then we can follow the same pipeline to decode a probability distribution for sampling poses.

APPENDIX I
TRANSITION AND OBSERVATION FUNCTION

Given a particle $\left(s_{t-1}^i, w_{t-1}^i\right)$, the motion a_{t-1} and the observation o_t, the transition model is described in Equation (14), where σ is the variance of motion infomation.

$$s_t^i = s_{t-1}^i + a_{t-1} + N(0, \sigma) \tag{14}$$

Observation function is described in Equation (15), where **D** is the number of directions a 2D lidar perceives, $(L_o^i$ and $(L_p^i$ are the range in the ith direction of the real observation and the observation obtained at the pose of particle p on the map. σ is the variance of the lidar range finder and L_{max} is the maximum range that the lidar can return. α and β are parameters set as 0.9 and 0.1 respectively.

$$\text{Score}(L_o, L_p) = \sum_{i=1}^{\mathbf{D}} \left[\alpha \times \frac{e^{-\frac{\left(L_o^i - L_p^i\right)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} + \beta \times \frac{1}{L_{\max}} \right]^3 \tag{15}$$

w_* is the output of the weighting model when given zero-difference in every direction as inputs, that is $w_* = \text{Score}(L, L)$, where $L = L_o = L_p$

APPENDIX II
DETAILED SETTINGS IN DSOM
TABLE III: Detailed Settings in DSOM

	Stride	Filters	Size		Stride	Filters	Size
Observation Encoder	2	64	(3,3)	Distribution Decoder	2	256	(3,3)
	2	64	(3,3)		2	256	(3,3)
	2	64	(3,3)		2	256	(3,3)
	2	64	(3,3)		2	256	(3,3)
	2	128	(3,3)		2	128	(3,3)
	2	128	(3,3)		2	128	(3,3)
	2	256	(3,3)		2	128	(3,3)
	2	256	(3,3)		2	64	(3,3)
	2	256	(3,3)		2	64	(3,3)
	2	256	(3,3)		2	32	(3,3)

	Stride	Filters	Size
Env. Encoder	2	64	(3,3)
	2	64	(3,3)
	2	128	(3,3)
	2	128	(3,3)
	2	128	(3,3)
	2	256	(3,3)
	2	256	(3,3)
	2	256	(3,3)
	2	256	(3,3)

	Input	Output
MLP	16	32
	32	32
	32	32

APPENDIX III
THE OUTPUT OF DSOM ON BICOCCA DATASET.

(a) Scene   (b) Prediction

Fig. 9: Visualization of the output of DSOM on Bicocca dataset.

APPENDIX IV

Algorithm 1: Adaptive Mixture MCL

Input: $\mathcal{P}_{t-1}$, o_t, a_{t-1}, w_*, $\mathbf{M}_{env}$, T_{cut}

Output: $\mathcal{P}_t$

1 **begin**

2    $\mathbf{PM} \leftarrow \mathrm{DSOM}(o_t, \mathbf{M}_{env})$; $\mathcal{P}_t$, $\mathcal{H}$, $\mathcal{L} \leftarrow \emptyset$

3   **for** $p^i = \left(s^i, w^i_{norm}, w^i\right) \in \mathbf{P}_{t-1}$, $T_i = \frac{w^i}{w_*}$ **do**

4   **if** $T_i > T_{cut}$ **then**

5        $\mathcal{H} \leftarrow \mathcal{H} \cup \{p^i\}$

6   **else**

7   with probability T_i: $\mathcal{H} \leftarrow \mathcal{H} \cup \{p^i\}$

8   with probability $(1 - T_i)$: $\mathcal{L} \leftarrow \mathcal{L} \cup \{p^i\}$

9   **end**

10   **end**

11   **for** $i = 1$ to $\|\mathcal{H}\|$ **do**

12   sample h from $\mathcal{H}$ according to w_{norm}

13   sample $h' \sim p\left(h'|a_{t-1}, h\right)$ & $w_{h'} = p\left(o_t|h'\right)$

14      $\mathcal{P}_t \leftarrow \mathcal{P}_t \cup \left(h', w_{h'}\right)$

15   **end**

16   **for** $i = 1$ to $\|\mathcal{L}\|$ **do**

17   sample l' according to $\mathbf{PM}$

18   sample $l \sim p\left(l'|a_{t-1}, l\right)$ & $w_{l'} = w_l$

19      $\mathcal{P}_t \leftarrow \mathcal{P}_t \cup \left(l', w_{l'}\right)$

20   **end**

21    $w_{norm} \leftarrow$ normalize $w \in \mathcal{P}_t$ & return $\mathcal{P}_t$

22 **end**

REFERENCES

[1] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[2] I. J. Cox, "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Transactions on robotics and automation*, vol. 7, no. 2, pp. 193–204, 1991.

