Continuous Occupancy Mapping in Dynamic Environments Using Particles

Gang Chen, Wei Dong, Peng Peng, Javier Alonso-Mora, and Xiangyang Zhu

***Abstract*—Particle-based dynamic occupancy maps were proposed in recent years to model the obstacles in dynamic environments. Current particle-based maps describe the occupancy status in discrete grid form and suffer from the grid size problem, wherein a large grid size is unfavorable for motion planning while a small grid size lowers efficiency and causes gaps and inconsistencies. To tackle this problem, this paper generalizes the particle-based map into continuous space and builds an efficient 3D egocentric local map. A dual-structure subspace division paradigm, composed of a voxel subspace division and a novel pyramid-like subspace division, is proposed to propagate particles and update the map efficiently with the consideration of occlusions. The occupancy status at an arbitrary point in the map space can then be estimated with the weights of the particles. To reduce the noise in modeling static and dynamic obstacles simultaneously, an initial velocity estimation approach and a mixture model are utilized. Experimental results show that our map can effectively and efficiently model both dynamic obstacles and static obstacles. Compared to the state-of-the-art grid-form particle-based map, our map enables continuous occupancy estimation and substantially improves the mapping performance at different resolutions.**

***Index Terms*—Mapping, Aerial Systems: Perception and Autonomy, Collision Avoidance, Dynamic Environment**

I. Introduction

THE particle-based map is originally proposed in [1] for dynamic and unstructured environments. Particles with position and velocity states are used to approximate both dynamic obstacles and static obstacles on the basis of sequential Monte Carlo (SMC) filtering. In recent works, [2] introduces the theory of random finite set (RFS) to particle-based maps. The probability hypothesis density (PHD) filter is applied to predict and update the particles and estimate the dynamics of the grids in the map. Later, [3]–[5] improve the particle-based maps by considering the mixture model, semantic information and high-level occupancy status inference, respectively. Due to the ability to model complex-shaped static and dynamic obstacles simultaneously, particle-based maps draw more attention in representing dynamic environments. Currently, the input form of particle-based maps is the ray-casting-generated measurement grid map originated from the first work [1], and thus the map is discretized with grids. This discrete form inhibits the state estimation resolution and brings the grid size problem, namely: large grids lead to a low resolution that is unfavorable for motion planning, while small grids increase the computation requirements and may cause gaps and inconsistencies [6]. Besides, desktop GPUs are required to run the particle-based maps in real-time, and a more efficient map is needed for applications in small-scale robotic systems.

This work proposes a dual-structure particle-based (DSP) map, a continuous dynamic occupancy map free from the grid size problem. The input of the map is the raw point cloud rather than the measurement grid map. A novel dual-structure map building paradigm, composed of a voxel subspace division for particle storage and resampling and a dynamic pyramid-like subspace division for occlusion-aware particle update, is proposed to model the local environment with particles that have continuous states. Under the Gaussian noise assumption, we demonstrate that this updating paradigm is effective and computationally efficient. To reduce the noise in simultaneously modeling static and dynamic obstacles, the importance of newborn particles is addressed by using non-Gaussian initial velocity estimation and a mixture model that adaptively allocates the number of static and dynamic particles. With a complete process of prediction, update, birth, and resampling of particles in the continuous space, the occupancy status at an arbitrary point in the map can be estimated using onboard CPU devices.

In the experimental tests, we first evaluated the dynamic obstacle velocity estimation precision of the map. Then the ablation study was conducted to identify the mapping parameters. Subsequently, comparison tests were carried out, involving a state-of-the-art particle-based dynamic occupancy map [5] and a widely used static occupancy map [7]. Results show that our map has the best occupancy status estimation performance in dynamic environments and competitive performance with [7] in static environments. Furthermore, we verified the DSP map in obstacle avoidance tasks of a mini quadrotor in different environments. To the best of the authors' knowledge, this is the first continuous particle-based occupancy map and the first dynamic occupancy map that can be applied to small-scale robotic systems like quadrotors.

The main contributions of this work include:

1) A novel dual-structure particle-based map building paradigm that enables continuous mapping of the occupancy status in dynamic environments.
2) The leverage of initial velocity estimation and an efficient mixture model to reduce noise in modeling static

Gang Chen, Wei Dong, Peng Peng, and Xiangyang Zhu are with the State Key Laboratory of Mechanical System and Vibration, School of Mechanical Engineering, Shanghai Jiaotong University, 200240, Shanghai, China. E-mails: {chg947089399, dr.dongwei, yc_pengpeng, mexyzhu}@sjtu.edu.cn.

Javier Alonso-Mora is with the Autonomous Multi-Robots Lab, Department of Cognitive Robotics, Delft University of Technology, 2628 CD, Delft, Netherlands. E-mail: j.alonsomora@tudelft.nl.

This work was supported in part by the National Natural Science Foundation of China Grant 51975348 and in part by Shanghai Rising-Star Program under Grant 22QA1404400.

Corresponding authors: Wei Dong and Xiangyang Zhu.

and dynamic obstacles simultaneously.
3) The complete procedures of building a DSP map that can be applied to onboard computing devices of small-scale robotic systems.
4) The released code at https://github.com/g-ch/DSP-map, including an example application in ROS.

The remaining content is organized as follows: Section II describes the related work. Section III presents the background knowledge of our map. Section IV explains the formulations of the world model and gives an overview of mapping procedures. Section V expresses the mapping procedures with the dual structure. In Section VI, more components for mapping are discussed. Section VII presents some implementation details. The experimental results and the conclusion are described in Section VIII and Section IX, respectively.

II. Related Work

A. Discrete Map and Continuous Map

Environment representation is fundamental to obstacle avoidance of robotics systems. One of the most popular representation approaches is occupancy mapping, which originated from [8] and is capable of modeling cluttered environments. Grid map (2D or 3D) is a kind of computationally efficient form to realize occupancy mapping. The environment is usually divided into discrete grids, and the occupancy status of each grid is updated with the ray casting algorithm [7], [9]–[11]. The size of the grids, however, is difficult to determine. Large grids lead to a low resolution that is unfavorable for motion planning. Small grids increase the computation requirements and cause gaps and inconsistencies when the input point clouds are sparse or noisy [6]. To avoid the grid size problem and allow arbitrary resolutions, the paradigm of building the map with continuous occupancy probability kernels rather than grids is proposed [6], [12], [13]. Free space and occupied points or segments are first generated with the input point clouds and then used to update the parameters in the kernel functions. The occupancy status at an arbitrary position can then be estimated with nearby kernels.

B. Occupancy Maps in Dynamic Environments

The maps mentioned above [8]–[13] are built under the assumption that the environment is static. As the robotic systems were deployed in dynamic environments, improvements have to be made to instantly represent the occupancy position of dynamic obstacles, such as pedestrians and other robots, and, even further, to predict the future positions of dynamic obstacles. An intuitive approach is to leverage independent detection and tracking of moving objects (DATMO) [14]–[16] to model the dynamic obstacles and utilize static occupancy maps still to represent the other objects. A prerequisite of DATMO is that the detection and shape models of the dynamic obstacles are well-trained [3], which conflicts with the unknown environment characters in many tasks. In addition, difficulties in data association [3] and the trail noise caused by obstacles movements in the static map [15], [17] are intractable. Therefore, improving the map itself directly by considering the dynamic obstacle assumption is required, and the dynamic occupancy map [18], [19] emerges accordingly.

Early dynamic occupancy maps treat the dynamic obstacles, such as pedestrians and robots, as spurious data in the map, and detect and remove the data to build a robust static map [18]–[21]. Starting from the latest decade [1], research works considering modeling the dynamics, mostly velocities, of the obstacles in the map have been carried out to improve the obstacle avoidance performance in dynamic environments. Various methods have been proposed in these works. Some apply the dynamic obstacle assumption to the existing structures of static occupancy maps. For example, [22] adopts optical-flow-based motion maps to estimate the velocity of grids and improves the Gaussian process occupancy map [12] to adapt to dynamic environments. [23] further improves [22] by learning dynamic areas with stochastic variational inference. In [24], point clouds from lidar are clustered and filtered to estimate the velocities of dynamic obstacles. The estimation is applied to generate non-stationary kernels in the Hilbert space to build the dynamic Hilbert map. With the popularity of deep learning methods, some recent works adopt neural networks to predict the velocity of each grid in a grid map [25], [26] or future occupancy status [27], [28].

C. Particle-based Dynamic Occupancy Maps

The particle-based map originates from the autonomous driving area [1], [29]. In a particle-based map, an obstacle is regarded as a set of point objects and the particles with velocities are used to model the point objects. Compared to the dynamic occupancy maps in II-B, the particle-based map is originally proposed for dynamic environments and has a stronger potential to improve the mapping performance in complex and highly dynamic environments. Nuss et al. [2] improves [1], [29] by introducing the RFS theory and deriving map-building procedures with the PHD filter and the Bernoulli filter. The improved map can be built in real-time in 2D space with GPU devices. Later, [5] generalizes [2] to 3D space.

In a cluttered environment with dynamic and static obstacles, multiple point objects, dynamic or static, need to be modeled, and denoising is of great importance. Two approaches are usually adopted to reduce the noise. The first approach is to use a mixture model [3], [5], [30], [31], which includes a separate static model and a dynamic model, to update the states of static and dynamic point objects independently. The mixture model works as dual PHD filters [30] or the grid-level inference [3], [5], [31]. Another approach is to apply additional information to reduce the noise in the updating procedure. For instance, [4] adds an extra semantic grid channel in the input to generate particles with semantic labels and update with the semantic association.

The above particle-based maps are still grid maps. Measurement grids generated by the ray casting method are adopted as the input, and the environment is described with discretized 2D or 3D girds. This discretized expression suffers from the grid size problem mentioned in Section I. The grid size also limits the state estimation resolution of the obstacles. Therefore, a continuous particle-based occupancy map is required. In

addition, since numerous particles are used, state-of-the-art particle-based maps usually rely on Desktop GPU devices for computation [2], [5]. To deploy the particle-based map on small-scale robotic systems, improving computational efficiency is necessary.

TABLE I
NOTATION IN THIS PAPER

Symbol	Meaning
X, X_k	RFS, RFS composed of point objects at time k.
$\mathrm{X}_k^{(\mathbb{V}_i)}, \mathrm{X}_k^{(\mathbb{P}_i)}$	RFS composed of point objects in a voxel subspace and a pyramid subspace, respectively, with index i at time k.
Z_k	RFS composed of measurement points at time k.
$\mathrm{S}_{k\|k-1}$	RFS composed of survived objects from $k-1$ to k.
$\mathrm{B}_{k\|k-1}$	RFS composed of newborn objects from $k-1$ to k.
$\mathrm{O}_k, \mathrm{C}_k$	RFS composed of the detected objects and clutter at k.
$\boldsymbol{x}^{(i)}$	State vector of an element or an object with index i.
$\boldsymbol{z}^{(i)}$	State vector of a measurement point with index i.
(p_x, p_y, p_z)	Point object coordinate in Cartesian coordinate system.
$(\gamma_k, \alpha_k, \beta_k)$	Point object coordinate in sphere coordinate system.
$\mathbb{M}, \mathbb{M}^f$	The map space. Visible space in the map space.
$\mathbb{V}_i, \mathbb{P}_i$	Voxel subspace and pyramid subspace with index i.
$\mathbb{A}^{\boldsymbol{x}_k}$	Activation space of a point object $\boldsymbol{x}_k$.
$\mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}$	Activation space of a particle $\tilde{\boldsymbol{x}}_k^{(i)}$.
$D_{\mathrm{X}}(\boldsymbol{x})$	PHD of RFS X at state $\boldsymbol{x}$.
$\tilde{\boldsymbol{x}}_k^{(i)}, \tilde{\boldsymbol{x}}_{b,k}^{(i)}$	The state vector of a particle and a newborn particle.
$\tilde{\boldsymbol{x}}_{s,k\|k-1}^{(i)}$	The state vector of a particle survived from $k-1$ to k.
$w_k^{(i)}$	The weight of a particle with index i at time k.
P_d, P_s	Detection and survival probability of an object.
N_k, M_k	Number of point objects and measurement points at k.
N_v, N_p	Number of voxel subspaces and pyramid subspaces.
N_f	Number of pyramid subspaces in FOV.
L_k	Number of particles at time k.
L_b	Number of newborn particles from a measurement point.
L_{max}	Allowed max particle number in $\mathbb{M}$ after resampling.
$L_{max}^{\mathbb{V}}$	Allowed max particle number in $\mathbb{V}_i$ after resampling.
λ_1, λ_2	Coefficients in the mixture motion model.
$\gamma_{k\|k-1}, \kappa_k$	Intensity of the newborn objects and clusters.
(l_x, l_y, l_z)	Size of the map space.
l	Side length or resolution of a voxel subspace.
n	The number of adjacent pyramids on each side in $\mathbb{A}^{\boldsymbol{x}_k}$.
Res	Resolution of the voxel filter for point cloud pre-process.
r_{min}	The radius of the robot sphere model.
(θ_h, θ_v)	Horizontal and vertical angle of the FOV.
θ	Angle of a pyramid subspace.
$\boldsymbol{Q}$	Gaussian noise covariance matrix in prediction step.
$W_d^{(\mathbb{V})}$	Weight sum of dynamic particles in a voxel subspace.
$W_s^{(\mathbb{V})}$	Weight sum of static particles in a voxel subspace.
$W_{d,s}^{(\mathbb{V})}$	Weight sum of all the particles in a voxel subspace.
$V(\cdot)$	Function to calculate the absolute velocity value.
$m(\cdot), pr(\cdot)$	Mass function and probability function in DST.
$bel(\cdot), pl(\cdot)$	Belief function and plausibility function in DST.
$\pi_{k\|k-1}(\cdot)$	State transition density function of a single object.
$g_k(\cdot)$	Measurement likelihood function of a single object.
$f_Q(\cdot)$	State transition function of a single point object.
$f_R(\cdot)$	Measurement function of a single point object.
$R(\cdot)$	Function that defines the measurement noise matrix.
$\rho(\cdot)$	Function that defines the standard deviation on each axis.

III. PRELIMINARIES

This section introduces the main concepts of RFS, PHD, PHD filter, and SMC-PHD filter. The relationship between the concepts is: PHD is a first moment of an RFS; PHD filter realizes multi-object tracking by propagating PHD; SMC-PHD filter is a particle-based implementation of the PHD filter and is used to fulfill prediction and update in our DSP map. The notations used in this section and the rest sections are shown in Table I.

A. Random Finite Set

An RFS is a finite set-valued random variable [2]. The number and the states of the elements in an RFS are random but finite. Let X denote an RFS and $\boldsymbol{x}^{(i)} \in \mathbb{M}$ denote the state vector of an element in X. $\mathbb{M}$ is $\boldsymbol{x}^{(i)}$'s state space, e.g., map space. Then X is expressed as:

$$\mathrm{X} = \left\{ \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)} \right\} \tag{1}$$

where $N \in \mathbb{N}$ is a random variable representing elements number in X and is called the cardinality of X. Specially, when $N = 0$, X is $\emptyset$. A common usage of the RFS is in the multi-object tracking area, where $\boldsymbol{x}^{(i)}$ is usually the state of an object and X is the set composed of the states of all objects. N varies as objects appear and disappear in the tracking range.

B. PHD

PHD [32], [33] is a first moment of an RFS and is raised to describe the multi-object density. The PHD of X at a state $\boldsymbol{x}$ is defined as:

$$D_{\mathrm{X}}(\boldsymbol{x}) = \mathbf{E}\left[\sum_{\boldsymbol{x}^{(i)} \in \mathrm{X}} \delta(\boldsymbol{x} - \boldsymbol{x}^{(i)}) \right] \tag{2}$$

where $\mathbf{E}[\cdot]$ is the expectation and $\delta(\cdot)$ is the Dirac function[1].

Two important properties of PHD are used in this work. The first property is that the integral of PHD is the expectation of the cardinality of X, which can be expressed as

$$\int D_{\mathrm{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \mathbf{E}[|\mathrm{X}|] \tag{3}$$

where $|\mathrm{X}|$ represents the cardinality of X.

Another property is that if $\mathrm{X}^{(1)}, \mathrm{X}^{(2)}, ..., \mathrm{X}^{(N')}$ are independent RFSs, and $\mathrm{X}^{(1)} \cup \mathrm{X}^{(2)} \cup ... \cup \mathrm{X}^{(N')} = \mathrm{X}$, then

$$D_{\mathrm{X}}(\boldsymbol{x}) = D_{\mathrm{X}^{(1)}}(\boldsymbol{x}) + D_{\mathrm{X}^{(2)}}(\boldsymbol{x}) + \cdots + D_{\mathrm{X}^{(N')}}(\boldsymbol{x}) \tag{4}$$

C. PHD Filter

The PHD filter [32] is an efficient filter that propagates the PHD in the prediction and the update step, and can be used to handle multiple object tracking problems. Let X_{k-1} and X_k denote the RFS composed of object states at time step $k-1$ and k, respectively. Suppose Z_k is the RFS composed of measurements, i.e., point cloud, to the objects at time k. In the prediction step of a typical PHD filter, the prior object states RFS $\mathrm{X}_{k|k-1}$ can be treated as the union of two independent subsets, which is $\mathrm{X}_{k|k-1} = \mathrm{S}_{k|k-1} \cup \mathrm{B}_{k|k-1}$, where $\mathrm{S}_{k|k-1}$ represents the persistent objects from the X_{k-1}, and $\mathrm{B}_{k|k-1}$ is the newly born objects. Note $\mathrm{S}_{k|k-1}$ and $\mathrm{B}_{k|k-1}$ are distinguished by birth time. They are both in map space $\mathbb{M}$ but don't share any element. $\mathrm{S}_{k|k-1}$ is usually modeled with

[1]Dirac function: $\delta(\boldsymbol{x}) = 0, \text{ if } \boldsymbol{x} \neq \mathbf{0}; \int \delta(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = 1.$

Multi-Bernoulli mixture (MBM). From X_{k-1} to $\mathrm{S}_{k|k-1}$, the objects have a probability of P_s to survive. Meanwhile, $\mathrm{B}_{k|k-1}$ is modeled as a Poisson point process (PPP) [32] with intensity $\gamma_{k|k-1}(\boldsymbol{x}_k)$. Similarly, in the update step, Z_k is expressed as $\mathrm{Z}_k = \mathrm{O}_k \cup \mathrm{C}_k$, where O_k is the detected objects set and C_k is the set of clutter. From X_k to Z_k, the objects have a probability of P_d to be detected. The clutter C_k are modeled as a PPP with intensity $\kappa_k(\boldsymbol{z}_k)$.

