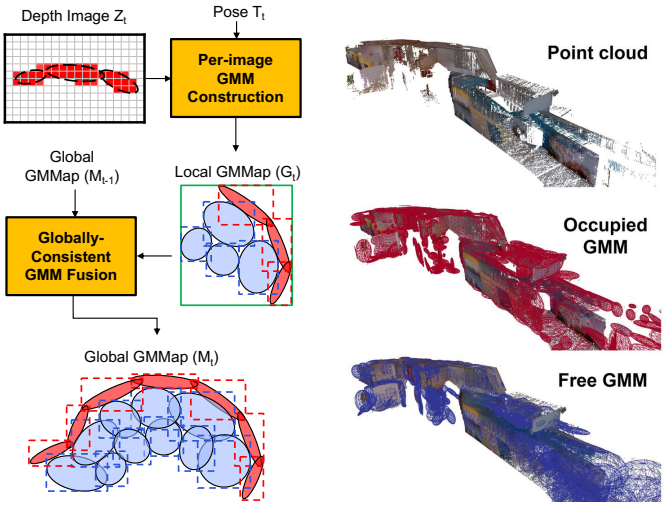
GMMap: Memory-Efficient Continuous Occupancy Map Using Gaussian Mixture Model

Peter Zhi Xuan Li, *Student Member, IEEE*, Sertac Karaman, *Member, IEEE*, Vivienne Sze, *Senior Member, IEEE*,

***Abstract*—Energy consumption of memory accesses dominates the compute energy in energy-constrained robots which require a compact 3D map of the environment to achieve autonomy. Recent mapping frameworks only focused on reducing the map size while incurring significant memory usage during map construction due to the multi-pass processing of each depth image. In this work, we present a memory-efficient continuous occupancy map, named GMMap, that accurately models the 3D environment using a Gaussian Mixture Model (GMM). Memory-efficient GMMap construction is enabled by the single-pass compression of depth images into local GMMs which are directly fused together into a globally-consistent map. By extending Gaussian Mixture Regression to model unexplored regions, occupancy probability is directly computed from Gaussians. Using a low-power ARM Cortex A57 CPU, GMMap can be constructed in real-time at up to 60 images per second. Compared with prior works, GMMap maintains high accuracy while reducing the map size by at least 56%, memory overhead by at least 88%, DRAM access by at least 78%, and energy consumption by at least 69%. Thus, GMMap enables real-time 3D mapping on energy-constrained robots.**

***Index Terms*—Mapping, RGB-D Perception, Sensor Fusion, Memory Efficiency**



(a) Incremental construction of the GMMap from a depth image Z_t and pose T_t obtained at time t. Each depth image is compressed into a local GMMap $\mathcal{G}_t$ which is then fused with the global GMMap $\mathcal{M}_{t-1}$.

(b) Visualization of the first floor of the MIT Stata Center and its GMMap representation consisting of GMMs representing occupied (red) and free (blue) regions. Each Gaussian is visualized as an ellipsoid in 3D.

Fig. 1. Illustration of GMMap's (a) memory-efficient construction procedure and (b) representation for the MIT's Stata Center. Even though Gaussians are continuous and unbounded, GMMs representing occupied and free regions in the environment are represented by red and blue ellipsoids, respectively, created at a Mahalanobis distance of two for ease of visualization. The leaf nodes of the R-tree that store the GMMs are illustrated by dotted rectangles. For the Stata Center, the GMMap models a continuous distribution of occupancy while requiring only 296KB to store.

I. Introduction

ENERGY-constrained microrobots could enable a wide variety of applications, from autonomous navigation, search and rescue, and space exploration [1]. Due to the limited battery capacity onboard these robots, the amount of energy available for actuation (*i.e.*, mechanical systems) and computation (*i.e.*, executing algorithms) is extremely limited. For actuation, researchers of these robots showcased mechanical systems that consume very low power (*i.e.*, under 100 mW) [2]–[6]. Thus, a key remaining factor for enabling autonomy is the lack of energy-efficient algorithms.

During the execution of algorithms, the energy consumption of memory operations (*e.g.*, reading and writing data stored in cache and DRAM) could dominate the total compute energy. For instance, the energy required for accessing on-chip memory (*e.g.*, cache) is more than an order-of-magnitude higher than that when performing a 32-bit multiplication [7]. The energy consumption of memory access increases with the size and distance of the memory from the processor. Within the same chip, accessing a higher-level L2 cache (a few MBs) requires up to an order-of-magnitude more energy than lower-level L0 and L1 caches (a few KBs). However, accessing data stored in a larger, off-chip memory such as DRAM (GBs of storage) requires more than two orders-of-magnitude higher energy than smaller, on-chip (local) CPU caches [7]. The memory (capacity) usage of an algorithm not only consists of output variables but also input and temporary variables allocated during computation. Thus, algorithms designed for many robotics applications, especially the ones involving energy-constrained robots, should be *memory efficient* such that: *i)* the number of *memory accesses* do not dominate; *ii)* amount of *memory (capacity) overhead* for storing input and temporary variables is small enough to remain in lower-level caches.

For mapping algorithms, both memory overhead and accesses could easily dominate. During map construction, the multi-pass processing of sensor measurements requires them to be stored (*i.e.*, as input and temporary variables) entirely in memory to support repeated accesses, which increases overhead and reduces the remaining memory for map storage. Incrementally updating/reconstructing a previously-observed

Authors are with the Massachusetts Institute of Technology, Cambridge, MA 02139, USA. Emails: {peterli, sertac, sze}@mit.edu. This work was partially funded by the NSF RTML 1937501 and the NSF CPS 1837212.

region in the map is typically performed by casting sensor measurement rays into the map. Since these rays diverge away from the sensor origin, memory accesses along these rays often lack spatial and temporal locality required for effective cache usage, and thus require a significant number of memory accesses to DRAM. Thus, achieving memory efficiency is both crucial and challenging for mapping algorithms.

In addition to achieving memory efficiency, the resulting map should satisfy the following requirements to enable memory-efficient, real-time processing of a variety of downstream applications that enable autonomy.