[3] T. Fukuda, S. Ito, N. Oota, F. Arai, Y. Abe, K. Tanaka, and Y. Tanaka, "Navigation system based on ceiling landmark recognition for autonomous mobile robot," in *Proceedings of IECON'93-19th Annual Conference of IEEE Industrial Electronics*. IEEE, 1993, pp. 1466–1471.

[4] R. Hinkel and T. Knieriemen, "Environment perception with a laser radar in a fast moving robot," in *Robot Control 1988 (Syroco'88)*. Elsevier, 1989, pp. 271–277.

[5] J. J. Leonard, H. F. Durrant-Whyte, and I. J. Cox, "Dynamic map building for an autonomous mobile robot," *The International Journal of Robotics Research*, vol. 11, no. 4, pp. 286–298, 1992.

[6] C. Stachniss and W. Burgard, "Particle Filters for Robot Navigation," *Foundations and trends in Robotics*, vol. 3, no. 4, pp. 211–282, 2014. [Online]. Available: http://www.qjps.com/article/Download/ROB-013

[7] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "An adaptive color-based particle filter," *Image and vision computing*, vol. 21, no. 1, pp. 99–110, 2003.

[8] K. Okuma, A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe, "A boosted particle filter: Multitarget detection and tracking," in *European conference on computer vision*. Springer, 2004, pp. 28–39.

[9] G. Kim, B. Park, and A. Kim, "1-day learning, 1-year localization: Long-term lidar localization using scan context image," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1948–1955, 2019.

[10] S. Xu, W. Chou, and H. Dong, "A robust indoor localization system integrating visual localization aided by cnn-based image retrieval with monte carlo localization," *Sensors*, vol. 19, no. 2, p. 249, 2019.

[11] M. Angelina Uy and G. Hee Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4470–4479.

[12] R. Jonschkowski, D. Rastogi, and O. Brock, "Differentiable particle filters: End-to-end learning with algorithmic priors," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[13] H. Yin, L. Tang, X. Ding, Y. Wang, and R. Xiong, "Locnet: Global localization in 3d point clouds for mobile vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 728–733.

[14] G. Welch and G. Bishop, "An introduction to the kalman filter," USA, Tech. Rep., 1995.

[15] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–193.

[16] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[17] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[18] H. Yin, Y. Wang, X. Ding, L. Tang, S. Huang, and R. Xiong, "3d lidar-based global localization using siamese neural network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1380–1392, 2019.

[19] X. Ma, P. Karkus, D. Hsu, and W. S. Lee, "Particle filter recurrent neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 5101–5108, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5952

[20] P. Karkus, D. Hsu, and W. S. Lee, "Particle filter networks with application to visual localization," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 169–178. [Online]. Available: http://proceedings.mlr.press/v87/karkus18a.html

[21] T. Van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.

[22] R. Ramón-Vigo, J. Pérez, F. Caballero, and L. Merino, "Navigating among people in crowded environment: Datasets for localization and human robot interaction," in *Proceedings of the Workshop on Robots in Clutter: Perception and Interaction in Clutter, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Citeseer, 2014.

[23] S. Ceriani, G. Fontana, A. Giusti, D. Marzorati, M. Matteucci, D. Migliore, D. Rizzi, D. G. Sorrenti, and P. Taddei, "Rawseeds ground truth collection systems for indoor self-localization and mapping," *Autonomous Robots*, vol. 27, no. 4, p. 353, 2009.

[24] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. G. Sorrenti, and J. D. Tardos, "Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets," in *In proceedings of IROS*, vol. 6, 2006, p. 93.

[25] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[26] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3. Kobe, Japan, 2009, p. 5.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, pp. 5099–5108, 2017.