Let $D_{\mathrm{S}_{k|k-1}}(\boldsymbol{x}_k)$ and $D_{\mathrm{B}_{k|k-1}}(\boldsymbol{x}_k)$ denote the PHD at $\boldsymbol{x}_k$ of RFS $\mathrm{S}_{k|k-1}$ and $\mathrm{B}_{k|k-1}$, respectively. Considering the property (4) and the MBM and PPP models, the general PHD filter [33] is described as:

$$\begin{aligned} D_{\mathrm{X}_{k|k-1}}(\boldsymbol{x}_k) &= D_{\mathrm{S}_{k|k-1}}(\boldsymbol{x}_k) + D_{\mathrm{B}_{k|k-1}}(\boldsymbol{x}_k) \\ &= P_s H_k(\boldsymbol{x}_k, \boldsymbol{x}_{k-1}) + \gamma_{k|k-1}(\boldsymbol{x}_k) \end{aligned} \tag{5}$$

$$H_k(\boldsymbol{x}_k, \boldsymbol{x}_{k-1}) = \int \pi_{k|k-1}(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) D_{\mathrm{X}_{k-1}}(\boldsymbol{x}_{k-1}) \mathrm{d}\boldsymbol{x}_{k-1} \tag{6}$$

$$D_{\mathrm{X}_k}(\boldsymbol{x}_k) = \left[1 - P_d + P_d \sum_{\boldsymbol{z}_k \in \mathrm{Z}_k} G_k(\boldsymbol{z}_k, \boldsymbol{x}_k)\right] D_{\mathrm{X}_{k|k-1}}(\boldsymbol{x}_k) \tag{7}$$

$$G_k(\boldsymbol{z}_k, \boldsymbol{x}_k) = \frac{g_k(\boldsymbol{z}_k|\boldsymbol{x}_k)}{\kappa_k(\boldsymbol{z}_k) + P_d \int g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) D_{\mathrm{X}_{k|k-1}}(\boldsymbol{x}_k) \mathrm{d}\boldsymbol{x}_k} \tag{8}$$

where Equation (5) and (6) show the prediction step, and Equation (7) and (8) present the update step. $\pi_{k|k-1}(\cdot)$ is the state transition density of a single object and $g_k(\cdot)$ is the single object measurement likelihood.

D. SMC-PHD Filter

Sequential Monte Carlo PHD (SMC-PHD) filter [34] [35] uses particles to represent PHD and is an efficient implementation of the PHD filter. Each particle has a weight and a state vector with the same dimension as an object's state. With the particles, the posterior PHD of X at time $k-1$ is approximated by

$$D_{\mathrm{X}_{k-1}}(\boldsymbol{x}_{k-1}) \approx \sum_{i=1}^{L_{k-1}} w_{k-1}^{(i)} \delta(\boldsymbol{x}_{k-1} - \tilde{\boldsymbol{x}}_{k-1}^{(i)}) \tag{9}$$

where L_{k-1} is the number of particles at time step $k-1$, $w_{k-1}^{(i)}$ is the weight of particle with index (i), and $\tilde{\boldsymbol{x}}_{k-1}^{(i)}$ denotes the state vector of particle (i). We distinguish the state of an object and the state of a particle with the tilde notation.

In the prediction step, with Equations (9), (5) and (6), the prior PHD of the RFS $\mathrm{X}_{k|k-1}$ at k is derived as:

$$\begin{aligned} D_{\mathrm{X}_{k|k-1}}(\boldsymbol{x}_k) &= D_{\mathrm{S}_{k|k-1}}(\boldsymbol{x}_k) + D_{\mathrm{B}_{k|k-1}}(\boldsymbol{x}_k) \\ &= \sum_{i=1}^{L_{k-1}} P_s w_{k-1}^{(i)} \pi_{k|k-1}(\boldsymbol{x}_k|\tilde{\boldsymbol{x}}_{k-1}^{(i)}) + \gamma_{k|k-1}(\boldsymbol{x}_k) \end{aligned} \tag{10}$$

Let $w_{s,k|k-1}^{(i)} = P_s w_{k-1}^{(i)}$. By sampling $\pi_{k|k-1}(\boldsymbol{x}_k|\tilde{\boldsymbol{x}}_{k-1}^{(i)})$ and $\gamma_{k|k-1}(\boldsymbol{x}_k)$ with particles, the above equation can be further derived as:

$$\begin{aligned} &D_{\mathrm{X}_{k|k-1}}(\boldsymbol{x}_k) \\ &= \sum_{i=1}^{L_{k-1}} w_{s,k|k-1}^{(i)} \delta(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_{s,k|k-1}^{(i)}) + \sum_{j=1}^{L_{b,k}} w_{b,k}^{(i)} \delta(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_{b,k}^{(j)}) \\ &\equiv \sum_{i=1}^{L_k} w_{k|k-1}^{(i)} \delta(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k^{(i)}) \end{aligned} \tag{11}$$

where $\tilde{\boldsymbol{x}}_{s,k|k-1}^{(i)}$ represents the particle state sampled from $\pi_{k|k-1}(\boldsymbol{x}_k|\tilde{\boldsymbol{x}}_{k-1}^{(i)})$ and $\tilde{\boldsymbol{x}}_{b,k}^{(j)}$ represents the particle state sampled from $\gamma_{k|k-1}(\boldsymbol{x}_k)$. $L_{b,k}$ and $w_{b,k}^{(i)}$ are the number and the weight of newborn particles at time k, respectively. The total number of particles after prediction is $L_k = L_{k-1} + L_{b,k}$.

In the update step, substitute $D_{\mathrm{X}_{k-1}}(\boldsymbol{x}_{k-1})$ in Equations (7) and (8) with the particle representation in the last row of (11). The posterior PHD at k is reformed into the summation of particles, which is

$$D_{\mathrm{X}_k}(\boldsymbol{x}_k) \approx \sum_{i=1}^{L_k} w_k^{(i)} \delta(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k^{(i)}) \tag{12}$$

where the particle state $\tilde{\boldsymbol{x}}_k^{(i)}$ remains the same as in the prediction step and the weight $w_k^{(i)}$ is given by:

$$w_k^{(i)} = \left[1 - P_d + \sum_{\boldsymbol{z}_k \in \mathrm{Z}_k} \frac{P_d g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(i)})}{\kappa_k(\boldsymbol{z}_k) + \mathrm{C}_k(\boldsymbol{z}_k)}\right] w_{k|k-1}^{(i)} \tag{13}$$

$$\mathrm{C}_k(\boldsymbol{z}_k) = \sum_{j=1}^{L_k} P_d w_{k|k-1}^{(j)} g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(j)}) \tag{14}$$

The SMC-PHD filter estimates the PHD of X by iterative prediction with Equation (9) to (11) and update with Equation (12) to (14). Details can be found in [34] [35].

IV. World Model and System Overview

A. World Model

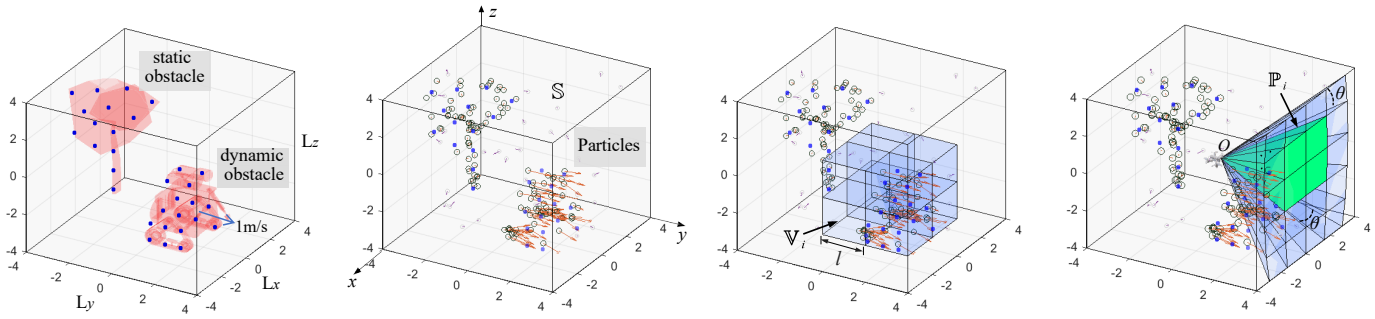
Our DSP map is an egocentric map built on multi-object tracking at the point object level in a continuous neighborhood space. Let $\mathbb{M}$ denote the neighborhood map space of the robot. $\mathbb{M}$ is a real space that has a cuboid boundary with size (l_x, l_y, l_z). The size can be set according to the range of the utilized sensors or the requirements from the motion planner. At the center of the cuboid is the robot. We consider the obstacles in $\mathbb{M}$ as point objects, similar to [2]. Fig. 1(a) reveals the relation between obstacles and point objects. One obstacle can correspond to multiple point objects. The point objects are used to estimate the occupancy status at an arbitrary position in the map. Since the occupancy status rather than the state of each obstacle is more important in an occupancy map, the mapping from point objects to obstacles is omitted and the assumption that all the point objects move independently is made. The same assumption is used in the existing works on particle-based maps [1] [2] [5].

For the reason that the obstacles are unknown, the number of the point objects in $\mathbb{M}$ and their states are random but finite. Therefore, these point objects can be modeled as an RFS. At a discrete time k, the RFS composed of the point object states is represented as

$$\mathrm{X}_k = \left\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N_k)}\right\} \tag{15}$$

where N_k is the number of point objects at time k, and $\boldsymbol{x}$ with index from 1 to N_k is the state vector of a point object. The state vector is given by the 3D position and velocity, namely

$$\boldsymbol{x} = [p_x, p_y, p_z, v_x, v_y, v_z]^T \tag{16}$$



(a) Point objects (b) Particles in the local map (c) Voxel subspaces (d) Pyramid subspaces

Fig. 1. Illustration of the world model. Subfigure (a) shows a cubic local environment with a static obstacle and a dynamic obstacle. The small blue points are the point objects that represent the two obstacles. The layout of the point objects depends on the measurement points. The blue points show one kind of layout. Subfigure (b) presents the particles (small hollow circles with arrows indicating the velocities) used to model the point objects. Subfigure (c) and Subfigure (d) are two different space division structures. The whole local environment is divided into subspaces, but only a part of the subspaces are plotted to have a clear view of their shapes. In Subfigure (d), the green pyramids indicate the current FOV.

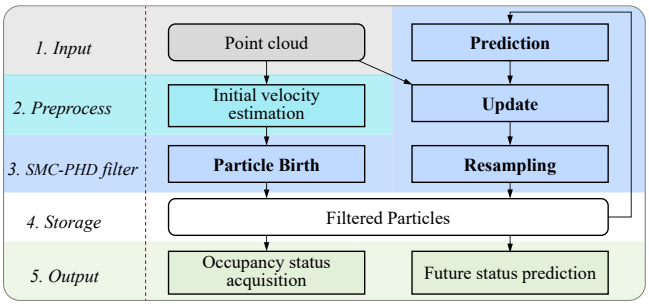


Fig. 2. System overview of our DSP map.

where the subscripts $\{x, y, z\}$ are used to represent the axes in Cartesian coordinate. The core of building the DSP map is to use the SMC-PHD filter to track the point objects in 3D continuous space and estimate X_k's PHD, which is then used to estimate the occupancy status of the map.

To realize effective and efficient SMC-PHD filtering in the continuous space, we divide $\mathbb{M}$ into two types of subspaces, i.e., the cubic voxel subspaces and the pyramid-like subspaces, by the position dimensions. The voxel subspaces are used for data storage and particle resampling. The pyramid-like subspaces are applied to handle limited sensor FOV and inevitable occlusions in the continuous space, and realize efficient particle update. Details are presented in Section V. The following describes how to acquire the subspaces and defines the sub-RFSs divided accordingly with the subspaces.

The voxel subspaces are divided in the cartesian coordinate (Fig. 1(c)). The voxels can fill up $\mathbb{M}$ but have no overlaps with each other. Assume the resolution of the voxel is l. Then the number of the voxels is $N_v = \frac{l_x \cdot l_y \cdot l_z}{l^3}$. Let $\mathbb{V}_i$ denote the i^{th} voxel subspace. Then X_k can be described as the union of these sub-RFSs, which is

$$\mathrm{X}_k = \mathrm{X}_k^{(\mathbb{V}_1)} \cup \mathrm{X}_k^{(\mathbb{V}_2)} \cup \cdots \cup \mathrm{X}_k^{(\mathbb{V}_{N_v})} \tag{17}$$

Since the voxels have no overlaps, any two sub-RFSs don't share a point object, and thus, the sub-RFSs are independent. In the SMC-PHD filter, the voxel subspaces are used to resample the particles in $\mathbb{M}$ in a uniform manner, which is described in Section V-D. In addition, these voxel subspaces are used to index and store the particles for efficiency purposes, as described in Section VII.

For the reason that the field of view (FOV) of a sensor is usually limited, and the occlusion prevents observations of the area behind obstacles, only a part of $\mathbb{M}$ is visible. Let $\mathbb{M}^f \subset \mathbb{M}$ denote the visible space. $\mathbb{M}^f$ must be distinguished from the occluded space to realize map updating. However, the voxel subspaces have a limited resolution and cannot continuously express $\mathbb{M}^f$. Thus, another division structure is still required.

Inspired by the perspective projection model for sensors, we also divide $\mathbb{M}$ into pyramid-like subspaces in the spherical coordinate (Fig. 1(d)). These subspaces are divided dynamically and uniformly in the sensor frame when the robot pose is given. (Details can be found in Section VII and Algorithm 2 in the Appendix.) The real shape of a pyramid-like subspace is composed of four near-triangular faces and one face on the map boundary face. For simplification, we loosely name the subspace as pyramid subspace in the following content.

In the spherical coordinate, the azimuth angle range is $[0, 2\pi]$ and the zenith angle range is $[0, \pi]$. Suppose the angle interval of the pyramid division, namely the pyramid angle, is $\theta > 0$. The number of these subspaces is $N_p = \frac{2\pi \cdot \pi}{\theta^2}$. To make N_p an integer, θ satisfies $I\theta = \pi$, where $I \in \mathbb{N}+$. Denote by $\mathbb{P}_i$ the i^{th} pyramid subspace, and by $\mathrm{X}_k^{(\mathbb{P}_i)}$ the RFS composed of point objects in $\mathbb{P}_i$. X_k satisfies

$$\mathrm{X}_k = \mathrm{X}_k^{(\mathbb{P}_1)} \cup \mathrm{X}_k^{(\mathbb{P}_2)} \cup \cdots \cup \mathrm{X}_k^{(\mathbb{P}_{N_p})} \tag{18}$$

The measurement of the point objects is the point cloud from sensors, such as stereo cameras or Lidars. The points in the point cloud at time k form a measurement RFS Z_k. In analogy to the point objects, Z_k is written as

$$\mathrm{Z}_k = \left\{ \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, ..., \boldsymbol{z}^{(M_k)} \right\} \tag{19}$$

where M_k represents the number of the measurement points, and each measurement point $\boldsymbol{z}$ consists of the 3D position, which is

$$\boldsymbol{z} = \{z_x, z_y, z_z\} \tag{20}$$

With the measurement points and the pyramid-like subspaces, we can determine the visible space $\mathbb{M}^f$ and occluded space. As is shown in green in Fig. 3. (a) and (b), $\mathbb{M}^f$ is the union of the free space and obstacle surface in each pyramid subspace in the FOV. Denote the visible space of pyramid subspace $\mathbb{P}_i$ by $\mathbb{P}_i^f$. When the pyramid angle θ of $\mathbb{P}_i$ equals the angular resolution of the sensor, there is either one or no measurement point in $\mathbb{P}_i$. If there is one point $\boldsymbol{z}$, the subspace behind the measurement point is occluded (painted in gray in Fig. 3), while the rest space is the visible pyramid subspace $\mathbb{P}_i^f$. $\mathbb{P}_i^f \subset \mathbb{P}_i$ and the length of $\mathbb{P}_i^f$ is $|\boldsymbol{z}|$. If there is no measurement point, $\mathbb{P}_i^f = \mathbb{P}_i$. Suppose the FOV is $\theta_h \times \theta_v$. The number of $\mathbb{P}_i^f$ is $N_f = \frac{\theta_h \theta_v}{\theta^2}$. Since the FOV usually cannot cover the whole neighborhood space, $N_f < N_p$. Then $\mathbb{M}^f = \mathbb{P}_1^f \cup \cdots \cup \mathbb{P}_{N_f}^f$, and Z_k can be divided into subsets with these visible pyramid subspaces, which is

$$\mathrm{Z}_k = \mathrm{Z}_k^{(\mathbb{P}_1^f)} \cup \mathrm{Z}_k^{(\mathbb{P}_2^f)} \cup \cdots \cup \mathrm{Z}_k^{(\mathbb{P}_{N_f}^f)} \tag{21}$$

With the measurement Z_k, the PHD of X_k is updated by using the SMC-PHD filter. The hollow circles with velocity arrows in Subfigures (b), (c), and (d) in Fig. 1 show the particles used in the SMC-PHD filter. The basic element in our map is the particle.