1) **Compactness**: A compact map can represent a larger portion of the environment in both on-chip (cache) and off-chip memory (DRAM). When accessing a region of the environment that does not reside in the cache, a compact map also reduces the number of energy-intensive DRAM accesses required to update the cache.
2) **Modeling unexplored regions**: In autonomous exploration, the robot seeks to minimize the number of unexplored regions while traversing in obstacle-free regions. Thus, the ability to model unexplored regions enables state-of-the-art autonomous exploration algorithms based on frontier [8] or mutual information [9], [10].
3) **Query compute efficiency**: During path planning and autonomous exploration, the robot needs to query multiple locations in the map to determine the current state of the environment [11]. The results of these queries are often used to make decisions, such as the next location to travel, in real time. Thus, the state of the environment should be efficiently computed from the map.

Current state-of-the-art mapping frameworks require the probabilistic modeling of *occupancy* (*i.e.*, whether or not an obstacle exists) at every location in the 3D environment. These frameworks can be classified based on their underlying probabilistic models used to infer occupancy. For instance, the well-known framework, OctoMap [12], contains a set of Bernoulli random variables for modeling the occupancy at a discrete set of homogeneous regions in the environment. Even though OctoMap could model unexplored regions and achieve query efficiency, OctoMap is not compact enough for storage on energy-constrained robots. By using more compact models (*e.g.*, set of Gaussians or kernels), recent frameworks (*e.g.*, NDT-OM [13], Hilbert Map [14], HGMM [15]) mostly focused on reducing the map size while incurring significant memory overhead and accesses for multi-pass processing of raw sensor measurements in nearly every stage of the mapping pipeline. In addition, the resulting maps produced by these frameworks cannot satisfy all of the above-mentioned requirements for enabling efficient downstream applications.

In this paper, we propose a continuous occupancy map comprised of a compact Gaussian Mixture Model (GMM), named GMMap, that is efficiently and accurately constructed from a sequence of depth images and poses of a robot. To achieve significantly higher memory efficiency than prior works, our GMMap accurately compresses each depth image into a compact GMM in a *single pass*, and *directly* operates on Gaussians in the GMM (*i.e.*, without other intermediate representations) for all remaining mapping operations. Our contributions are summarized as follows:

1) **Single-pass compression**: A *single-pass* procedure that accurately compresses a depth image into a local GMM for both free and occupied regions. Prior works [14]–[19] require significant overhead for storing the entire image in memory due to *multi-pass* processing.
2) **Gaussian-direct map construction**: A novel procedure that *directly* fuses the local GMMs across multiple images into a globally-consistent GMM without casting sensor rays (*i.e.*, one ray for each pixel in the depth image) into the map. Prior works [12], [13], [15], [20] require a significant number of memory accesses during ray casting in order to update the previously-observed region that intersects with all sensor rays.
3) **Gaussian-direct occupancy query**: An extension of Gaussian Mixture Regression to *directly* compute occupancy from GMM while accounting for unexplored regions. Prior works require constructing and storing intermediate representations for modeling unexplored regions [14], [17] or do not model them at all [15].

In our previous work [21], we proposed the Single-Pass Gaussian Fitting (SPGF) algorithm that enables single-pass compression of depth image into a GMM representing only the occupied region (*i.e.*, a part of the first contribution) but not the obstacle-free region. In this work, we not only extend our previous work to also construct a GMM representing the free region (*i.e.*, the first contribution) but also illustrate how to directly operate on Gaussians during map construction and occupancy query (*i.e.*, the second and third contribution). An overview of the GMMap and its representation for the first floor of MIT's Stata Center is illustrated in Fig. 1.

This paper is organized as follows. After analyzing existing works in Section II, we describe how the occupancy is compactly represented and efficiently estimated from our GMMap in Section III. Memory-efficient algorithms that incrementally and accurately construct the GMMap given a sequence of depth images are presented in Section IV. Finally, we validate GMMap against existing works in terms of mapping accuracy, memory footprint, throughput, and energy consumption across multiple environments in Section V.

II. Related Work

Constructing an accurate and compact representation of the 3D environment is crucial for enabling many downstream robotics applications such as path planning and autonomous exploration. During the past few decades, many frameworks proposed different models to represent the distribution of the occupancy probability (*i.e.*, the likelihood that a region contains an obstacle) across the 3D environment. These models exhibit different trade-offs in memory and computational efficiency during the construction and querying of the map.

Discrete representations: Some of the most popular mapping frameworks discretize the environment into cubic regions (*i.e.*, grids in 2D and voxels in 3D) such that each region contains a Bernoulli random variable representing the occupancy probability and is assumed to be spatially independent of each

other. One of the earliest 2D mapping frameworks, the occupancy grid map [22], discretizes the environments into equally-sized grids. However, the map size is prohibitively large in 3D because the size scales cubically with the dimensions of the voxels and the environment. To reduce map size in 3D, OctoMap [12] and other voxel-based methods (*e.g.*, [23], [24]) store the occupancy probabilities in voxels whose sizes can adapt to homogeneous regions in the environment. However, OctoMap and other voxel-based methods suffer from artifacts associated with voxelization and require a large amount of memory accesses during construction. To incrementally construct the map given a set of sensor rays (more than 300,000 in each 640×480 depth image), each ray is cast into the map to update the subset of voxels such rays intersect. Since these rays diverge away from the sensor origin, memory accesses along these rays often lack spatial and temporal locality for effective cache usage (especially if the map is too large to fit in caches). Since voxel-based methods are often not compact, updating the map requires a significant amount of memory accesses (more than 300,000 per image) to off-chip DRAM.

Non-parametric representations: To relax the spatial independence assumption in discrete map representations, Gaussian Process (GP) was proposed to estimate a continuous distribution of occupancy [25] using a covariance function that captures the spatial correlation among all sensor measurements. Since GP requires the storage of *all* sensor measurements (since the beginning of the robotics experiment) to update the covariance function, the memory overhead scales with the total number of measurements N. During a map query, the covariance function generates a large matrix that requires $O(N^3)$ to invert, which greatly reduces the query efficiency. To enable faster map construction and query, recent non-parametric methods such as GPOctoMap [26] and BGKOctoMap-L [20] discretize the environment into blocks of octrees (*i.e.*, a test-data octree). For subsets of measurements (*i.e.*, training data) that lie within each block, GPOctoMap and BGKOctoMap-L update the octrees in each block and its neighbors (*i.e.*, extended blocks) using GP and Bayesian Generalized Kernel (BGK) inference, respectively. Similar to OctoMap, both GPOctoMap and BGKOctoMap-L directly operate on sensor rays that are cast into the map during incremental construction and require significant memory accesses to DRAM. In addition, both frameworks require the storage of training data for each block during map construction which incurs significantly larger memory overhead than OctoMap.

Semi-parametric representations: To create an extremely compact representation of the environment, several frameworks compress the sensor measurements using a set of parametric functions (*e.g.*, Gaussians or other kernels) which are then used to infer occupancy. One of the well-known semi-parametric representations is the Normal Distribution Transform Occupancy Map (NDT-OM) [13] that partitions the environment into large voxels such that measurements within each voxel are represented by a Gaussian. Since measurements within a voxel could belong to multiple objects, representing them with a single Gaussian often leads to a loss of accuracy in the resulting map. Due to the ineffective cache usage during the casting all sensor rays (*i.e.*, more than 300,000) from each depth image into the map, NDT-OM also requires significant memory access to DRAM during map construction.

To further reduce map size, recent frameworks, such as Hilbert Map (HM) [14], Fast Bayesian Hilbert Map (Fast-BHM) [27], Variable Resolution GMM (VRGMM) map [17], Hierarchical GMM (HGMM) map [15], compress sensor rays into special kernels (in HM) or Gaussians (in VRGMM and HGMM). Such compression is performed using techniques such as Quick-Means (QM) [14], Hierarchical Expectation-Maximization (H-EM) [16], Region Growing (RG) [18], Self-Organizing GMMs (SOGMM) [19], and Integrated Hierarchical GMMs (IH-GMM) [28]. However, these techniques require significant memory overhead to store all sensor measurements (more than 300,000 pixels in a 640×480 depth image) due to their *multi-pass* processing. Even though the resulting maps are compact after compression, they either could not model unexplored regions (in HGMM), or require online training (for a logistic regression classifier in HM) and intermediate representations to model these regions (using Monte Carlo sampling to create an intermediate grid map in VRGMM). Even though our GMMap is also classified as a semi-parametric representation, we can accurately construct and query the map directly using Gaussians (while preserving unexplored regions) to reduce memory overhead and accesses.

III. Occupancy Representation & Estimation

In this section, we describe how to compactly model a continuous distribution of occupancy using a Gaussian Mixture Model (GMM) in the proposed GMMap. In addition, we illustrate how to directly estimate the occupancy probability from Gaussians using Gaussian Mixture Regression (GMR) while accounting for the initial unknown state of the environment so that the unexplored regions are preserved. Unless stated otherwise, matrices are denoted using regular uppercase letters, random variables are denoted using bold uppercase letters, and vectors/scalars are denoted using regular lowercase letters. In addition, a lower or upper calligraphic letter represents an individual Gaussian (*i.e.*, parameterized by mean and covariance) or a set of Gaussians in a GMM (*i.e.*, parameterized by a set of mean, covariance, and weights), respectively. Depending on the context, we use these calligraphic letters to interchangeably represent either general Gaussian (or GMM) distributions or their instantiations with specific parameters.

Let $\boldsymbol{X} \in \mathbb{R}^3$ denote the 3D coordinate in the world frame. Let $\boldsymbol{O} \in \mathbb{R}$ denote the occupancy value such that regions with values greater than one are occupied with obstacles, and regions with values less than zero are obstacle free. In addition, unexplored regions have an occupancy value near 0.5. Let $\boldsymbol{P}$ denote the joint random variable such that

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{O} \end{bmatrix}. \tag{1}$$

The map $\mathcal{M}$ of the 3D environment is represented by the following GMM which is an *unnormalized* distribution for the joint variable $\boldsymbol{P}$, *i.e.*,

$$\mathcal{M}_{\boldsymbol{P}}(p) \sim \sum_{i=1}^{K} \pi_i \mathcal{N}\left(p \mid \mu_i, \Sigma_i\right), \tag{2}$$

where K is the number of Gaussians, and $\mathcal{N}(\cdot)$ is a Gaussian distribution. The weight π_i, mean μ_i, and covariance Σ_i are the parameters of the ith Gaussian such that

$$\mu_i = \begin{bmatrix} \mu_{i\boldsymbol{X}} \\ \mu_{i\boldsymbol{O}} \end{bmatrix}, \ \Sigma_i = \begin{bmatrix} \Sigma_{i\boldsymbol{X}} & \Sigma_{i\boldsymbol{XO}} \\ \Sigma_{i\boldsymbol{OX}} & \Sigma_{i\boldsymbol{O}} \end{bmatrix}. \tag{3}$$

Note that the GMM in Eqn. (2) can be compactly stored because each Gaussian is parameterized by μ_i, Σ_i, and π_i. For the rest of the paper, we drop the index i for all variables when we refer to any Gaussian in the GMM.

During the experiment, the robot makes a sequence of range measurements. Each range measurement consists of a ray that originates from the robot, passes through a free region, and ends at the surface of an obstacle (occupied region). Regions that are traversed by all measurement rays are *observed* by the robot. We determine the parameters of the GMM in Eqn. (2) using range measurements such that it compactly models all observed regions. Thus, regions that have not been observed (*i.e.*, unexplored) cannot be modeled by the GMM alone.

To compactly model the unexplored region, we use the *unexplored prior* $\mathcal{Q}_{\boldsymbol{O}|\boldsymbol{X}}$ with its weight π_0 to represent the initial unknown state of the entire environment, *i.e.*,

$$\mathcal{Q}_{\boldsymbol{O}|\boldsymbol{X}}(o \mid x) = \mathcal{N}(o \mid \mu_0, \sigma_0^2), \tag{4}$$

where

$$\mu_0 = 0.5, \ \ \sigma_0^2 = 0.25. \tag{5}$$

The weight π_0 should be set to a large value such that measurements from multiple timesteps are required to shift the occupancy value of an unexplored region (*i.e.*, 0.5) towards zero (free region) or one (occupied region) during GMR.