B. System Overview

An overview of the procedures to build our DSP map can be found in Fig. 2. The core procedure is filtering the PHD of X_k with the SMC-PHD filter. In the SMC-PHD filter, the prediction step and update step iteratively update the PHD. The particle birth step generates new particles and is then used in the prediction step. A resampling step is added after the update step to prevent degeneration and control the maximum number of particles. The pyramid subspace division is used in the update step to distinguish visible space and improve computational efficiency. The voxel subspace division is used in the resampling step to realize efficient and uniform particle resampling. Details about the SMC-PHD filtering procedure in our map can be found in Section V. The filtered result is particles with position and velocity states. Then the map output can be calculated with the particles into two forms designed for motion planning. The first form is the current occupancy status, and the second is the prediction of future occupancy status. An initial velocity estimation procedure is introduced to reduce the noise in mapping. Details about the output and initial velocity estimation are presented in Section VI. The particles used in all the procedures are stored in the voxel subspaces.

V. Mapping with Dual Structure

This section presents the core procedures to build our map with the dual-structure space divisions, including prediction, update, particle birth, and resampling steps. The prediction and the particle birth are conducted in space $\mathbb{M}$. In the update step, the pyramid subspaces $\mathbb{P}_i$ are utilized to update the point objects' PHD efficiently. The voxel subspaces $\mathbb{V}_i$ are adopted in the resampling step.

A. Prediction

The prediction step predict the prior PHD of $\mathrm{X}_{k|k-1}$ and the general form has been described in Section III-D. In our map, the motion model of a single point object is defined by the constant velocity (CV) model, then a point object $\boldsymbol{x}_k$ that survived from $k-1$ is predicted by:

$$\boldsymbol{x}_k = f_Q\left(\boldsymbol{x}_{k-1}\right) + \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \Delta t \boldsymbol{I}_{3\times3} \\ 0_{3\times3} & \boldsymbol{I}_{3\times3} \end{bmatrix} \boldsymbol{x}_{k-1} + \boldsymbol{\xi} \tag{22}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{\xi}$ is the noise. The noise is supposed to obey a Gaussian distribution with a covariance $\boldsymbol{Q}$, which is $\boldsymbol{\xi} \sim \mathcal{N}(0, \boldsymbol{Q})$.

Then the state transition density in Equation (6) turns to a Gaussian probability density:

$$\pi_{k|k-1}(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) = \mathcal{N}\left(\boldsymbol{x}_k; f_Q\left(\boldsymbol{x}_{k-1}\right), \boldsymbol{Q}\right) \tag{23}$$

Thus, from Equation (10) to (11), $\pi_{k|k-1}(\boldsymbol{x}_k|\tilde{\boldsymbol{x}}_{k-1}^{(i)})$ can be sampled by particles using the Gaussian probability density, and $\tilde{\boldsymbol{x}}_{s,k|k-1}^{(i)}$ in Equation (11) is given by

$$\tilde{\boldsymbol{x}}_{s,k|k-1}^{(i)} = f_Q\left(\tilde{\boldsymbol{x}}_{k-1}^{(i)}\right) + \boldsymbol{u} \tag{24}$$

where $\boldsymbol{u}$ is a noise sampled from $\mathcal{N}(0, \boldsymbol{Q})$.

The weight and state of newborn particles in Equation (11) are described later in Section V-C.

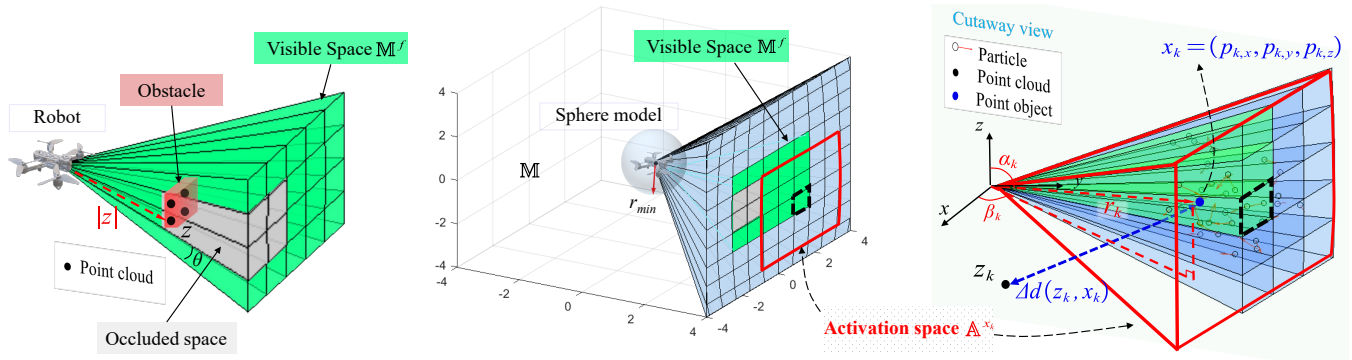
B. Update

The update step utilizes the measurement Z_k to get the posterior PHD of X_k. Two major points are addressed in the update step. The first point is to tackle the limited FOV and the occlusion. Since the FOV of a sensor is usually limited, and the occlusion prevents observations to the area behind obstacles, Z_k can only be in the visible space $\mathbb{M}^f$ defined in Section IV. The objects that do not belong to $\mathbb{M}^f$ are in an unknown area and should not be updated; otherwise, their existence probability will be falsely reduced. Thus, the notations in Equation (12) should contain a superscript f, such as $D_{\mathrm{X}_k}^f(\boldsymbol{x}_k)$, $w_k^{f,(i)}$ and $\tilde{\boldsymbol{x}}_k^{f,(i)}$. Let superscript $\bar{f}$ represent the definitions in $\mathbb{M} \setminus \mathbb{M}^f$. Then by using the property in (4), the PHD of the all objects in $\mathbb{M}$ should be estimated as:

$$D_{\mathrm{X}_k}(\boldsymbol{x}_k) = D_{\mathrm{X}_k}^f(\boldsymbol{x}_k) + D_{\mathrm{X}_k}^{\bar{f}}(\boldsymbol{x}_k) \tag{25}$$

where $D_{\mathrm{X}_k}^{\bar{f}}(\boldsymbol{x}_k) = D_{\mathrm{X}_{k|k-1}}^{\bar{f}}(\boldsymbol{x}_k)$ because the objects in $\mathbb{M}\setminus\mathbb{M}^f$ are not updated. $D_{\mathrm{X}_k}^f(\boldsymbol{x}_k)$ can be updated with Equation (12), (13) and (14) by considering the point objects and particles in $\mathbb{M}^f$ only. For notation simplification, the superscript f is omitted in what following. We still use Equation (12), (13) and (14) to represent the general update form but now $\tilde{\boldsymbol{x}}_k^{(i)} \in \mathbb{M}^f$ and L_k only counts the particles in $\mathbb{M}^f$.

The second point is to reduce the computational complexity. It should be noted that $\mathrm{C}_k(\boldsymbol{z}_k)$ in Equation (13) and (14) is controlled by $\boldsymbol{z}_k$, and thus, for every $w_k^{(i)}$, $\mathrm{C}_k(\boldsymbol{z}_k)$ can be shared for the same $\boldsymbol{z}_k$. Therefore, to calculate all the required $\mathrm{C}_k(\boldsymbol{z}_k)$, the multiplication operation and PDF calculation calculation in (14) should be performed $L_k \cdot M_k$ times, where M_k is the cardinality of Z_k. In addition, considering the summation



(a) FOV space composed of visible pyramid subspaces (b) Pyramid subspaces in map and FOV (c) Update in activation space

Fig. 3. Illustration of the pyramid subspaces in the FOV and the update step. Subplot (a) shows the visible space and the occluded space. Subplot (b) reveals the visible space and the map space in the same plot. The pyramid subspaces out of the current FOV are partially presented to have a clear view. For a point object x that lies in a pyramid subspace belonging to the visible space, such as the subspace outlined with black dashes, we define the activation space of x as the union of this pyramid subspace and its adjacent n pyramid subspaces. The adjacent n pyramid subspaces of a pyramid are within the range of n rows and n columns from the pyramid. $n = 2$ in this case. Subplot (c) shows a cutaway view of the activation space. Point cloud measurement $\boldsymbol{z}_k$ locates out of the activation space. The distance from the point object x to the map center is r_k.

operations in Equation (12) and (13), another $L_k \cdot M_k$ times of multiplication, division and PDF calculation operations should be performed. The algorithmic complexity is $O(L_k M_k)$. In an unknown environment, there could be many obstacles and over a million particles can be required to approximate the states of the point objects. Hence, $L_k \cdot M_k$ can be very large, and the efficiency of the map is not adequate. The following considers using the pyramid subspaces to reduce the complexity.

Considering the measurement noise of the commonly used point cloud sensors, such as depth camera and Lidar, the single object measurement likelihood $g_k(\cdot)$ can be assumed as a Gaussian distribution, which is

$$g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{z}_k; f_R(\boldsymbol{x}_k), R(\boldsymbol{x}_k)) \tag{26}$$

where $f_R(\boldsymbol{x}_k) = [\boldsymbol{I}_{3\times3}, 0_{3\times3}] \cdot \boldsymbol{x}_k$ since the measurement is only position. Unlike the prediction covariance $\boldsymbol{Q}$, the measurement covariance $R(\boldsymbol{x}_k)$ is usually not constant but related to the distance d_k of the obstacle in regular sensor models.

Firstly, we assume that the measurement error of a point object $\boldsymbol{x}_k \in \mathbb{M}^f$ is independent on each axis, and the standard deviation on each axis is equally $\rho(d_k)$, where $\rho(\cdot)$ is a function. The coordinate of $\boldsymbol{x}_k$ in the sphere coordinate system is (r_k, α_k, β_k) (Fig. 3 (c)), where $r_k = \sqrt{p_{k,x}^2 + p_{k,y}^2 + p_{k,z}^2}$, $\alpha_k = \arccos \frac{p_{k,z}}{r_k}$, and $\beta_k = \arccos \frac{p_{k,x}}{r \sin(\alpha_k)}$, if $p_{k,y} \geq 0$, and $\beta_k = \arccos \frac{p_{k,x}}{r \sin(\alpha_k)} + \pi$, if $p_{k,y} < 0$.

r_k and α_k are bounded. Suppose the robot is in a sphere model with radius $r_{min} > 0$. The space inside the sphere is considered free and not updated. Let $r_{max} = \frac{1}{2}\sqrt{l_x^2 + l_y^2 + l_z^2}$. Then $r_k \in [r_{min}, r_{max}]$. Considering the sensor has a limited FOV with vertical view angle $\theta_v < \pi$, we have $\alpha_k \in \left[\frac{\pi - \theta_v}{2}, \frac{\pi + \theta_v}{2}\right]$.

With the coordinates in the sphere coordinate system, the covariance turns to $R(\boldsymbol{x}_k) = \rho^2(r_k)\boldsymbol{I}_{3\times3}$ and $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k)$ can then be rewritten as

$$g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) = \prod_{i\in\{x,y,z\}} \mathcal{N}\left(z_{k,i}; p_{k,i}, \rho^2(r_k)\right) \tag{27}$$

where $z_{k,i}$ and $p_{k,i}$ are the single-axis position of a measurement and an object, respectively, at time k, which are described in (20) and (16).

In Equation (14), $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k)$ involves the transition density from $\boldsymbol{x}_k$ to every observed $\boldsymbol{z}_k$ and thus makes the update step time-consuming. Let ϵ denote a small constant. If $\boldsymbol{z}_k$ satisfies Condition $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) < \epsilon$, the approximation $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) = 0$, i.e. the strategy that the $\boldsymbol{z}_k$ is omitted in (14), is applied to improve the update efficiency. To find the $\boldsymbol{z}_k$ that satisfies the condition, we define an activation space $\mathbb{A}^{\boldsymbol{x}_k}$ for point object $\boldsymbol{x}_k$. The activation space indicates that $\boldsymbol{z}_k$ out of $\mathbb{A}^{\boldsymbol{x}_k}$ satisfies $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) \approx 0$. $\mathbb{A}^{\boldsymbol{x}_k}$ is the adjacent space of $\boldsymbol{x}_k$, and is composed of the union of the pyramid subspace where $\boldsymbol{x}_k$ is and the adjacent n pyramid subspaces. The adjacent n pyramid subspaces are within the range of n rows and n columns from the pyramid subspace. For example, Fig. 3. (b) and (c) show the activation space of $\boldsymbol{x}_k$ with $n = 2$. The number of pyramid subspaces in $\mathbb{A}^{\boldsymbol{x}_k}$ is $(2n+1)^2$.

Let $\Delta d(\boldsymbol{z}_k, \boldsymbol{x}_k)$ denote the Euclidean position distance between an measurement point $\boldsymbol{z}_k$ and the point object $\boldsymbol{x}_k$. Consider the 3d Gaussian probability density, $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k)$ can be further written as:

$$g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) = \frac{1}{(2\pi)^{\frac{3}{2}}\rho^3(r_k)}\, e^{-\frac{\Delta d(\boldsymbol{z}_k, \boldsymbol{x}_k)^2}{2\rho^2(r_k)}} \tag{28}$$

If $\boldsymbol{z}_k \notin \mathbb{A}^{\boldsymbol{x}_k}$, the absolute azimuth angle and the zenith angle difference between $\boldsymbol{x}_k$ and $\boldsymbol{z}_k$ is no less than $n\theta$. Let $\theta' = n\theta$. In Appendix A, we derive that the lower bound of $\Delta d(\boldsymbol{z}_k, \boldsymbol{x}_k)$ is $r_k \sin\theta' \sin\alpha_k$. Thus for $\forall \boldsymbol{z}_k \notin \mathbb{A}^{\boldsymbol{x}_k}$, the maximum density is

$$g_{k,max}(\boldsymbol{z}_k|\boldsymbol{x}_k) = \frac{1}{(2\pi)^{\frac{3}{2}}\rho^3(r_k)}\, e^{-\frac{(r_k \sin\theta' \sin\alpha_k)^2}{2\rho^2(r_k)}} \tag{29}$$

When $(r_k \sin\theta' \sin\alpha_k)^2$ increases, $g_{k,max}(\boldsymbol{z}_k|\boldsymbol{x}_k)$ decreases monotonically.

Let $g_{k,max}(\boldsymbol{z}_k|\boldsymbol{x}_k) = \epsilon$. Equation (29) can be reformed as

$$\theta' = \arcsin \sqrt{\frac{2\rho^2(r_k)}{r_k^2 sin^2 \alpha_k} \ln \left[\epsilon (2\pi)^{\frac{3}{2}} \rho^3(r_k)\right]^{-1}} \tag{30}$$

Since $r_k \in [r_{min},\ r_{max}]$ and $\alpha_k \in \left[\frac{\pi-\theta_v}{2}, \frac{\pi+\theta_v}{2}\right]$ are in close intervals, θ' must have a maximum value θ'_{max}. For example,

Case 1: when $\rho(r_k)$ equals a constant value σ, θ'_{max} is

$$\theta'_{max} = \arcsin \sqrt{\frac{2\sigma^2}{r_{min}^2 cos^2 \frac{\theta_v}{2}} \ln \left[\epsilon (2\pi)^{\frac{3}{2}} \sigma^3\right]^{-1}} \tag{31}$$

Case 2: when $\rho(r_k) = \sigma' r_k$, which means the measurement standard deviation grows linearly with r_k, then

$$\theta'_{max} = \arcsin \sqrt{\frac{2\sigma'^2}{cos^2 \frac{\theta_v}{2}} \ln \left[\epsilon (2\pi)^{\frac{3}{2}} \sigma'^3 r_{min}^3\right]^{-1}} \tag{32}$$

Therefore, given a threshold ϵ, θ'_{max} can be calculated and the parameter n for the activation space is $n = \lceil \frac{\theta'_{max}}{\theta} \rceil$. Then $\forall \boldsymbol{z}_k \notin \mathbb{A}^{\boldsymbol{x}_k}$, $g_k(\boldsymbol{z}_k|\boldsymbol{x}_k) \leq \epsilon \approx 0$. Note the formula in the square root symbol in Eq. (31) or (32) should be in the range $[0, 1]$, which generally holds given real-world sensor parameters and robot size. Special cases when θ_v is near π or $\rho(r_k)$ is very large can make the condition invalid. Then the strategy of increasing r_{min} or decreasing θ_v in the map can be adopted to make the condition valid. The strategy increases the sphere model size or decreases the pyramid number, and thus, sacrifices part of the space to be updated.

If the measurement variances on each axis are not identical, the upper envelope of the variances can be taken as $\rho(r_k)$, and the above inference still holds. If the measurement errors on each axis are not independent, Equations (27) to (32) cannot hold but the derived result can be used as an approximation to determine n. In the following context, we suppose the measurement errors on each axis are independent.