Unlike prior semi-parametric representations that estimate occupancy probability using either a classifier that requires additional online training [14], [29] or intermediate representations that require additional memory overhead [17], we efficiently preserve these regions by incorporating the unexplored prior into the Gaussian Mixture Regression (GMR) [30]. We describe the GMR procedure used to estimate occupancy directly from Gaussians as follows.

Using the GMM in Eqn. (2) and the unexplored prior in Eqn. (4), the occupancy $\boldsymbol{O}$ conditioned on the query location $\boldsymbol{X} = x$ is computed as

$$\mathbb{P}_{\boldsymbol{O}|\boldsymbol{X}}(o \mid x) = \sum_{i=0}^{K} \omega_i(x) \mathcal{N}\left(o \mid m_i(x), \sigma_i^2(x)\right), \tag{6}$$

where

$$\omega_i(x) = \begin{cases} \frac{\pi_0}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_{j\boldsymbol{X}}, \Sigma_{j\boldsymbol{X}}) + \pi_0}, & \text{if } i = 0, \\ \frac{\pi_i \mathcal{N}(x|\mu_{i\boldsymbol{X}}, \Sigma_{i\boldsymbol{X}})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_{j\boldsymbol{X}}, \Sigma_{j\boldsymbol{X}}) + \pi_0}, & \text{otherwise,} \end{cases} \tag{7}$$

$$m_i(x) = \begin{cases} \mu_0, & \text{if } i = 0, \\ \mu_{i\boldsymbol{O}} + \Sigma_{i\boldsymbol{OX}} \Sigma_{i\boldsymbol{X}}^{-1} (x - \mu_{i\boldsymbol{X}}), & \text{otherwise,} \end{cases} \tag{8}$$

$$\sigma_i^2(x) = \begin{cases} \sigma_0^2, & \text{if } i = 0, \\ \Sigma_{i\boldsymbol{O}} - \Sigma_{i\boldsymbol{OX}} \Sigma_{i\boldsymbol{X}}^{-1} \Sigma_{i\boldsymbol{XO}}, & \text{otherwise.} \end{cases} \tag{9}$$

The expected occupancy value and its variance at location x is regressed using GMR as

$$m(x) = \mathbb{E}[\boldsymbol{O}|\boldsymbol{X} = x] = \sum_{i=0}^{K} \omega_i(x) m_i(x), \tag{10}$$

$$v(x) = \text{Var}[\boldsymbol{O}|\boldsymbol{X} = x] = \sum_{i=0}^{K} \omega_i(x) \left(m_i(x)^2 + \sigma_i^2(x)\right) - m(x)^2. \tag{11}$$

The occupancy value can transition suddenly across the boundaries separating occupied and free regions (*e.g.*, at surfaces of obstacles). To better capture such transitions, each Gaussian in the GMMap models either an occupied or free region, but not both. Thus, the set of Gaussians representing occupied regions is defined as *occupied Gaussians* with an occupancy value of *one* (*i.e.*, $\mu_{\boldsymbol{O}} = 1$). In addition, the set of Gaussians representing free regions is defined as *free Gaussians* with an occupancy value of *zero* (*i.e.*, $\mu_{\boldsymbol{O}} = 0$).

Representing occupied and free regions separately also guarantees that the expectation $m(x)$ in Eqn. (10) is bounded within $[0, 1]$. Thus, the expectation $m(x)$ becomes the *occupancy probability* of the environment at the query location x. In addition, the covariance terms $\Sigma_{\boldsymbol{XO}}$, $\Sigma_{\boldsymbol{OX}}$ and $\Sigma_{\boldsymbol{O}}$ in Eqn. (3) become zero for all Gaussians, which significantly simplifies the entire GMR procedure and reduces the memory required to store the Gaussians in our map.

Even though occupied and free regions are modeled separately, the occupancy probability predicted by GMR in Eqn. (10) is dependent on the relative contribution of all Gaussians because *each Gaussian has no bound*. In particular, the relative contributions of occupied and free Gaussians in GMR provide a probabilistic way to resolve inconsistencies/noise of the sensor measurements, especially when obstacle and free Gaussians overlap near the obstacle surfaces. Although each Gaussian has no bound, its distribution tapers off from its mean at an exponential rate. Thus, the entire GMR procedure from Eqn. (6) to (11) can be accurately and efficiently approximated using a small subset of Gaussians whose Mahalanobis distances between their means to the query location x are less than a threshold α_M, *i.e.*,

$$\sqrt{(x - \mu_{\boldsymbol{X}})^\top \Sigma_{\boldsymbol{X}}^{-1} (x - \mu_{\boldsymbol{X}})} \leq \alpha_M, \tag{12}$$

where μ_X and Σ_X are parameters of a Gaussian defined in Eqn. (3). In our experiments, we chose $\alpha_M = 2$ to ensure that more than 95% of the Gaussian distribution is considered.

To efficiently obtain the subset of Gaussians that satisfies Eqn. (12) in $O(\log(K))$ time (where K is the total number of Gaussians), we store the GMMap using an R-tree [31] constructed with bounding boxes that are axis-aligned with the world frame. Since the surface that satisfies the equality in Eqn. (12) for each Gaussian can be visualized as an ellipsoid in 3D, the bounding box at the leaf node of the R-tree for each Gaussian is sized to enclose such ellipsoid. In most figures, occupied and free Gaussians are represented by red and blue ellipsoids, respectively. The occupied and free Gaussians in GMMap with their corresponding bounding boxes (dotted rectangles) are illustrated at the bottom of Fig. 1a.

IV. Memory-Efficient Map Construction

In this section, we present a memory-efficient framework to construct the GMMap $\mathcal{M}$ (*i.e.*, Eqn. (2)). At each timestep t, we *incrementally* constructs the GMMap $\mathcal{M}$ by updating the previous GMMap $\mathcal{M}_{t-1}$ with current measurements from the depth image $Z_t \in \mathbb{R}^{U\times V}$ obtained at pose $T_t \in \mathbb{SE}(3)$. As illustrated in Fig. 1a, our framework consists of the following two procedures executed sequentially for each depth image:

1) **Per-image GMM construction:** The depth image Z_t with width U and height V obtained at pose T_t is compressed into a compact local GMMap $\mathcal{G}_t$. A memory-efficient algorithm is proposed in Section IV-B to perform such compression one pixel at a time in a *single pass* through the depth image. Memory overhead is greatly reduced by avoiding the storage of the entire depth image in memory which is required for prior multi-pass approaches [14]–[19].
2) **Globally-consistent GMM fusion:** The local GMMap $\mathcal{G}_t$ is fused into the previous global GMMap $\mathcal{M}_{t-1}$ to obtain the updated GMMap $\mathcal{M}_t$. A memory-efficient algorithm is proposed in Section IV-C to perform such fusion *directly* using Gaussians. The amount of memory accesses is greatly reduced by avoiding casting rays (more than 300,000) from each 640×480 image into the map as seen in prior works [12], [13], [15], [20].

To efficiently update Gaussian parameters during the above-mentioned procedures with little memory overhead, we present preliminaries that illustrate *in-place* construction of Gaussians using the method of moments (MoM) [32] in Section IV-A.

A. Efficiently Updating Gaussian Parameters

We illustrate how to efficiently update the parameters of Gaussians in-place *given* the correspondences between measurements and Gaussians. In the method of moments (MoM) [32], the first and second moments of a Gaussian are intermediate representations of its mean and covariance, respectively. Let $\boldsymbol{P} = [\boldsymbol{X}, \boldsymbol{O}]^\top$ denote the joint variable for the 3D coordinate $\boldsymbol{X}$ (with respect to the current sensor origin) and its occupancy $\boldsymbol{O}$. The unnormalized first $m^{(1)} \in \mathbb{R}^4$ and second $M^{(2)} \in \mathrm{S}^4_+$ moments of each Gaussian are defined as

$$m^{(1)} = \xi\mathbb{E}[\boldsymbol{P}],\ \ M^{(2)} = \xi\mathbb{E}[\boldsymbol{P}^2], \tag{13}$$

where ξ is a normalization constant. Thus, the mean μ and covariance Σ of each Gaussian defined in Eqn. (3) can be recovered *in-place* from the unnormalized moments as

$$\mu = \frac{1}{\xi}m^{(1)},\ \ \Sigma = \frac{1}{\xi}M^{(2)} - \mu\mu^\top. \tag{14}$$

Unnormalized moments can be incrementally updated without relying on past measurements (which do not need to be stored in memory). Thus, during map construction, the moments for each Gaussian are stored instead of its mean and covariance. Recall that from Section III, each measurement (ray) consists of a point (*i.e.*, end of a ray representing the surface of an obstacle) and a line (*i.e.*, from the sensor origin to the end of the ray representing free region). Fusing a *point* $p = [x, 1]^\top \in \boldsymbol{P}$ into an occupied Gaussian is computed as

$$m^{(1)} \leftarrow m^{(1)} + p, \tag{15a}$$
$$M^{(2)} \leftarrow M^{(2)} + pp^\top, \tag{15b}$$
$$\xi \leftarrow \xi + 1. \tag{15c}$$

Fusing a *line* from the sensor origin to the endpoint $p = [x, 0]^\top$ into a free Gaussian is computed as

$$m^{(1)} \leftarrow m^{(1)} + \frac{\|p\|}{2}p, \tag{16a}$$
$$M^{(2)} \leftarrow M^{(2)} + \frac{\|p\|}{3}pp^\top, \tag{16b}$$
$$\xi \leftarrow \xi + \|p\|. \tag{16c}$$

Note that the second term on the right side of Eqn. (16a) and (16b) is the closed-from expression for the first and second moments of the line, respectively. Our closed-form expression is accurate and more computationally efficient than prior works [14], [15] that approximate both moments by using points sampled at a fixed interval along the line.

When regressing occupancy using GMR, the unnormalized weight π of each occupied or free Gaussian should represent the amount of occupied or free evidence in the region where such Gaussian resides. For each free Gaussian, its weight π equals to the total length of all line segments used during construction. When a line from sensor origin to $p = [x, 0]^\top$ is fused into a free Gaussian, its weight π is updated as

$$\pi \leftarrow \pi + \|p\|. \tag{17}$$

To ensure that the occupancy regressed using GMR is meaningful, the weights for occupied Gaussians should have the same unit as those for the free Gaussians. Thus, when a new endpoint p is fused into an occupied Gaussian, its weight is also updated using Eqn. (17).

Lastly, the Gaussian containing the fusion of two occupied or free Gaussians indexed by i and j is computed as

$$m^{(1)} \leftarrow m_i^{(1)} + m_j^{(1)}, \tag{18a}$$
$$M^{(2)} \leftarrow M_i^{(2)} + M_j^{(2)}, \tag{18b}$$
$$\xi \leftarrow \xi_i + \xi_j, \tag{18c}$$
$$\pi \leftarrow \pi_i + \pi_j. \tag{18d}$$

B. Per-Image GMM Construction

As illustrated in Fig. 2, we present a single-pass algorithm that constructs a local GMMap $\mathcal{G}_t$ given the depth image Z_t obtained at pose T_t. From Section III, occupied and free regions in the environment are separately modeled using occupied (visualized using red ellipsoids) and free (visualized using blue ellipsoids) Gaussians, respectively. Thus, our algorithm, described in Alg. 1, creates both types of Gaussians in the local map $\mathcal{G}_t$ by executing the following procedures sequentially:

1) **SPGF*** (Line 2 in Alg. 1) : A memory-efficient algorithm that constructs the occupied GMM $\mathcal{G}_{t,\text{occ}}$ and a compact free GMM basis $\mathcal{F}_{t,\text{free}}$ in a *single pass* through the image Z_t using only the endpoints of the sensor rays.