At the particle level, substitute $\boldsymbol{x}_k$ with $\tilde{\boldsymbol{x}}_k^{(i)}$, and then, $g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(i)}) \approx 0$ if $\boldsymbol{z}_k \notin \mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}$, where $\mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}$ is the activation space of particle $\tilde{\boldsymbol{x}}_k^{(i)}$ and $\mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}} = \mathbb{A}^{\boldsymbol{x}_k}$ if $\tilde{\boldsymbol{x}}_k^{(i)}$ is in the same pyramid subspace of $\boldsymbol{x}_k$. Let $L_k^{\mathbb{A}^{\boldsymbol{z}_k}}$ denote the number of particles whose activation space includes $\boldsymbol{z}_k$. Equations (13) and (14) can be expressed as:

$$w_k^{(i)} = \left[1 - P_d + \sum_{\boldsymbol{z}_k \in \mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}} \frac{P_d g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(i)})}{\kappa_k(\boldsymbol{z}_k) + \mathrm{C}_k(\boldsymbol{z}_k)}\right] w_{k|k-1}^{(i)} \tag{33}$$

$$\mathrm{C}_k(\boldsymbol{z}_k) = \sum_{j=1}^{L_k^{\mathbb{A}^{\boldsymbol{z}_k}}} P_d w_{k|k-1}^{(j)} g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(j)}) \tag{34}$$

$L_k^{\mathbb{A}^{\boldsymbol{z}_k}}$ is about $\frac{(2n+1)^2}{N_f}$ times of L_k and $N_f = \frac{\theta_h \theta_v}{\theta^2}$ is the number of pyramid subspaces in $\mathbb{M}_f$. Hence, the complexity of the update step in (33) and (34) is about $\frac{(2n+1)^2}{N_f}$ times to Equations (13) and (14). To speed up computing, $\theta = \theta'_{max}$ is adopted in practice. Then $n = \lceil \frac{\theta'_{max}}{\theta} \rceil = 1$ and the computational complexity is reduced to $\frac{\theta^2}{\theta_h \theta_v}$ times of (13) and (14). Take $r_{min} = 0.15$ m, $\sigma' = 1\%$, and $\epsilon = 0.01$ as example. With Equation (32), it can be derived that $\theta = \theta'_{max} = 3^\circ$ and $\frac{\theta^2}{\theta_h \theta_v} \approx 0.002$.

C. Particle Birth

Following the method in [35], we generate newborn particles with measurement points Z_k. Since $\mathrm{Z}_k \in \mathbb{M}_f$, the newborn particles are also in $\mathbb{M}_f$. For measurement point $\boldsymbol{z}_k \in \mathrm{Z}_k$, we generate particles with a number of L_b. Then the number of newborn particles in total is $M_k L_b$. The position of each newborn particle is sampled from the Gaussian noise model in Eq. (26). Normally, the velocity of the newborn particle is randomly sampled in a feasible velocity range. However, this random sampling leads to heavy noise, and the convergence speed is slow. Thus we sample the velocity of each newborn particle through an initial velocity estimation method, which is described in Section VI-A and VI-B. The weight of these particles are set to be $\frac{v_{k|k-1}^b}{M_k L_b}$, where $v_{k|k-1}^b = \int \gamma_{k|k-1}(\boldsymbol{x}_k) d\boldsymbol{x}_k$ is a parameter that controls the expected number of newborn objects.

According to [35], the weight of the newborn particle is calculated separately in the update step. Then the weight update Equations (33) and (34) are reformed to represent the survived particles and the newborn particles separately:

$$w_{s,k}^{(i)} = \left[1 - P_d + \sum_{\boldsymbol{z}_k \in \mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}} \frac{P_d g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(i)})}{\kappa_k(\boldsymbol{z}_k) + \mathrm{C}'_k(\boldsymbol{z}_k)}\right] w_{s,k|k-1}^{(i)} \tag{35}$$

$$w_{b,k}^{(j)} = \sum_{\boldsymbol{z}_k \in \mathbb{A}^{\tilde{\boldsymbol{x}}_k^{(i)}}} \frac{w_{b,k|k-1}^{(j)}}{\kappa_k(\boldsymbol{z}_k) + \mathrm{C}'_k(\boldsymbol{z}_k)} \tag{36}$$

$$\mathrm{C}'_k(\boldsymbol{z}_k) = \sum_{j=1}^{M_k L_b} w_{b,k|k-1}^{(j)} + \sum_{j=1}^{L_{s,k}^{\mathbb{A}^{\boldsymbol{z}_k}}} P_d w_{k|k-1}^{(j)} g_k(\boldsymbol{z}_k|\tilde{\boldsymbol{x}}_k^{(j)}) \tag{37}$$

where $L_{s,k}^{\mathbb{A}^{\boldsymbol{z}_k}} \leq L_k^{\mathbb{A}^{\boldsymbol{z}_k}}$ is the number of survived particles whose activation space includes $\boldsymbol{z}_k$. $w_{b,k|k-1}^{(j)}$ is the prior weight of the newborn particle and is $\frac{v_{k|k-1}^b}{M_k L_b}$.

D. Resampling

The resampling step is to constrain the number of particles and prevent degeneration. After resampling, the cardinality expectation and the posterior PHD of X_k, i.e., $\mathbf{E}[|\mathrm{X}_k|]$ and $D_{\mathrm{X}_k}(\boldsymbol{x}_k)$, should not change. With $D_{\mathrm{X}_k}(\boldsymbol{x}_k)$ in the form of (12), the cardinality expectation of X_k is estimated by (3)

$$\mathbf{E}[|\mathrm{X}_k|] = \int \sum_{i=1}^{L_k} w_k^{(i)} \delta(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k^{(i)}) \mathrm{d}\boldsymbol{x}_k = \sum_{i=1}^{L_k} w_k^{(i)} \tag{38}$$

For a single voxel subspace $\mathbb{V}_j$, the cardinality expectation $\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_j)}|]$ can also be estimated with the weights of the particles inside, which is

$$\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_j)}|] = \int D_{\mathrm{X}_k^{(\mathbb{V}_j)}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \sum_{i=1}^{L_k^{(\mathbb{V}_j)}} w_k^{(i)} \tag{39}$$

where $L_k^{(\mathbb{V}_j)}$ represents the number of particles in $\mathbb{V}_j$ at time k. Although some particles outside of $\mathbb{V}_j$ but close to $\mathbb{V}_j$ may

be relevant to $D_{\mathrm{X}_k^{(\mathbb{V}_j)}}(\boldsymbol{x})$, they are not considered and thus the approximately equal sign is used.

Then the resampling is conducted by rejection sampling [36] in each voxel subspace. The voxel subspace rather than the whole map is used. The reason is that if an area contains only low-weight particles, rejection sampling in the whole map may reject all these particles and decrease the occupancy probability of the area falsely. Let $L_{max}^{\mathbb{V}}$ and L_{max} denote the allowed maximum number of particles in a voxel subspace and in the map, respectively, after resampling. $L_{max} = L_{max}^{\mathbb{V}} N_v$. Then the number of particles after resampling is

$$\hat{L}_k^{(\mathbb{V}_j)} = \begin{cases} L_{max}^{\mathbb{V}}, & \text{if } L_k^{(\mathbb{V}_j)} > L_{max}^{\mathbb{V}} \\ L_k^{(\mathbb{V}_j)}, & \text{otherwise} \end{cases} \tag{40}$$

The weight of the particles in $\mathbb{V}_j$ after resampling is identically

$$\hat{w}_k^{(i)} = \frac{\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_j)}|]}{\hat{L}_k^{(\mathbb{V}_j)}} \tag{41}$$

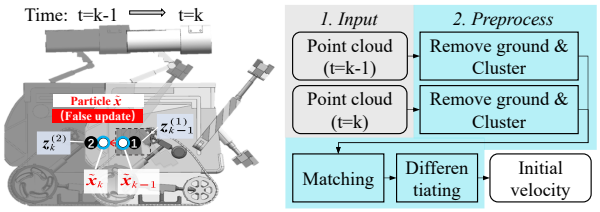
VI. Extensions in Mapping

This section proposes some important extension modules. Firstly, the initial velocity estimation module for newborn particles and a mixture model composed of a static model and a constant velocity model are proposed to reduce the noise in mapping. Then the occupancy status estimation and future status prediction modules, which generate the output designed for motion planning, are expressed. Finally, several useful extra extensions are discussed.

A. Initial Velocity Estimation

The particle-based maps model the obstacles as point objects. This model is very friendly with particle-based tracking but works only at the sub-object level, which will cause non-negligible noise when the obstacle has a relatively large volume. Specifically, the noise is caused by the false update of the particles. Fig. 4 (a) illustrates the false update. The false update leads to many particles with a large weight but a wrong velocity, and further causes heavy noise in predicting the occupancy status of the area out of the FOV or at a future time. When the velocity of the newborn particle is randomly generated, the particle false update problem occurs frequently.

To alleviate the problem and reduce noise, we add an object-level estimation by considering initial velocities for the newborn particles. The procedures to acquire the initial velocities from two adjacent point clouds are shown in Fig. 4 (b). The point cloud that obviously belongs to static obstacles, like the ground, is segmented by considering the height dimension and assigned zero velocity. The rest point cloud is clustered, and the result clusters are matched with the clusters extracted from the last frame. Then the velocity of each cluster can be estimated by differentiating the position of the matched clusters' centers. We use the Euclidean cluster extraction based on k-d tree [37] for clustering and the Kuhn-Munkras (KM) algorithm for matching. In the matching process, the position of the cluster center and the number of points in the cluster are used as features. If a cluster at time k cannot be matched, this cluster is regarded as a new obstacle, and no velocity estimation result is assigned.



(a) Particle false update (b) Initial velocity estimation

Fig. 4. (a) Illustration of the particle false update problem and (b) the initial velocity estimation procedures. In subfigure (a), a dynamic obstacle (a mobile robot) moves rightwards from $t = k-1$ to $t = k$. At $t = k-1$, a measurement point $\boldsymbol{z}_{k-1}^{(1)}$, on the right side of the obstacle, is observed. With $\boldsymbol{z}_{k-1}^{(1)}$, a particle with state $\tilde{\boldsymbol{x}}_{k-1}$ is generated in the particle birth step (Section V-C). Since $\boldsymbol{z}_{k-1}^{(1)}$ doesn't provide velocity, the commonly used way is to give $\tilde{\boldsymbol{x}}_{k-1}$ a random velocity, e.g., a velocity to the left. At $t = k$, following the CV model, $\tilde{\boldsymbol{x}}_{k-1}$ moves to $\tilde{\boldsymbol{x}}_k$. Since the particle's velocity is the opposite to the obstacle's velocity, this particle should belong to noise and its weight should be decreased in the update procedure at k. However, due to the large size of the robot, another measurement point $\boldsymbol{z}_k^{(2)}$, close to $\tilde{\boldsymbol{x}}_k$, may be observed at k and falsely increased the weight of the particle in the update step. We call this problem the particle false update. Note the problem also exists in modeling large-size static obstacles and is more frequent when multiple dynamic and static obstacles exist, in which case particles generated from one obstacle may be falsely updated with the measurement from another obstacle.

The velocity estimated by position differentiating between two adjacent inputs is quite noisy because of three main reasons. The first reason is that the position error of the point cloud measurement is amplified and propagated to the velocity estimation by differentiating. The second reason is that the position of a cluster center varies when using point clouds observed from different angles, and the third is that the clustering and matching result might contain many errors in complex environments. We have assumed that the measurement noise of the point cloud is Gaussian noise. Thus the noise caused by the first reason is still Gaussian noise. However, the noise caused by the latter two reasons can be very random. Therefore, the estimated velocity cannot be regarded as the velocity measurement and utilized in the update step. We thus adopt this estimated velocity as a reference of initial particle velocities in the particle birth step on the basis of a mixture model. Details are presented in Section VI-B.

B. Mixture Model

To further reduce the noise caused by the false update and model static objects better, we adopt a mixture model. The mixture model supposes the state of a point object is the combination of two components, i.e., $\boldsymbol{x}_k = \lambda_1 \boldsymbol{x}_{k,d} + \lambda_2 \boldsymbol{x}_{k,s}$. λ_1 and λ_2 are weight coefficients that satisfy $\lambda_1 + \lambda_2 = 1$. $\boldsymbol{x}_{k,d}$ is a dynamic object state component. $\boldsymbol{x}_{k,s}$ is a static object state component with zero velocity. Since the environment is unknown, the value of λ_1 and λ_2 should not be fixed but should be updated in the filtering process. We assume that the objects in one voxel subspace, a small subspace, have the same weight coefficients. In each voxel subspace, the dynamic object states and the static object states can be regarded as two independent RFSs, $\mathrm{X}_d^{(\mathbb{V})}$ and $\mathrm{X}_s^{(\mathbb{V})}$, respectively. Then λ_1 and λ_2 is estimated by the ratio between $|\mathrm{X}_d^{(\mathbb{V})}|$ and $|\mathrm{X}_s^{(\mathbb{V})}|$.

Using the property described in Equation (3) and the same deduction in Equation (38), $|\mathrm{X}_d^{(\mathbb{V})}|$ and $|\mathrm{X}_s^{(\mathbb{V})}|$ can be estimated with the weight summation of the particles. A particle $\tilde{\boldsymbol{x}}^{(i)}$ might correspond to $\boldsymbol{x}_{k,d}$ or $\boldsymbol{x}_{k,s}$, depending on the transition density $\pi_{k|k-1}(\boldsymbol{x}_{k,d}|\tilde{\boldsymbol{x}}_{k-1}^{(i)})$ or $\pi_{k|k-1}(\boldsymbol{x}_{k,s}|\tilde{\boldsymbol{x}}_{k-1}^{(i)})$. For simplicity, the particle's speed, namely $V(\tilde{\boldsymbol{x}}^{(i)})$ is used as the feature to determine the correspondence, and the Dempster Shafer theory (DST) is adopted to approximate $|\mathrm{X}_d^{(\mathbb{V})}|$ and $|\mathrm{X}_s^{(\mathbb{V})}|$. The time subscript k is omitted to simplify the notation. The universe of DST, in our case, is $U = \{\mathrm{d}, \mathrm{s}\}$, where d is the dynamic hypothesis and s is the static hypothesis. The power set is $2^U = \{\emptyset, \{\mathrm{d}\}, \{\mathrm{s}\}, U\}$. The mass function $m(A)$ has the properties that $\sum_{A\in 2^U} m(A) = 1$ and $m(\emptyset) = 0$.

We suppose the weight summation of particles that satisfy $V(\tilde{\boldsymbol{x}}^{(i)}) = 0$ is $W_s^{(\mathbb{V})}$ and the weight summation of particles that satisfy $V(\tilde{\boldsymbol{x}}^{(i)}) \geq \hat{V}$ is $W_d^{(\mathbb{V})}$, where $\hat{V}$ is a threshold suggesting that particles with a velocity larger than $\hat{V}$ correspond to $\boldsymbol{x}_{k,d}$ rather than $\boldsymbol{x}_{k,s}$. The particles with $0 < V(\tilde{\boldsymbol{x}}^{(i)}) < \hat{V}$, however, can correspond to $\boldsymbol{x}_{k,d}$ or $\boldsymbol{x}_{k,s}$. Suppose the weight summation of these particles is $W_{d,s}^{(\mathbb{V})}$. Then the masses are defined with

$$\begin{aligned} m(\{\mathrm{d}\}) &= \frac{W_d^{(\mathbb{V})}}{W^{(\mathbb{V})}}, \quad m(\{\mathrm{s}\}) = \frac{W_s^{(\mathbb{V})}}{W^{(\mathbb{V})}} \\ m(U) &= \frac{W_{d,s}^{(\mathbb{V})}}{W^{(\mathbb{V})}}, \quad W^{(\mathbb{V})} = W_d^{(\mathbb{V})} + W_s^{(\mathbb{V})} + W_{d,s}^{(\mathbb{V})} \end{aligned} \tag{42}$$

which describes the basic belief. Then the belief and the plausibility are

$$\begin{aligned} &bel(\{\mathrm{d}\}) = m(\{\mathrm{d}\}),\ pl(\{\mathrm{d}\}) = m(\{\mathrm{d}\}) + m(U) \\ &bel(\{\mathrm{s}\}) = m(\{\mathrm{s}\}),\ pl(\{\mathrm{s}\}) = m(\{\mathrm{s}\}) + m(U) \end{aligned} \tag{43}$$

According to DST, the probability is between the belief and the plausibility. We simply take the median as the probability estimation, which is $pr(\cdot) = \frac{bel(\cdot)+pl(\cdot)}{2}$. The cardinalities of dynamic and static objects in a voxel are approximated by

$$\begin{aligned} |\mathrm{X}_d^{(\mathbb{V})}| &\approx W^{(\mathbb{V})} \left[bel(\{\mathrm{d}\}) + pl(\{\mathrm{d}\})\right]/2 \\ |\mathrm{X}_s^{(\mathbb{V})}| &\approx W^{(\mathbb{V})} \left[bel(\{\mathrm{s}\}) + pl(\{\mathrm{s}\})\right]/2 \end{aligned} \tag{44}$$

Then the coefficients λ_1 and λ_2 are estimated with the ratio of the cardinalities:

$$\begin{aligned} \lambda_1 &= \frac{|\mathrm{X}_d^{(\mathbb{V})}|}{|\mathrm{X}_d^{(\mathbb{V})}| + |\mathrm{X}_s^{(\mathbb{V})}|} = \frac{W_d^{(\mathbb{V})}}{W^{(\mathbb{V})}} + \frac{1}{2}\frac{W_{d,s}^{(V)}}{W^{(\mathbb{V})}} \\ \lambda_2 &= \frac{|\mathrm{X}_s^{(\mathbb{V})}|}{|\mathrm{X}_d^{(\mathbb{V})}| + |\mathrm{X}_s^{(\mathbb{V})}|} = \frac{W_s^{(\mathbb{V})}}{W^{(\mathbb{V})}} + \frac{1}{2}\frac{W_{d,s}^{(V)}}{W^{(\mathbb{V})}} \end{aligned} \tag{45}$$

In the prediction step, the motion model in Eq. (22) from time $k-1$ to k turns to

$$\boldsymbol{x}_k = \lambda_1 \left[f_Q\left(\boldsymbol{x}_{k-1,d}\right) + \boldsymbol{\Sigma}\right] + \lambda_2 \left[\boldsymbol{x}_{k-1,s} + \boldsymbol{\Sigma}'\right] \tag{46}$$

where $\boldsymbol{\Sigma}'$ is the Gaussian noise whose velocity dimension is zero. At the particle level, with Eq. (45) it can be derived that particles satisfying $V(\tilde{\boldsymbol{x}}^{(i)}) > \hat{V}$ and half number of particles satisfying $0 < V(\tilde{\boldsymbol{x}}^{(i)}) < \hat{V}$ correspond to $\boldsymbol{x}_{k-1,d}$, and their prior states are sampled with Eq. (24). The rest particles correspond to $\boldsymbol{x}_{k-1,s}$, and their prior states are sampled with $\tilde{\boldsymbol{x}}_{k|k-1}^{(i)} = \tilde{\boldsymbol{x}}_{k-1}^{(i)} + \boldsymbol{u}'$, where $\boldsymbol{u}'$ doesn't contain velocity noise. The update step remains the same because the measurement does not contain velocity observation.