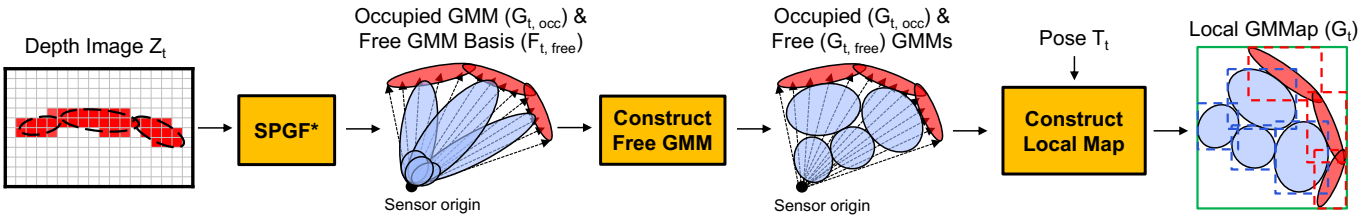


Fig. 2. **Per-image GMM construction**: Constructing a local GMMap $\mathcal{G}_t$ that accurately represents both occupied and free regions from the current depth image Z_t obtained at pose T_t. Rays associated with each pixel in the depth image are illustrated with dotted arrows. Occupied and free GMMs are illustrated with red and blue ellipsoids, respectively. Dotted rectangles in the map $\mathcal{G}_t$ represent the bounding boxes at the leaf nodes of the R-tree. The green rectangle represents the bounding box at the root node of the R-tree that encloses the entire map $\mathcal{G}_t$.

Algorithm 1: Per-Image GMM Construction

Input: Depth image Z_t, pose T_t
Output: Local GMMap $\mathcal{G}_t$

1 **function** constructLocalGMM(Z_t, T_t)
2 $\quad \mathcal{G}_{t,\text{occ}}, \mathcal{F}_{t,\text{free}} \leftarrow$ SPGF*(Z_t)
3 $\quad \mathcal{G}_{t,\text{free}} \leftarrow$ constructFreeGMM($\mathcal{F}_{t,\text{free}}$)
4 $\quad \mathcal{G}_t \leftarrow \mathcal{G}_{t,\text{free}} \cup \mathcal{G}_{t,\text{occ}}$
5 $\quad \mathcal{G}_t \leftarrow$ transform($\mathcal{G}_t, T_t$)
6 $\quad \mathcal{G}_t \leftarrow$ constructRtree($\mathcal{G}_t$)
7 $\quad$ **return** $\mathcal{G}_t$
8 **subfunction** SPGF*(Z_t)
9 $\quad \mathcal{Q} \leftarrow \varnothing, \mathcal{Q}_{\text{prev}} \leftarrow \varnothing$
10 $\quad$ **for** ($v = 0;\ v < V;\ v = v + 1$) {
11 $\quad\quad L_v \leftarrow$ extractScanline(Z_t, v)
12 $\quad\quad \mathcal{S} \leftarrow$ scanlineSegmentation(L_v)
13 $\quad\quad$ **if** $v = 0$ **then**
14 $\quad\quad\quad \mathcal{Q}_{\text{prev}} \leftarrow \mathcal{S}$
15 $\quad\quad$ **else**
16 $\quad\quad\quad \mathcal{Q}_{\text{prev}}, \mathcal{Q}_{\text{comp}} \leftarrow$ segmentFusion($\mathcal{Q}_{\text{prev}}, \mathcal{S}$)
17 $\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{Q}_{\text{comp}}$
18 $\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{Q}_{\text{prev}}$
19 $\quad \mathcal{G}_{t,\text{occ}}, \mathcal{F}_{t,\text{free}} \leftarrow \mathcal{Q}$
20 $\quad$ **return** $\mathcal{G}_{t,\text{occ}}, \mathcal{F}_{t,\text{free}}$

As illustrated in Fig. 3, operations within SPGF* are extended from our prior work SPGF [21].

2) **Construct Free GMM** (Line 3 in Alg. 1): By only considering the endpoints of the sensor rays in SPGF*, Gaussians represented by the free GMM basis $\mathcal{F}_{t,\text{free}}$ cannot represent free region encoded within the camera frustum very well (see Fig. 4a). Thus, the basis $\mathcal{F}_{t,\text{free}}$ is processed to construct the free GMM $\mathcal{G}_{t,\text{free}}$ that better represents the free region (see Fig. 4b).
3) **Construct Local Map** (Lines 4 to 6 in Alg. 1): Occupied $\mathcal{G}_{t,\text{occ}}$ and free $\mathcal{G}_{t,\text{free}}$ GMMs are transformed to the world frame using the pose T_t. Then, these GMMs are inserted into the R-tree to create the local map $\mathcal{G}_t$.

SPGF* (Lines 8 to 20 in Alg. 1): The SPGF* algorithm constructs an occupied GMM $\mathcal{G}_{t,\text{occ}}$ and a compact free GMM basis $\mathcal{F}_{t,\text{free}}$ by processing one scanline (*i.e.*, a row of pixels) at a time in a *single pass* through the entire depth image Z_t, which avoids the storage of the entire depth image in memory required for prior *multi-pass* approaches [14]–[19]. The SPGF* algorithm is a direct extension of our prior work SPGF [21] which exploits the intrinsic properties of the depth camera to accurately infer the correspondences between Gaussians and depth measurements. We briefly summarize SPGF for creating obstacle Gaussians as follows.

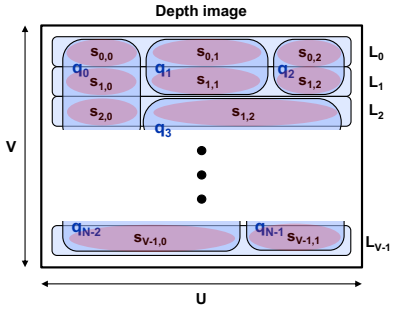
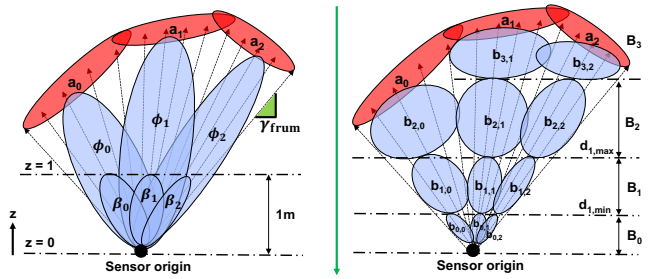


Fig. 3. Single-pass processing of the depth image in SPGF* for constructing the set of Gaussians $\mathcal{Q}$ where each element $q_j \in \mathcal{Q}$ contains one occupied Gaussian a_j and a free Gaussian basis f_j. In Scanline Segmentation, each row (indexed by v) of the depth image is partitioned into a set of line segments $\mathcal{S} = \{s_{v,i}\}$, which are fused across rows to form $q_j \in \mathcal{Q}$ in Segment Fusion.

Recall that each depth image contains a set of *organized* depth measurements such that measurements that are neighbors in the image are likely neighbors on the same planar obstacle surface. In SPGF, we derive distance-based thresholds to accurately determine which subset of neighboring measurements belong to the same planar surface (and Gaussian) across scanlines (*i.e.*, rows) of each image. As illustrated in Fig. 3, each scanline is denoted by L_v, where v is the row index. In Scanline Segmentation (SS, Line 12), pixels from each scanline are partitioned into a set of line segments $\mathcal{S}$ such that each segment $s \in \mathcal{S}$ represents a locally planar surface with distinct orientation. Specifically, neighboring pixels belong to the same line segment if the distances among them fall below an adaptive threshold derived from the orientation of the surface and camera parameters. In Segment Fusion (SF, Line 16), segments are fused across successive scanlines to form a set of completed Gaussians $\mathcal{Q}_{\text{comp}}$ (appended to output in Line 17) and incomplete Gaussians $\mathcal{Q}_{\text{prev}}$ (for fusion with the next scanline). Specifically, segments across the neighbor scanlines belong to the same surface (and Gaussian) if they are sufficiently close and parallel with each other.

To create a free Gaussian basis for each obstacle Gaussian, the implementation of SS and SF in SPGF* is almost identical to those in SPGF except for the following differences. Recall that each pixel in the depth image is a sensor ray that



(a) Visualization of the free Gaussian basis $f_j = (\phi_j, \beta_j)$ associated with each occupied Gaussian a_j. These bases cannot represent the free region faithfully (*e.g.*, near the obstacles).

(b) Visualization of the free Gaussian $b_{i,j}$ that is recovered in subregion $\mathcal{B}_i$ from basis $f_j = (\phi_j, \beta_j)$. The set of all free Gaussians accurately represents the free region.

Fig. 4. Visualization of (a) the free Gaussian basis which can be used to recover (b) a corresponding set of free Gaussians in subregion $\mathcal{B}_i$ whose size increases with the distance from the sensor origin.

originates from the robot. In SPGF, only occupied GMM $\mathcal{G}_{t,\text{occ}}$ is constructed using endpoints of the sensor rays from all depth pixels in the image. In particular, Eqn. (15) and (18) are used to construct each occupied Gaussian (say a_j) in SS and SF, respectively. Since we would like to construct Gaussians associated with the free region as well, SPGF* constructs two free Gaussians ϕ_j (using the entire sensor ray) and β_j (using normalized sensor ray with depth $z = 1$) concurrently with each occupied Gaussian a_j. Those free Gaussians are constructed using Eqn. (16) in SS and Eqn. (18) in SF. Thus, for SPGF*, each element in the set $\mathcal{S}$, $\mathcal{Q}$, $\mathcal{Q}_{\text{prev}}$ and $\mathcal{Q}_{\text{comp}}$ of Alg. 1 includes the occupied Gaussian a_j with its associated free Gaussians ϕ_j and β_j.

Fig. 4a illustrates the free Gaussians ϕ_j and β_j associated with each occupied Gaussian a_j. Note that the free Gaussians ϕ_j and β_j do not represent the free space traversed by the sensor rays very well. However, these free Gaussians can be used to reconstruct a better representation of free space as illustrated in Fig. 4b during the subsequent procedure. Thus, we define each free Gaussian basis f_j such that $f_j = (\phi_j, \beta_j)$. The set of all free Gaussian bases generated at the output of SPGF* forms the free GMM basis $\mathcal{F}_{t,\text{free}}$.

Since the criteria for constructing and updating the Gaussians in SPGF* is identical to SPGF, SPGF* inherits many desirable properties from SPGF. Since SS in Line 12 dominates SPGF* and can be executed independently for each scanline, SPGF* can be parallelized by concurrently executing SS for different scanlines across multiple CPU or GPU cores. Due to single-pass pixel-per-pixel processing in SS, only one pixel is stored in memory at any time. Thus, SPGF* is memory-efficient and avoids the storage of the entire depth image in memory as seen in most prior works.

Construct Free GMM (Alg. 2): In this section, we present Alg. 2 that directly generates the set of free Gaussians $\mathcal{G}_{t,\text{free}}$ from their basis $\mathcal{F}_{t,\text{free}}$. These Gaussians should accurately and compactly model the free space traversed by the sensor rays (*i.e.*, within the viewing frustum). In prior works [14], [15], the free Gaussians $\mathcal{G}_{t,\text{free}}$ are inefficiently constructed from a large number of free-space points sampled at a fixed interval along all sensor rays. In contrast, the free Gaussians $\mathcal{G}_{t,\text{free}}$ in GMMap are directly constructed from their basis $\mathcal{F}_{t,\text{free}}$ with little computational and memory overhead.

The free space is contained within the viewing frustum which is a pyramidal region with significantly different symmetries than each elliptical equipotential surface of the Gaussian distribution. Thus, as illustrated in Fig. 4a, free Gaussians (*i.e.*, ϕ and β) from the basis $\mathcal{F}_{t,\text{free}}$ cannot faithfully represent the free region (especially near the obstacles). To achieve a more accurate representation, we partition the viewing frustum into subregions $\{\mathcal{B}_0, \mathcal{B}_1, \dots\}$ along the z-axis that is perpendicular to the image plane of the camera. Each subregion $\mathcal{B}_i$ is enclosed between two partitioning planes $z = d_{i,\text{max}}$ and $z = d_{i,\text{min}}$. As illustrated in Fig. 4b, free Gaussians are constructed to model each subregion separately.