In the particle birth step, the velocities of the newborn particles are assigned based on λ_1 and λ_2 in the corresponding voxel subspace, and the initial velocity estimation results in Section VI-A. If a measurement point is labeled static, e.g., the ground, in the initial velocity estimation procedure, the velocities of the particles generated from this point are all zero. Otherwise, the mixture model is used. The number of dynamic particles generated from a measurement point is $\lambda_1 L_b$, and the number of static particles is $\lambda_2 L_b$. According to the discussion of estimation noise in Section VI-A, the velocities of dynamic particles are composed of two parts: velocities sampled from a Gaussian distribution and random velocities. Since real-world sensors usually contain heavy noise, we set a large variance for the Gaussian distribution, and the particles with random velocities take $0.5\lambda_1 L_b$. If too few particles exist in the voxel subspace where the measurement point belongs, e.g., the situation when the voxel subspace is observed for the first time, an initial guess of $\lambda_1 = \lambda_2 = 0.5$ is used.

C. Occupancy Status Estimation

At an arbitrary point $\boldsymbol{p}$ in the map, the occupancy status is estimated by the cardinality expectation of point objects in a small neighborhood space of $\boldsymbol{p}$. Assume the point objects representing an obstacle are uniformly distributed in the space occupied by the obstacle and has no overlap. The distance between two adjacent point objects is l'. Then, in a cubic neighborhood space with side length l' and centered by $\boldsymbol{p}$, there should be either one or no point object. In our case, the point cloud is pre-filtered by a voxel filter with resolution Res. Thus, $l' = Res$. Denote the cubic neighborhood space by $\mathbb{V}_p$ and the RSF composed of the point objects in $\mathbb{V}_p$ by $\mathrm{X}_k^{\mathbb{V}_p}$. According to Equation (3) and (12), the expectation of the cardinality of $\mathrm{X}_k^{\mathbb{V}_p}$ is calculated with

$$\mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|] = \int D_{\mathrm{X}_k^{\mathbb{V}_p}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \sum_{\tilde{\boldsymbol{x}}_k^{(i)} \in \mathbb{V}_p} w_k^{(i)} \tag{47}$$

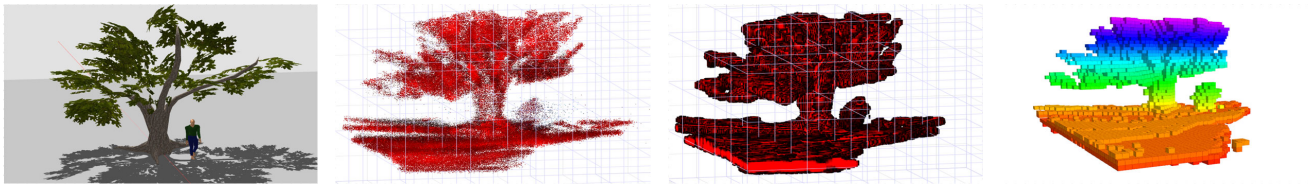
which is the weight summation of particles in $\mathbb{V}_p$.

We denote the occupancy probability at $\boldsymbol{p}$ by $P_{occ}(\boldsymbol{p})$. Since $\mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|]$ represents the expectation of the point object number in $\mathbb{V}_p$, and $\mathbb{V}_p$ is occupied as long as there is a point object inside, $P_{occ}(\boldsymbol{p})$ can be estimated by $P_{occ}(\boldsymbol{p}) = \mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|]$, if $\mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|] \leq 1$. In practice, $\mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|]$ can be larger than one because of the noise in the input data and camera motions. If $\mathbf{E}[|\mathrm{X}_k^{\mathbb{V}_p}|] > 1$, $P_{occ}(\boldsymbol{p}) = 1$ is adopted.

The occupancy probability of a general voxel subspace, such as the voxel subspace $\mathbb{V}_i$ defined in Section IV with side length l, is estimated with

$$P_{occ}(\mathbb{V}_i) = \begin{cases} \mathrm{Min}\{\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_i)}|] \cdot (\frac{l'}{l})^3, 1\}, & \text{if } l \leq l' \\ \mathrm{Min}\{\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_i)}|], 1\}, & \text{otherwise} \end{cases} \tag{48}$$

where $\mathbf{E}[|\mathrm{X}_k^{(\mathbb{V}_i)}|]$ is calculated with weight summation of particles in $\mathbb{V}_i$ like Equation (47). A scale factor $(\frac{l'}{l})^3$ is applied because the volume of $\mathbb{V}_i$ is $(\frac{l}{l'})^3$ times smaller than



(a) RGB image view (b) Particle view (c) Continuous occupancy status (d) Voxelized occupancy status

Fig. 5. Occupancy status estimation of the DSP map in a scenario with a static obstacle and a dynamic obstacle. (a) shows the scenario with an RGB image. (b) illustrates the particles in the current DSP map. The color is painted in the HSV color space, with the hue keeping red and the saturation indicating the particle's weight. Higher saturation, i.e., a greater proportion of red relative to black, indicates a larger weight. The estimated occupancy status in a continuous form is shown in (c). Higher saturation indicates a larger occupied probability. (d) utilizes the voxel subspaces to calculate a 3d grid map. The color of the grids changes with their z-axis height. The obstacles are from a scenario in the pedestrian street world in Fig. 8, and the map is built with the recorded flight data used in Section VIII-C. Some parts of the tree and the ground are missing because they haven't been observed.

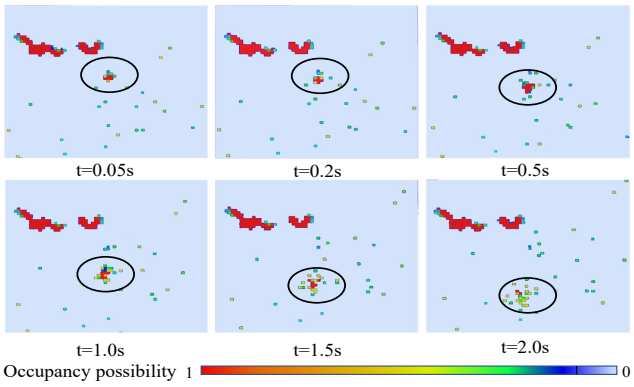


Fig. 6. Future occupancy status prediction. We predict the future occupancy status of the scenario in Fig. 5 at six future times. Only the layer $z = 1.6m$ is shown to have a clear view. The black ellipses show the predicted occupancy status of the pedestrian. The pedestrian walks with a constant velocity to the bottom side. The red occupied area in the upper left corresponds to the tree's trunk and branches.

$\mathbb{V}_p$. If $l > l'$, the estimated point object number $\mathbf{E}[|\mathbf{X}_k^{(\mathbb{V}_i)}|]$ can be larger than one even if the estimation has no error. If $\mathbf{E}[|\mathbf{X}_k^{(\mathbb{V}_i)}|] > 1$, $P_{occ}(\mathbb{V}_i) = 1$ is adopted. With the occupancy probability, a probability threshold can then be used to get a binary occupancy status, i.e., occupied or free. Fig. 5 shows an example occupancy estimation result in a scenario with a static obstacle and a dynamic obstacle. The voxelized map is shown in Subfigure (d). It should be noted that this voxelized map doesn't suffer from the grid size problem because the mapping process is realized in the continuous space.

D. Future Occupancy Status Prediction

Predicting the future occupancy status is very useful for motion planning in dynamic environments. In our DSP map, the future occupancy status prediction is fulfilled by predicting the position of the particles according to the motion model in (22) and (46), and then using (47) and (48) for occupancy status estimation. Fig. 6 presents the future occupancy estimation results of the map shown in Fig. 5. The occupancy status of the static obstacle, the tree, almost stays the same in each plot. The occupied position of the dynamic obstacle, the pedestrian, is predicted to move down with the CV model. The occupied grids are spreading, and their occupancy probabilities are getting lower as the prediction time increases. The reason is the uncertainty in velocity estimation, which is reflected by the variance of particles' velocities. The estimation uncertainty also causes noise in other parts of the plots. Since future occupancy status prediction in dynamic environments has inevitable uncertainty, the predicted occupancy probability can be used as the risk in motion planning algorithms.

E. DSP Static Map

By assuming the point objects as static objects and using only the static model described in Section VI-B, the DSP map turns to a static map, named the DSP-Static map. In this case, the number of particles used in this map can be very small since the velocity dimension is not considered, which means the DSP-Static map is more computationally efficient. Compared to the voxel map for static environments, the DSP-Static map is continuous and free from the voxel size problem. In the experiment section, the DSP-Static map is also tested.

VII. Implementation

This section describes an implementation of the DSP map. The implementation includes the data structure to realize subspace division and particle storage, and the specific algorithms used to build the DSP map.

A. Data Structure

The number of particles in the map can be up to one million. Thus, storage and operation of the particles are important to efficiency. Three techniques are utilized to improve efficiency: 1) The voxel subspaces are used to store particles while the pyramid subspaces only store the indexes of particles; 2) Large arrays with preallocated size rather than unordered sets, which represent RFSs natively, are used to store elements in an RFS; 3) The operations of adding and deleting particles are simplified using a flag variable. The first technique is to reduce memory consumption, while the second is to avoid dynamic memory allocation and increase the cache hit rate. The last technique is employed to simplify operations on particles. Detailed data structure can be found in Appendix D.

B. Mapping Algorithms

A flowchart showing the order of the algorithms used for mapping is presented in Fig. 7. After the input point cloud

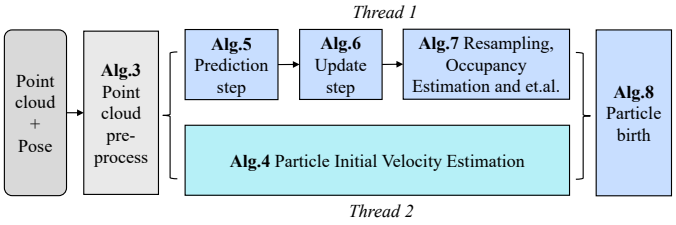


Fig. 7. Flowchart of the algorithms used to implement the DSP map.

is pre-filtered by a voxel filter with resolution Res and transformed to the map frame, two threads are opened to run the particle initial velocity estimation in parallel with prediction, update and resampling. Resampling, occupancy estimation and mixture model coefficients calculation are conducted in one loop to improve efficiency. Future occupancy status prediction is not shown but is realized by predicting particle states to more future times in the prediction step. Detailed algorithms can be found in Appendix E.

VIII. Experimental Results

This section first evaluates the velocity estimation precision of the DSP map since velocity estimation ability is a major difference between static maps and dynamic maps. Then the DSP map is tested with different parameters to evaluate the effect of the parameters on mapping performance and identify the best parameter values. With the identified parameter values, the mapping performance is further compared with existing works in different simulation worlds with different resolutions. To test the practicality of using the DSP map on robotics platforms, we also show computational efficiency comparison results on an NVIDIA Jetson board. Finally, a demo of using this map for drone obstacle avoidance is presented.

A. Velocity Estimation

The velocity estimation experiments were conducted with the data collected in an indoor testing field with the Nokov motion capture system. An Intel Realsense d435 camera was fixed at an edge of the testing field to collect the point cloud. Two pedestrians, wearing helmets with markers, walked around in the testing field, and their trajectories estimated by the motion capture system were recorded synchronously with the point cloud. The experiments can be divided into two groups. In the first group, the pedestrians tried to walk at a constant velocity. In the second group, the pedestrians walked randomly and freely. Fig. 8 (a) shows the data collection scenario.

We compared the velocities estimated by four different point-cloud-based methods. The first method differentiates the center position of two matched clusters, and no filter is adopted. The matching is achieved by the KM algorithm. The second is a multi-object tracker realized by the KM algorithm and Kalman Filters (KF) with a CV model. The input of the KF is the center positions of matched clusters. The third method is the DSP map with the suffix "Random", whose newborn particles have random velocities. The fourth method is the DSP map with the suffix "Dynamic", whose newborn particles

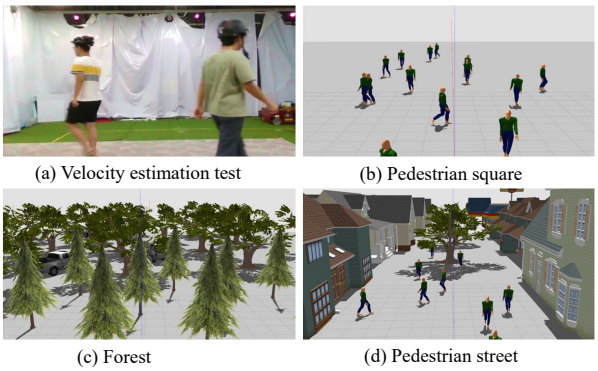


Fig. 8. Real-world scenario for velocity estimation test (a), and simulation worlds for mapping performance comparison (b)-(d). The pedestrian square world (b) contains only dynamic obstacles (the ground is excluded in the evaluation tests), while the forest world (c) contains only static obstacles. The pedestrian street world (d) contains both static obstacles and dynamic obstacles.

TABLE II
Velocity Estimation Results of Different Methods.

Group	Constant velocity			Random walking		
Metric	RMSE	Var.	MBD	RMSE	Var.	MBD
KM-Diff.	0.583	-	-	0.656	-	-
KM-KF	0.286	0.479	0.470	0.309	0.481	0.476
DSP-Random	0.332	1.083	0.641	0.353	1.077	0.641
DSP-Dynamic	**0.277**	**0.318**	**0.398**	**0.302**	**0.335**	**0.417**

consider initial velocity estimation. Since our maps do not explicitly segment the objects, the state of a pedestrian was estimated with the particle cluster near the pedestrian's real position. Table II presents the estimation results of the two groups. We consider a pedestrian walking from one side of the testing field to another a tracklet. Over thirty tracklets were collected in each group. Fig. 9 shows the velocity estimation curves of a typical tracklet.

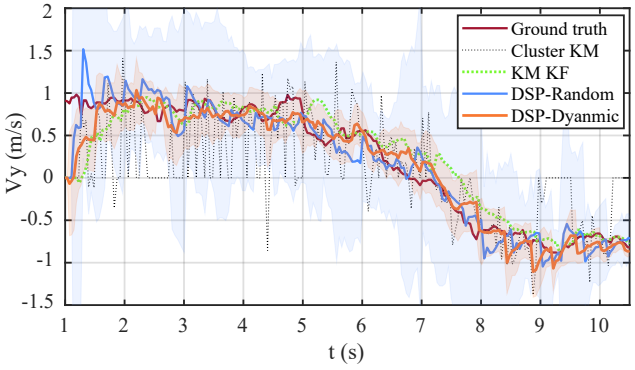


Fig. 9. Velocity estimation curves of a typical tracklet. The serrated orange and blue background show the variance of the estimation results from the DSP-Dynamic map and the DSP-Random map, respectively. At $t = 6s$ to $t = 8s$, the pedestrian turns back.

Three metrics are used for evaluation. The root mean square error (RMSE) reflects the estimation precision evaluated with the mean of the velocity estimation distribution and the ground truth from the motion capture system. The Var. is the mean variance of the different axes on every point. The differentiating method outputs a single value rather than a

distribution, and thus a dash is placed in its Var. in Table II. For a particle-based map, a large variance means the particles would disperse to a large scale of the area and cause much noise in the map. Mean Bhattacharyya distance (MBD) measures the similarity between the estimated and ground-truth velocity distribution. MBD considers both mean value and variance and is a composite metric. The results in Table II show that DSP-Dynamic performs best with all three metrics. The differentiated velocity has a large RMSE, and the error can be huge sometimes, as Fig. 9 shows. Using KF can reduce the error, but the Var. is over 30% larger than that of DSP-Dynamic, and the MBD is over 12% larger. Compared to DSP-random, DSP dynamic decreases over 14% on RMSE, over 68% on Var., and over 34% on MBD, showing the importance of the initial velocity estimation.

B. Mapping with Different Parameters

Inspired by [2] [5] [24], we evaluated the occupancy mapping performance by assessing the binary classification results, i.e., free or occupied, of the voxel subspaces. The metrics include average precision, recall, F1-Score[2], and time consumption of a complete mapping process. The tested parameters include maximum particle number L_{max}, the voxel size Res of the voxel filter for the point cloud pre-process before mapping, the pyramid subspace angle θ, and the voxel subspace size l. When Res is larger, the measurement point number M_k is smaller. Each parameter was tested with three levels. A full factorial experiment was conducted with data collected in the pedestrian street world (Fig. 8 (d)), where both static and dynamic obstacles exist. The world is built in the Gazebo[3] simulation software. A simulated IRIS quadrotor with a Realsense camera is controlled manually to collect point cloud and pose data for mapping.

To generate the ground truth occupancy map, we densely and uniformly sampled points from the mesh surfaces of the static objects in the world and generated a Euclidean Distance Field (EDF) for the objects using the sampled points. The EDF changes caused by pedestrians are updated online at each evaluation step using the mesh and pose of the pedestrians. A voxel subspace was considered occupied if the distance value at the voxel's center position was no larger than $\frac{l}{2}$ and free otherwise. We also added a label array to distinguish the observed and unobserved labels of the voxels. The array is updated using the ray-casting approach with dense rays. Only the observed voxels were considered in the evaluation.

The result is shown in Fig. 10 (a). When L_{max} increases from 0.8 million to 2.4 million, the precision does not have a noticeable change, while the average time consumption increases from 50 ms to 160 ms. The recall rises from 0.22 to 0.30 when L_{max} increases to 1.6 million but almost remains unchanged when L_{max} increases further. The F1-Score has the same trend as the recall. Raising Res leads to fewer measurement points in the point cloud and shows a positive effect on precision but a negative effect on recall. The balanced metric F1-Score reaches the maximum value of 0.38 when Res is 0.1 m. The time consumption decreases as Res increases.

The pyramid subspace angle θ slightly affects precision, recall, and F1-Score. The F1-Score increases merely 0.005 when θ grows from one degree to five degrees. Meanwhile, the time consumption increases from 67 ms to 144 ms. The voxel subspace size l positively correlates to all the metrics. When l is larger, the number of voxels to classify is less, and the occupancy status of a voxel is easier to determine because more measurement points and particles are contained in one voxel. As a result, the precision, recall, and F1-Score all improve. However, a larger voxel size is usually unfavorable in motion planning. The time consumption rises because the particle operations in Algorithm 1 are slower with more particles in one voxel subspace.

To achieve the best F1-Score with an acceptable time consumption (about 100 ms), $L_{max} = 1.6 \times 10^6$, $Res = 0.1$, and $\theta = 3^\circ$ are chosen. We further compare the performance of our map with other maps using different resolutions in the following experiment.

C. Mapping Performance Comparison

In this experiment, we compared our DSP map with a static local occupancy map named Ewok [7] and a state-of-the-art particle-based dynamic occupancy map named K3DOM [5]. K3DOM is the only 3D dynamic occupancy map with a released code currently. We also compared our map with two variants: one uses newborn particles with random velocities and considers the constant velocity model only, i.e., extensions in Section VI-A and VI-B are not adopted; another uses static newborn particles and considers the static motion model, i.e., the extension in Section VI-E. To distinguish the variants, we call our map with particle initial velocity estimation and mixture model DSP-Dynamic map, and the variants DSP-Random map and DSP-static map, respectively.

K3DOM runs on NVIDIA RTX 2060 GPU, and the rest maps run on AMD Ryzen 4800HS CPU in the tests. The map size (l_x, l_y, l_z) is (10 m, 10 m, 6 m). The rest parameters in K3DOM and Ewok remain the same as the original settings in the released code. No voxel filter is used for point cloud pre-processing in K3DOM and Ewok to reach their best performances. In DSP map and its variants, the initial weight of the particle is 0.0001. Three different voxel sizes, from 0.1 m to 0.3 m, were tested in the simulation worlds shown in Fig. 8 (b) to (d). Using different occupancy probability thresholds, which determine the binary status, i.e., occupied or free, we draw precision-recall curves in Fig 10 (b). Snapshots of different maps can be found in Fig. 11 and Fig. 12.

In Fig 10 (b), a larger area under the curve (AUC) suggests a better overall performance in classifying the occupancy status with different thresholds. The specific AUC values are presented in Table III. The DSP-Dynamic map has the largest AUC in the two worlds with pedestrians and a comparable AUC with Ewok and DSP static map in the static forest world. When the voxel size is 0.1 m and 0.2 m, the recall of Ewok and K3DOM is relatively low because of the gaps and inconsistencies in high-resolution grid maps. Red dashed

[2]F1-Score $= \frac{2precision \cdot recall}{precision + recall}$ is a balanced metric of precision and recall.

[3]Gazebo simulation software: https://gazebosim.org/home

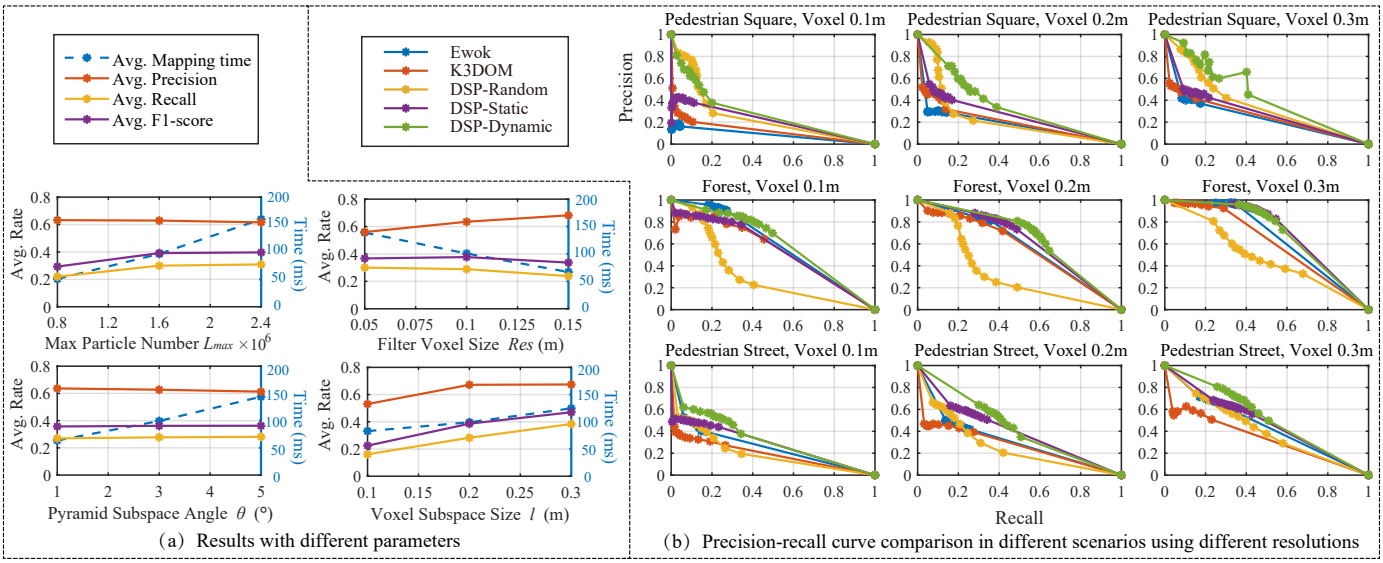


Fig. 10. (a) Mapping performance with different parameters. Each parameter has three levels and is analyzed in an individual plot. (b) Precision-recall curve comparison. Results in different worlds are shown in different rows, and results with different voxel sizes are shown in different columns.

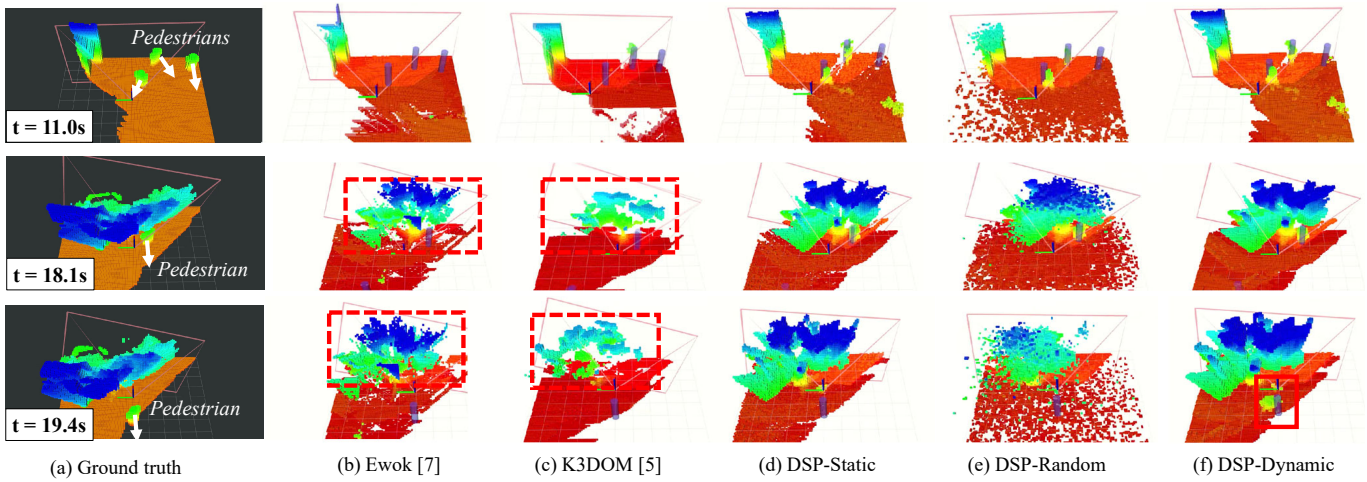
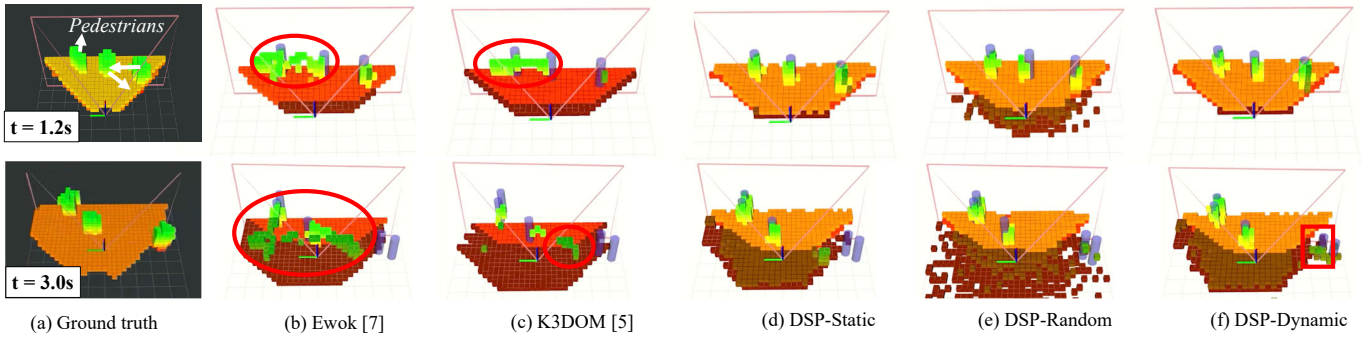


Fig. 11. Snapshots of different maps in the pedestrian street world when the mapping resolution is 0.1 m. A pedestrian moves along the direction indicated by the white arrow from $t = 18.1$s to $t = 19.4$s. The color of the voxels changes with their z-axis height. The pink lines show the current FOV of the camera. The voxels in the FOV are painted brighter than those out of the FOV. The semi-transparent blue cylinders in the maps present the real position of the pedestrians. The red rectangle in the DSP-Dynamic map at $t = 19.4$s outlines an area out of the FOV corresponding to a pedestrian. The pedestrian is out of the FOV, and thus, its occupancy status is predicted. Red dashed boxes show typical gaps and inconsistencies in grid maps when the resolution is high.

boxes in Fig. 11 illustrate the gaps and inconsistencies. In Fig. 12, the areas in the red ellipses show that Ewok has noticeable trail noise, which can lower the precision, when the voxel size is 0.3 m. The dynamic occupancy map K3DOM has less trail noise. In comparison, our DSP-Dynamic map doesn't suffer from gaps, inconsistencies, or trail noise.

DSP-Random, which doesn't have initial velocity estimation and uses only the CV model, has obvious noise in the area out of FOV, especially when representing static obstacles. Column (e) in Fig. 11 shows the noise. Consequently, the AUC of DSP-Random is the smallest in the forest world. DSP-Static adopts only the static motion model and achieves the best AUC in the forest world when the voxel size is 0.2 m and 0.3 m. However, it cannot predict the future occupancy status of dynamic obstacles, and the AUCs in the worlds with pedestrians are smaller than DSP-Dynamic. The red rectangles in Column (f) in Fig. 11 and Fig. 12 show the predicted occupancy status of a pedestrian out of the FOV in the DSP-Dynamic map.

Table IV shows the best F1-Score, i.e., the highest classification performance that each map reaches with different probability thresholds in Fig 10 (b). When the testing scenario is the forest world, and the voxel size is 0.3 m, the DSP-Dynamic map's score is slightly lower than the DSP-Static map's. In all other situations, DSP-Dynamic has the highest best F1-Score. Note in the pedestrian square world, where only dynamic obstacles exist, the dynamic map K3DOM has a lower best F1-Score than the static map Ewok when the voxel size is 0.3 m. The reason is that although Ewok has a low precision due to its heavy trail noise, its recall rate is



(a) Ground truth (b) Ewok [7] (c) K3DOM [5] (d) DSP-Static (e) DSP-Random (f) DSP-Dynamic

Fig. 12. Snapshots of different maps in the pedestrian square world when the mapping resolution is 0.3 m. The red ellipses show the trail noise caused by moving pedestrians in Ewok [7] and K3DOM [5]. From $t = 1.2$s to $t = 2.8$s, the movement of three pedestrians causes more trail noise. The red rectangle in Column (f) shows the predicted occupancy of a pedestrian out of FOV.

higher than K3DOM's. However, from Table III, it can be seen that the AUC, which evaluates the overall classification performance when using different occupancy probability thresholds, of K3DOM is still higher than that of Ewok.

The average F1-score and AUC of our DSP-Dynamic map in different worlds with different resolutions are 0.46 and 0.47, respectively. In comparison, the average F1-score and AUC of the existing particle-based dynamic occupancy map K3DOM are 0.33 and 0.37, respectively. Our map increases the F1-score by 39.4% and AUC by 27.0%. If only the two worlds that contain dynamic obstacles are considered, the average F1-score and AUC increase from 0.24 to 0.39 (62.5% increase) and 0.27 to 0.40 (48.1% increase), respectively.

TABLE III
AUC COMPARISON

World	Pedestrian square			Forest			Pedestrian street		
Voxel size (m)	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
Ewok [7]	0.10	0.22	0.27	0.59	0.59	0.66	0.28	0.34	0.46
K3DOM [5]	0.15	0.23	0.28	0.55	0.57	0.61	0.27	0.33	0.37
DSP-Random	0.24	0.24	0.36	0.32	0.35	0.50	0.27	0.31	0.44
DSP-Static	0.24	0.29	0.32	0.56	**0.61**	**0.69**	0.34	0.42	0.49
DSP-Dynamic	**0.29**	**0.36**	**0.43**	**0.60**	0.59	0.65	**0.36**	**0.43**	**0.50**

TABLE IV
BEST F1-SCORE COMPARISON

World	Pedestrian square			Forest			Pedestrian street		
Voxel size (m)	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
Ewok [7]	0.07	0.19	0.24	0.42	0.50	0.52	0.24	0.32	0.40
K3DOM [5]	0.14	0.19	0.22	0.53	0.53	0.44	0.27	0.32	0.32
DSP-Random	0.24	0.24	0.35	0.29	0.29	0.44	0.25	0.28	0.39
DSP-Static	0.17	0.24	0.29	0.48	**0.59**	**0.66**	0.30	**0.41**	0.49
DSP-Dynamic	**0.26**	**0.36**	**0.43**	**0.58**	**0.59**	0.65	**0.36**	**0.41**	**0.50**

D. Robotics Platform Efficiency Tests

This section first compares the efficiency of Ewok [7], K3DOM [5], and our DSP map (with particle initial velocity estimation and mixture motion model) on NVIDIA Jetson Xavier NX, which is a small computing board widely used on robotics platforms. Xavier NX has a 384-core NVIDIA Volta GPU with 48 Tensor Cores and a 6-core NVIDIA Carmel ARM v8.2 CPU. K3DOM [5] runs on the GPU, and the rest maps run on the CPU. The average time consumption of each map with different voxel sizes is shown in Fig. 13. The map size in the test is (10 m, 10 m, 6 m).

When the voxel size is 0.1 m, our DSP map is the fastest. The existing particle-based dynamic occupancy map K3DOM is 4.5 times slower than the DSP map. The static map Ewok runs fastest when the voxel size is 0.2 m or 0.3 m. K3DOM is the second fastest, and our DSP map is the slowest. However, with the results in Fig. 10 (a), we can further raise the computational efficiency of our map by sacrificing a little performance on the F1-Score. Fig. 10 (a) indicates that decreasing the pyramid subspace angle θ from 3° to 1° can reduce the computation time while the F1-Score drops merely 1%. In addition, increasing the filter voxel size Res from 0.1 m to 0.15 m can also reduce the computation time, and the F1-Score decreases 12% accordingly. If $\theta = 1°$ is used in the tests on Xavier NX, the DSP map's computation time is only 0.56 times that of K3DOM's when the voxel size is 0.2 m and is close to K3DOM's when the voxel size is 0.3 m. If $Res = 0.15m$ is further adopted, the DSP map's computation time is shorter than the K3DOM's for all tested voxel sizes.

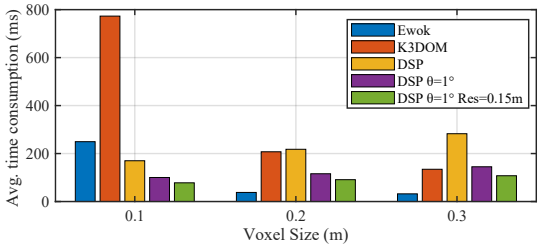


Fig. 13. Time consumption of different maps on Jeston Xavier NX.

We also tested the computation efficiency of our DSP map on two other onboard computers for robotics platforms: an Intel NUC with a Core i7-10710u CPU and an Up core board with an Intel Atom x5-z8350 CPU. When $\theta = 1°$ and $Res = 0.15$ m are adopted, and the voxel size is 0.2 m, the average time consumption on the two boards is about 133 ms and 254 ms, respectively. For robotics obstacle avoidance tasks without fast movement, a smaller map can be used to reduce time consumption. For example, when the map size is reduced to (8 m, 8 m, 3 m), the average time consumption on the Up

core board is below 150 ms.

In Appendix B, we present a test with omnidirectional and multi-channel Lidar point cloud data. The time consumption is about two times when using point cloud data from the Realsense camera, which has a limited FOV. Improvements in computational efficiency will be conducted further to realize real-time mapping with multi-channel Lidars.

E. Applications

Fig. 14 presents several snapshots of building the DSP-Dynamic map in different scenarios. The localization was realized by a Realsense T265 tracking camera, and the point cloud was from a Realsense d435 camera. To further demonstrate the effectiveness and efficiency of our map in robotic systems. We deployed the DSP-Dynamic map on a mini quadrotor with a weight of only 320 grams and utilized a sampling-based motion planning method [38] to realize obstacle avoidance in environments with static and dynamic obstacles. The method samples motion primitives and evaluates the collision risk of each motion primitive with the current and predicted particles in the DSP-Dynamic map. Details can be found in [38]. The point cloud was collected from a Realsense d435 camera, and everything, including mapping and motion planning, was performed on the CPU of a low-cost Up core computing board. Fig. 15 shows the testing scenarios. The testing demos can be found at https://youtu.be/seF_Oy4YbXo.

IX. Conclusion

This paper presents a novel dual-structure particle-based 3D local map, named DSP (dynamic) map, that allows continuous occupancy mapping of dynamic environments. Voxel subspaces and pyramid-like subspaces are adopted to achieve efficient updates in continuous space. The initial velocity estimation and a mixture model are considered to reduce noise. Experiments show that the DSP map can increase the dynamic obstacle velocity estimation performance by over 30% on MBD, compared to other tested point-cloud-based methods. In occupancy status estimation tests, the DSP map increases the F1-Score of the state-of-the-art particle-based occupancy map from 0.33 to 0.46 (39.4% increase) and the AUC from 0.37 to 0.47 (27.0% increase) on average. Furthermore, efficiency tests and a real-world application demo demonstrated the broad prospect of this map in obstacle avoidance tasks of small-scale robotic systems. Future works will consider two main points. The first is to introduce semantic information to this map to better identify and model different obstacles and further predict their future states with multiple hypotheses. The second is to connect this dynamic local map to a global static map to achieve global mapping in dynamic environments.

Appendix A
Lower Bound Distance Calculation

This appendix calculates the lower bound distance from a point object to a measurement point whose azimuth angle and zenith angle differences with the point object are no less than θ'. Fig. 16 shows two limiting cases where the zenith and azimuth angle difference are θ', respectively. In subplot (a), when $\boldsymbol{P}_{\boldsymbol{z}_k}$ is in the same vertical plane with $\boldsymbol{P}_{\boldsymbol{x}_k}$ and $\boldsymbol{P}_{\boldsymbol{x}_k}\boldsymbol{P}_{\boldsymbol{z}_k} \perp \boldsymbol{P}_{\boldsymbol{z}_k}\boldsymbol{O}$, the minimum distance between $\boldsymbol{P}_{\boldsymbol{x}_k}$ and $\boldsymbol{P}_{\boldsymbol{z}_k}$ exists and is $|\boldsymbol{P}_{\boldsymbol{x}_k}\boldsymbol{P}_{\boldsymbol{z}_k}| = r_k sin\theta'$. In Subplot (b), $|\boldsymbol{P}_{\boldsymbol{x}_k}\boldsymbol{P}_{\boldsymbol{z}_k}| \geq |\boldsymbol{P}'_{\boldsymbol{x}_k}\boldsymbol{P}'_{\boldsymbol{z}_k}|$, where $|\boldsymbol{P}'_{\boldsymbol{x}_k}\boldsymbol{P}'_{\boldsymbol{z}_k}|$ is the distance between the projection points of $\boldsymbol{P}_{\boldsymbol{x}_k}$ and $\boldsymbol{P}_{\boldsymbol{z}_k}$. When $\boldsymbol{P}'_{\boldsymbol{x}_k}\boldsymbol{P}'_{\boldsymbol{z}_k} \perp \boldsymbol{P}'_{\boldsymbol{z}_k}\boldsymbol{O}$, $|\boldsymbol{P}'_{\boldsymbol{x}_k}\boldsymbol{P}'_{\boldsymbol{z}_k}|$ has the minimum value $r_k sin\alpha_k sin\theta'$. Since $r_k sin\alpha_k sin\theta' \leq r_k sin\theta'$, the lower bound of $|\boldsymbol{P}_{\boldsymbol{x}_k}\boldsymbol{P}_{\boldsymbol{z}_k}|$ is $r_k sin\alpha_k sin\theta'$.

Appendix B
Test with Lidar Input

This appendix presents a qualitative test result with point cloud data from a simulated Velodyne HDL-32E Lidar. This Lidar is an omnidirectional and 32-channel Lidar with a horizontal resolution of 0.16° and a vertical resolution of 1.33°. The FOV is $360^\circ \times 40^\circ$. The point cloud data is collected in the pedestrian street world shown in Fig. 8 (d). The mapping parameters are $\theta = 3^\circ$, $Res = 0.15m$, and $L_{max} = 1.6 \times 10^6$. The map size is (10 m, 10 m, 6 m). Fig.17 shows several snapshots of the mapping result.

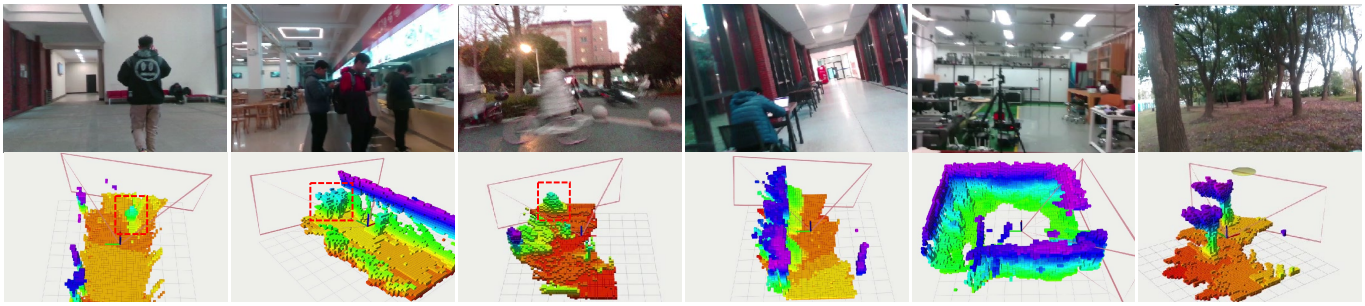
When the voxel size for discrete occupancy status estimation is 0.1 m, 0.2 m, and 0.3 m, the time consumption for mapping is 194.9 ms, 259.4 ms, and 470.4 ms, respectively. In Section V-B, we have discussed that the complexity in the update procedure is $O(\frac{\theta^2}{\theta_h \theta_v} L_k M_k)$. The omnidirectional character of the used Lidar increases $\theta_h \theta_v$ but also increases the particle number M_k in $\mathbb{M}^f$. The result is that the time consumption is about two times the time consumption of using a depth camera with a smaller FOV ($90^\circ \times 60^\circ$). This time consumption is not small enough for real-time usage. Further improvements in computational efficiency will be conducted to realize real-time mapping with this kind of Lidar.

Appendix C
Unknown Area Representation

Representing the unknown area is very useful in exploration tasks. For static grid maps, the grids are initialized with a tag "unknown", and the tag is removed when a ray generated from point cloud measurement passes through or hits the grid. In the DSP map, the unknown area can be represented by the update time of the particles. Adding time stamps on the common particles doesn't work because the particles are born only in the area with obstacles, and thus the unknown area and the free area cannot be distinguished. Therefore, when a new area appears in the map, a small number of static particles, named time particles, which have a zero weight and a time stamp, can be uniformly added to the map. When the measurement point cloud comes, the time particles only update their timestamp to the current time. Then the unknown property of each area can be evaluated by checking the time stamp.

Appendix D
Data Structure Details

Algorithm 1 shows the data structure to store particles in voxel subspaces and basic operations used in the mapping algorithms. ptc_voxel_array is a fixed-size array that stores particles in voxels. The maximum particle number that can



(a) Pedestrian (b) Canteen (c) Outdoor cyclist (d) Corridor with desks (e) Lab (f) Woods

Fig. 14. Snapshots of building the DSP-Dynamic map in different scenarios. The first row shows the RGB image in current FOV. The second row presents the voxelized map view with a resolution of 0.15 m. The pink outlines show the FOV. Red dashed boxes indicate the dynamic obstacles in current FOV.

Fig. 15. The testing scenarios for obstacle avoidance. (a) and (b) are dynamic environments. (c) is a static environment. Red rectangles outline the quadrotor.

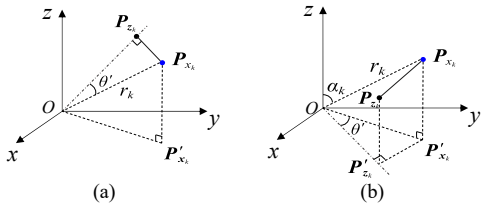


Fig. 16. Calculation of the lower bound distance from a point object position $\boldsymbol{P}_{\boldsymbol{x}_k}$ to a measurement point the position $\boldsymbol{P}_{\boldsymbol{z}_k}$. The azimuth angle and zenith angle difference between $\boldsymbol{P}_{\boldsymbol{z}_k}$ and $\boldsymbol{P}_{\boldsymbol{x}_k}$ is no less than θ'. In subplot (a),the zenith angle difference is θ'. In subplot (a), the azimuth angle difference is θ'. $\boldsymbol{O}$ is the origin point. α_k is the zenith angle of $\boldsymbol{P}_{\boldsymbol{x}_k}$. r_k is the distance from $\boldsymbol{O}$ to $\boldsymbol{P}_{\boldsymbol{x}_k}$. $\boldsymbol{P}'_{\boldsymbol{x}_k}$ and $\boldsymbol{P}'_{\boldsymbol{z}_k}$ are the projection point of $\boldsymbol{P}_{\boldsymbol{x}_k}$ and $\boldsymbol{P}_{\boldsymbol{z}_k}$ in the $x-y$ plane, respectively.

be stored in a voxel subspace is $\frac{\eta_1 L_{max}}{N_v}$, where $\eta_1 > 1$ is an empirical factor used to allocate large storage space. If no storage space is left in the array, the particle to be added is omitted. Note $\frac{\eta_1 L_{max}}{N_v}$ is larger than $L^{\mathbb{V}}_{max}$ used in the resampling step because, in the prediction and particle birth steps, more than $L^{\mathbb{V}}_{max}$ particles may enter one voxel subspace and should all be stored. In the resampling step, the number of particles is reduced to $L^{\mathbb{V}}_{max}$ with Eq. (40).

Algorithm 2 illustrates how the indexes of particles are stored in the pyramid subspaces. $vr_ptc_pyd_array$ is the array to store particle indexes in pyramid subspaces. $\eta_2 > 1$ is another empirical factor that works similar to η_1. In practice, $\eta_1 = \eta_2 = 3$ is adopted. Note that the pyramid subspaces are divided dynamically with the sensor's orientation, and thus, all the particle indexes in $vr_ptc_pyd_array$ must be updated in real-time. Therefore, $vr_ptc_pyd_array$ is emptied after each mapping process and recalculated in the prediction step (Algorithm 5).

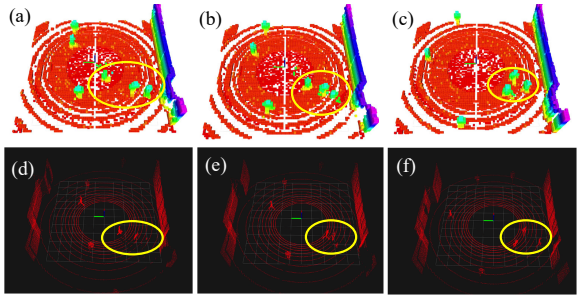


Fig. 17. Snapshots of mapping with the point cloud from Lidar. The top row shows the DSP map, and the bottom row shows the point cloud data. The yellow ellipses outline three pedestrians. In (d) and (e), the two pedestrians on the right are occluded or partially occluded but are clearly shown in the map. In (f), the pedestrians are detected by the Lidar again.

APPENDIX E
ALGORITHMS FOR MAPPING

The algorithms to realize DSP map building are illustrated in Algorithm 3 to 8. In practice, we use C++ for programming. Algorithms 5 to 8 correspond to the prediction, update, particle birth and resampling steps in Section V, respectively. These algorithms show one way to implement the mapping methods. The computational complexities of Algorithm 5, 6, 7 and 8 are $O(L_{max})$, $O(M_k L_{max}\theta^2)$, $O(L_{max})$, and $O(M_k)$, respectively. The overall computational complexity of the map is then $O(M_k L_{max}\theta^2)$. Note the lookup operation in Function $getVacancyIdx$ is disregarded in the computational complexity analysis to simplify the expression. The lookup operation is confined to a small subspace and costs little computing resource.

REFERENCES

[1] Danescu, R., Oniga, F., Nedevschi, and S., "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1331–1342, 2011.

[2] D. Nuss, S. Reuter, M. Thom, T. Yuan, G. Krehl, M. Maile, A. Gern, and K. Dietmayer, "A random finite set approach for dynamic occupancy grid maps with real-time application," *Intl. J. Robot. Research (IJRR)*, vol. 37, no. 8, pp. 841–866, 2017.

[3] G. Tanzmeister and D. Wollherr, "Evidential grid-based tracking and mapping," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1454–1467, 2017.

Algorithm 1: Particle Storage and Operation in Voxel Subspaces

Struct Particle $Particle = \{flag, weight, v_x, v_y, v_z, p_x, p_y, p_z\}$ ▷ Struct of the states of a particle. $flag$ is initialized with 0;
$L_s^{\mathbb{V}} \leftarrow \eta_1 \cdot L_{max}/N_v$ ▷ Max particle number stored in a voxel. $\eta_1 > 1$ is an empirical factor. $N_v = \frac{l_x l_y l_z}{l^3}$. $L_{max}/N_v = L_{max}^{\mathbb{V}}$;
$ptc_voxel_array[N_v][L_s^{\mathbb{V}}]\{Particle\}$ ▷ This array stores the particles in voxel subspaces. N_v is the number of voxel subspaces;
Function addParticleToVoxel($Particle$)
  $idx_voxel \leftarrow$getIdxVoxel($Particle.p_x, Particle.p_y, Particle.p_z$) ▷ Calculate the voxel index of a particle with the particle's position;
  $i \leftarrow$ getVacancyIdx($ptc_voxel_array[idx_voxel]$) ▷ Traverse $ptc_voxel_array[idx_voxel]$ and check the $flag$ until a vacancy with $flag = 0$ is found;
  $Particle.flag \leftarrow 1$, $ptc_voxel_array[idx_voxel][i] \leftarrow Particle$ ▷ Add $Particle$ to the vacancy in ptc_voxel_array;
  return $idx_voxel,\ i$;
Function deleteParticleInVoxel(idx_voxel, idx_ptc)
  $ptc_voxel_array[idx_voxel][idx_ptc].flag \leftarrow 0$ ▷ Delete a particle by setting its flag to zero;
Function moveParticleToNewVoxel(idx_voxel, idx_ptc)
  deleteParticleInVoxel(idx_voxel, idx_ptc) ▷ Delete the particle in the original voxel;
  return addParticleToVoxel($ptc_voxel_array[idx_voxel][idx_ptc]$) ▷ Add the particle to a new voxel and return indexes;

Algorithm 2: Particle Index Storage and Operation in Pyramid Subspaces

Struct Virtual_Particle $Vr_Particle = \{flag, idx_voxel, idx_ptc\}$ ▷ Struct of a virtual particle, which contains indexes mapping to the particle in $ptc_voxel_array[idx_voxel][idx_ptc]$. $flag$ is initialized with 0;
$L_s^A \leftarrow \eta_2 \cdot \frac{L_{max} \cdot \theta^2}{360 \times 180}$ ▷ Max particle number in a pyramid. $\eta_2 > 1$ is an empirical factor;
$vr_ptc_pyd_array[N_f][L_s^A]\ \{Vr_Particle\}$ ▷ This array stores the Virtual_Particles in each pyramid;
$pyd_neighbors_array[N_f][n^2 - 1]$ ▷ This array stores the neighbor pyramids' indexes in the activation range. n is defined in Section V-B;
Function getIdxPyd(p_x, p_y, p_z) ▷ This function calculates the pyramid index of a particle or a point.
  $\{p_{x,s}, p_{y,s}, p_{z,s}\} \leftarrow$ worldToSensorFrame(p_x, p_y, p_z) ▷ Calculate the position of a particle in sensor frame;
  if pointInFov($p_{x,s}, p_{y,s}, p_{z,s}$) **then**
    $idx_pyd \leftarrow$ getIdxPydSensorFrame($p_{x,s}, p_{y,s}, p_{z,s}$) ▷ Calculate the pyramid index of a point using sensor frame position;
    return idx_pyd;
  else return -1;
Function addParticleIdxToPyd($idx_pyd, idx_voxel, idx_ptc$) ▷ This function adds the index of a particle (a virtual particle) to $vr_ptc_pyd_array$.
  $j \leftarrow$ getVacancyIdx($vr_ptc_pyd_array[idx_pyd]$) ▷ Traverse $vr_ptc_pyd_array[idx_pyd]$ until a vacancy ($flag = 0$) is found;
  $vr_ptc_pyd_array[idx_pyd][j] \leftarrow \{1, idx_voxel, idx_ptc\}$ ▷ Add the particle index to the vacancy;

Algorithm 3: Point Cloud Preprocess

$ptcl_array[M_k]\{p_x, p_y, p_z\}$ ▷ The input array that stores the point cloud filtered by a voxel filter with resolution Res;
$M_s^A \leftarrow \frac{\theta^2}{\theta_{snsr}^2}$ ▷ Calculates the maximum measurement point stored in a pyramid. θ_{snsr} is the angle resolution of the sensor;
$ptcl_pyd_array[N_f][M_s^A]\{p_x, p_y, p_z\}$ ▷ This array stores the point cloud in each pyramid in the map frame;
$pyd_pt_num_array[N_f]$, $pyd_length_array[N_f]$ ▷ The arrary of points number and the visible length of each pyramid;
for $i = 1 : M_k$ **do**
  $pt \leftarrow ptcl_array[i]$, $idx_pyd \leftarrow$ getIdxPyd($pt.p_x, pt.p_y, pt.p_z$) ▷ Get the pyramid subspace index of a point;
  rotateAndStore($ptcl_pyd_array, pt, sensor_ort$) ▷ Rotate pt to the map frame with sensors' orientation and store pt in $ptcl_pyd_array$;
  $pyd_pt_num_array[idx_pyd] + +$ ▷ Update the number of measurement points in a pyramid;
  $eu_dist \leftarrow$ squareEuclideanDist(pt) ▷ Calculate square Euclidean distance from pt to the map center;
  if $eu_dist > pyd_length_array[idx_pyd]$ **then**
    $pyd_length_array[idx_pyd] \leftarrow eu_dist$ ▷ Update the visible length of a pyramid in the FOV;

Algorithm 4: Particle Initial Velocity Estimation

$ptcl_vel_array[N_f][M_s^A]\{p_x, p_y, p_z\}$ ▷ This array stores the estimated velocities of points in the point cloud;
$ptcl_vel_array =$ calPointVelocity($ptcl_array$) ▷ Calculate $ptcl_vel_array$ with procedures in Fig. 4 (b);

[4] A. Vatavu, M. Rahm, S. Govindachar, G. Krehl, A. Mantha, S. R. Bhavsar, M. R. Schier, J. Peukert, and M. Maile, "From particles to self-localizing tracklets: A multilayer particle filter-based estimation for dynamic grid maps," *IEEE Intelligent Transportation Systems Magazine*, vol. 12, no. 4, pp. 149–168, 2020.

[5] M. Youngjae, K. Do-Un, and C. Han-Lim, "Kernel-based 3-d dynamic occupancy mapping with particle tracking," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2021, pp. 5268–5274.

[6] K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference," *IEEE Trans. Robot.*, vol. 35, no. 4, pp. 953–966, 2019.

[7] V. Usenko, L. V. Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for mavs using uniform b-splines and 3d circular buffer," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst. (IROS)*. IEEE, 2017, pp. 215–222.

[8] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, vol. 2, 1985, pp. 116–121.

[9] A. Hornung, M. W. Kai, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[10] D. Duberg and P. Jensfelt, "Ufomap: An efficient probabilistic 3d mapping framework that embraces the unknown," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6411–6418, 2020.

[11] P. Gohl, D. Honegger, S. Omari, M. Achtelik, and R. Siegwart, "Omnidirectional visual obstacle detection using embedded fpga," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst. (IROS)*. IEEE, 2015, pp. 3938–3943.

[12] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *Intl. J. Robot. Research (IJRR)*, vol. 31, no. 1, pp. 42–62, 2011.

[13] F. Ramos and L. Ott, "Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent," *Intl. J. Robot. Research (IJRR)*, vol. 35, no. 14, pp. 1717–1730, 2015.

[14] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte,

Algorithm 5: Prediction Step

Empty $vr_ptc_pyd_array[N_f][L_s^A]\{Vr_Particle\}$ ▷ Empty $vr_ptc_pyd_array$ and recalculate later to realize dynamical division of the pyramid subspaces;
$pyd_vr_ptc_num_array[N_f]$ ▷ The array to store the number of particle indexes in a pyramid;
for i in Range(N_v) **do**
  for j in Range($L_s^{\mathbb{V}}$) **do**
    if $ptc_voxel_array[i][j].flag$ *is 0* **then**
      continue ▷ Ignore the array element that doesn't contain a particle currently;
    $particle \leftarrow$predictParticleState($ptc_voxel_array[i][j]$) ▷ Predict the particle's state with the mixture model in Eq. (22) and (46).
    $idx_voxel, idx_ptc \leftarrow$moveParticleToNewVoxel($i, j$) ▷ Move the particle with the predicted state;
    $idx_pyd \leftarrow$getIdxPyd($particle.x, particle.y, particle.z$) ▷ Get the pyramid index of the particle after prediction;
    if $idx_pyd \geq 0$ **then**
      addParticleIdxToPyd($idx_pyd, idx_voxel, idx_ptc$) ▷ Add the particle index to $vr_ptc_pyd_array$ for update;
      $pyd_vr_ptc_num_array[idx_pyd] + +$ ▷ Update the number of particle indexes in the pyramid;

Algorithm 6: Update Step

$\mathrm{C}'_k_array[N_f][M_s^A]$ ▷ This array stores variable $\mathrm{C}'_k(\boldsymbol{z}_k)$ for each $\boldsymbol{z}_k$ and is defined in Eq. (37);
for $i = 1 : N_f$ **do**
  for $j = 1 : pyd_pt_num_array[i]$ **do**
    $pt \leftarrow ptcl_pyd_array[i][j]$ ▷ Get the point in the point clould one by one;
    $\mathrm{C}'_k_array[i][j] \leftarrow$calculateCk($pt$,$vr_ptc_pyd_array$, $pyd_neighbors_array$) ▷ Calculate C'_k with Eq. (37);
for $i = 1 : N_f$ **do**
  for $j = 1 : pyd_vr_ptc_num_array[i]$ **do**
    $ptc \leftarrow ptc_voxel_array\,[vr_ptc_pyd_array[i][j].id_voxel]\,[vr_ptc_pyd_array[i][j].idx_ptc]$ ▷ Get the particle to update;
    $ptc_dist \leftarrow squareEuclideanDist(ptc)$ ▷ Calculate square Euclidean distance from ptc to the map center;
    if $pyd_length_array[i] > 0$ and $ptc_dist \leq pyd_length_array[i]$ **then**
      if $ptc.flag$ is 1 **then**
        $ptc.weight \leftarrow$updateWgt($ptc, \mathrm{C}'_k_array, ptcl_pyd_array$) ▷ Update particle weight with (35);
      else
        $ptc.weight \leftarrow$updateWgtNewBorn($ptc, \mathrm{C}'_k_array, ptcl_pyd_array$) ▷ Update newborn particle weight with (36);

Algorithm 7: Resampling, Occupancy Estimation and Mixture Model Coefficients Calculation

$occ_voxel_array[N_v]$ ▷ This array stores the occupancy probability of each voxel subspace;
$dst_lambda_array[N_v][2]$ ▷ This array stores the coefficients λ_1 and λ_2 of each voxel subspace in DST (Section VI-B);
for $i = 1 : N_v$ **do**
  $dst_lambda_array[i] \leftarrow$ calculateDSTCoefficients($ptc_voxel_array[i]$) ▷ Calculate the DST coefficients with (45);
  $occ_voxel_array[i] \leftarrow$calculateOccPr($weight_voxel_array[i]$) ▷ Calculate a voxel's occupancy probability with (48);
  $ptc_voxel_array[i] \leftarrow$ resample($ptc_voxel_array[i]$) ▷ Resample the particles in a voxel using rejection sampling [36];

Algorithm 8: Particle Birth

for $i = 1 : N_f$ **do**
  for $j = 1 : pyd_pt_num_array[i]$ **do**
    $pt \leftarrow ptcl_pyd_array[i][j]$, $pt_vel \leftarrow ptcl_vel_array[i][j]$ ▷ Traverse every measurement point;
    for $k = 1 : L_b$ **do**
      $pct \leftarrow$ addNoise(pt), ▷ Add noise to pt according to Section VI-B to generate a new particle;
      addParticleToVoxel(pct), ▷ Store the particle in ptc_voxel_array;

"Simultaneous localization, mapping and moving object tracking," *Intl. J. Robot. Research (IJRR)*, vol. 26, no. 9, pp. 889–916, 2007.

[15] G. Chen, W. Dong, X. Sheng, X. Zhu, and H. Ding, "An active sense and avoid system for flying robots in dynamic environments," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 2, pp. 668–678, 2021.

[16] J. Lin, H. Zhu, and J. Alonso-Mora, "Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2020, pp. 2682–2688.

[17] A. Unnikrishnan, J. Wilson, L. Gan, A. Capodieci, P. Jayakumar, K. Barton, and M. Ghaffari, "Dynamic semantic occupancy mapping using 3d scene flow and closed-form bayesian inference," *IEEE Access*, vol. 10, pp. 97 954–97 970, 2022.

[18] C.-C. Wang and C. Thorpe, "Simultaneous localization and mapping with detection and tracking of moving objects," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, vol. 3, 2002, pp. 2918–2924.

[19] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun, "Map building with mobile robots in dynamic environments," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, vol. 2, 2003, pp. 1557–1563.

[20] J. Saarinen, H. Andreasson, and A. J. Lilienthal, "Independent markov chain occupancy grid maps for representation of dynamic environment," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst. (IROS)*, 2012, pp. 3489–3495.

[21] J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, "3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments," *Intl. J. Robot. Research (IJRR)*, vol. 32, no. 14, pp. 1627–1644, 2013.

[22] S. T. O'Callaghan and F. T. Ramos, *Gaussian Process Occupancy Maps for Dynamic Environments*. Cham: Springer International Publishing, 2016, pp. 791–805.

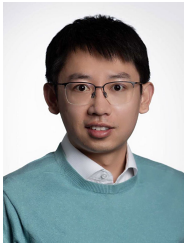
[23] R. Senanayake, S. O'Callaghan, and F. Ramos, "Learning highly dynamic environments with stochastic variational inference," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2017, pp. 2532–2539.

[24] V. Guizilini, R. Senanayake, and F. Ramos, "Dynamic hilbert maps: Real-time occupancy predictions in changing environments," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2019, pp. 4091–4097.

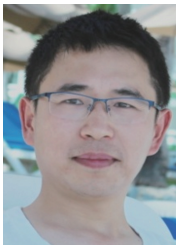
[25] M. Schreiber, V. Belagiannis, C. Gläser, and K. Dietmayer, "Motion estimation in occupancy grid maps in stationary settings using recurrent neural networks," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2020, pp. 8587–8593.

[26] M. Schreiber, V. Belagiannis, C. Glser, and K. Dietmayer, "Dynamic occupancy grid mapping with recurrent neural networks," in *Proc. of*

the IEEE Intl. Conf. on Robot. and Autom. (ICRA), 2021, pp. 6717–6724.
[27] M. Toyungyernsub, E. Yel, J. Li, and M. J. Kochenderfer, "Dynamics-aware spatiotemporal occupancy prediction in urban environments," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 836–10 841.
[28] H. Thomas, M. G. de Saint Aurin, J. Zhang, and T. D. Barfoot, "Learning spatiotemporal occupancy grid maps for lifelong navigation in dynamic scenes," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 484–490.
[29] R. Danescu and S. Nedevschi, "A particle-based solution for modeling and tracking dynamic digital elevation maps," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 3, pp. 1002–1015, 2014.
[30] H. Fan, T. P. Kucner, M. Magnusson, T. Li, and A. J. Lilienthal, "A dual phd filter for effective occupancy filtering in a highly dynamic environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 2977–2993, 2018.
[31] S. Steyer, G. Tanzmeister, and D. Wollherr, "Grid-based environment estimation using evidential mapping and particle tracking," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 3, pp. 384–396, 2018.
[32] R. P. S. Mahler, "Multitarget bayes filtering via first-order multitarget moments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1152–1178, 2003.
[33] B. Ristic, *Particle Filters for Random Set Models*. Springer Publishing Company, Incorporated, 2013.
[34] B. N. Vo, S. Singh, and A. Doucet, "Sequential monte carlo methods for multi-target filtering with random finite sets," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 41, no. 4, pp. 1224–1245, 2005.
[35] B. Ristic, D. Clark, and B. N. Vo, "Improved smc implementation of the phd filter," in *International Conference on Information Fusion*, 2010, pp. 1–8.
[36] G. Casella, C. P. Robert, and M. T. Wells, "Generalized accept-reject sampling schemes," *Lecture Notes-Monograph Series*, pp. 342–347, 2004.
[37] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.
[38] G. Chen, P. Peng, P. Zhang, and W. Dong, "Risk-aware trajectory sampling for quadrotor obstacle avoidance in dynamic environments," *arXiv preprint arXiv:2201.06645*, 2022.



Gang Chen received the B.E. degree and Ph.D. degree in mechanical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2016 and 2022, respectively. He is currently a Postdoc researcher at the Cognitive Robotics Department, Delft University of Technology, the Netherlands. His research interest is in perception and perception-aware planning for obstacle avoidance.



Wei Dong received the B.S. degree and Ph.D. degree in mechanical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2009 and 2015, respectively. He is currently an associate professor in the Robotic Institute, School of Mechanical Engineering, Shanghai Jiao Tong University. For years, his research group was champions in several national-wide autonomous navigation competitions of unmanned aerial vehicles in China. In 2022, he was selected into the Shanghai Rising-Star Program for distinguished young scientists. His research interests include cooperation, perception and agile control of unmanned systems.

Peng Peng received the B.E. degree in mechanical engineering from Harbin Institute of Technology, Harbin, China, in 2020. He is now a graduate student at the State Key Laboratory of Mechanical System and Vibration, School of Mechanical Engineering, Shanghai Jiao Tong University. His research interest is on navigation and motion planning of intelligent robots.

Javier Alonso-Mora is an Associate Professor at the Delft University of Technology, where he leads the Autonomous Multi-robots Lab. He received his Ph.D. degree in robotics from ETH Zurich, in partnership with Disney Research Zurich, and he was a Postdoctoral Associate at the Computer Science and Artificial Intelligence Lab (CSAIL) of the Massachusetts Institute of Technology. His research focuses on navigation, motion planning, and control of autonomous mobile robots, with a special emphasis on multi-robot systems, mobile manipulation, ondemand transportation, and robots that interact with other robots and humans in dynamic and uncertain environments. He currently serves as associate editor for IEEE Transactions on Robotics and for Springer Autonomous Robots. He is the recipient of a talent scheme VENI award from the Netherlands Organization for Scientific Research (2017), the ICRA Best Paper Award on Multi-robot Systems (2019) and an ERC Starting Grant (2021).

Xiangyang Zhu received the B.S. degree from the Department of Automatic Control Engineering, Nanjing Institute of Technology, Nanjing, China, in 1985, the M.Phil. degree in instrumentation engineering and the Ph.D. degree in control engineering, both from Southeast Engineering, in 1989 and 1992, respectively. He is currently a chair professor of mechatronics and the director of Meta Robotics Institute at Shanghai Jiao Tong University, Shanghai, China. His research interests cover a wide range of topics in robotics and the cross-disciplinary technology of biomechatronics, including robotic manipulation planning, neuroprosthetics and neuro-interfacing, human-machine hybrid systems, and soft robotics.

Dr. Zhu has received a number of awards including the National Science Fund for Distinguished Young Scholars from NSFC in 2005, and the Cheung Kong Distinguished Professorship from the Ministry of Education in 2007. He is serving on the editorial board of IEEE Transactions on Cybernetics and Journal of Bionic Engineering.