The free Gaussians in each subregion (Fig. 4b) can be directly recovered from each basis $f = (\phi, \beta)$. Let the index i_f denote the *minimum index* across all subregions containing the endpoints of rays used to construct Gaussian ϕ. For instance, the index $i_f = 2$ for basis $f_0 = (\phi_0, \beta_0)$ in Fig. 4b. The subregion $\mathcal{B}_{i_f}$ is the difference between the region from sensor origin to the obstacle (represented by Gaussian ϕ) and the region from the sensor origin to the partitioning plane $d_{i_f,\text{min}}$ (represented by Gaussian β scaled to $d_{i_f,\text{min}}$). Each remaining subregion is the difference between regions from the sensor origin to two enclosing partitioning planes $d_{i_f,\text{min}}$ and $d_{i_f,\text{max}}$ (represented by Gaussian β scaled to $d_{i_f,\text{min}}$ and $d_{i_f,\text{max}}$). For each basis $f = (\phi, \beta)$, the parameters of the free Gaussian g (*i.e.*, first moment $m_g^{(1)}$, second moment $M_g^{(2)}$, normalizing constant ξ_g, and weight π_g) in subregion $\mathcal{B}_i$ are directly recovered from the parameters of each basis as follows:

$$m_g^{(1)} = \begin{cases} m_\phi^{(1)} - d_{i,\text{min}}^2 m_\beta^{(1)}, & \text{if } i = i_f, \\ m_\beta^{(1)} \left(d_{i,\text{max}}^2 - d_{i,\text{min}}^2\right), & \text{if } 0 \le i < i_f, \end{cases} \tag{19a}$$

$$M_g^{(2)} = \begin{cases} M_\phi^{(2)} - d_{i,\text{min}}^3 M_\beta^{(2)}, & \text{if } i = i_f, \\ M_\beta^{(2)} \left(d_{i,\text{max}}^3 - d_{i,\text{min}}^3\right), & \text{if } 0 \le i < i_f, \end{cases} \tag{19b}$$

$$\xi_g = \pi_g = \begin{cases} \xi_\phi - d_{i,\text{min}} \xi_\beta, & \text{if } i = i_f, \\ \xi_\beta \left(d_{i,\text{max}} - d_{i,\text{min}}\right), & \text{if } 0 \le i < i_f. \end{cases} \tag{19c}$$

See Fig. 4b for an illustration of the recovered free Gaussians $\mathcal{B}_{i,j}$ in subregion $\mathcal{B}_i$ generated from basis f_j.

To retain high mapping fidelity, each subregion is sized according to its spatial resolution (*i.e.*, the density of the sensor rays) such that regions with higher resolution are modeled by smaller Gaussians. Since the sensor rays emanate outwards from the origin, the spatial resolution of each subregion $\mathcal{B}_i$ decreases as its index i increases (see Fig. 4b). To ensure that the maximum size of each Gaussian is inversely proportional to the spatial resolution, the distance between the partitioning planes that enclose each subregion $\mathcal{B}_i$ should increase with index i. Thus, given the maximum slope of the frustum's boundary γ_{frum} along the z axis (see Fig. 4a) and the initial distance d_0 between partitioning planes, the locations of these

Algorithm 2: Free GMM Construction From Basis

Input: Free GMM basis $\mathcal{F}_{t,\text{free}}$
Output: Free GMM $\mathcal{G}_{t,\text{free}}$

1 **function** constructFreeGMM($\mathcal{F}_{t,\text{free}}$)
2 $\quad\mathcal{B} \leftarrow \varnothing, \mathcal{G}_{t,\text{free}} \leftarrow \varnothing$
3 $\quad i_{\max} \leftarrow 0$
$\quad$// Sort basis using its subregion index i_f
4 $\quad$**foreach** $f \in \mathcal{F}_{t,\text{free}}$ **do**
5 $\quad\quad i_f \leftarrow \text{region}(f)$
6 $\quad\quad \mathcal{F}_{t,\text{free}} \leftarrow \mathcal{F}_{t,\text{free}} \setminus f$
7 $\quad\quad \mathcal{B}_{i_f} \leftarrow \mathcal{B}_{i_f} \cup f$
8 $\quad\quad i_{\max} \leftarrow \max(i_{\max}, i_f)$
9 $\quad$**for** ($i = i_{max};\ i \geq 0;\ i = i - 1$) {
$\quad\quad$// See Eqn. (20)
10 $\quad\quad d_{i,\min}, d_{i,\max} \leftarrow \text{computePartitioningPlanes}(i)$
11 $\quad\quad \mathcal{Q} \leftarrow \varnothing$
$\quad\quad$// Recover free Gaussians from bases
12 $\quad\quad$**foreach** $f \in \mathcal{B}_i$ **do**
$\quad\quad\quad$// See Eqn. (19)
13 $\quad\quad\quad g \leftarrow \text{recoverFreeGaussian}(f, d_{i,\min}, d_{i,\max})$
14 $\quad\quad\quad \mathcal{B}_i \leftarrow \mathcal{B}_i \setminus f$
$\quad\quad\quad$// Each tuple q contains a Gaussian q_g generated from its basis q_f
15 $\quad\quad\quad q \leftarrow (q_g \leftarrow g, q_f \leftarrow f)$
16 $\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup q$
$\quad\quad$// Fuse free Gaussians in subregion $\mathcal{B}_i$
17 $\quad\quad$**while** notEmpty($\mathcal{Q}$) **do**
18 $\quad\quad\quad q \leftarrow \text{front}(\mathcal{Q})$
19 $\quad\quad\quad \mathcal{C} \leftarrow \text{findNeighbors}(\mathcal{Q}, q)$
20 $\quad\quad\quad \text{isFused} \leftarrow \text{false}$
21 $\quad\quad\quad$**foreach** $c \in \mathcal{C}$ **do**
$\quad\quad\quad\quad$// See Eqn. (18)
22 $\quad\quad\quad\quad r \leftarrow \text{fuseGaussianAndBasis}(c, q)$
23 $\quad\quad\quad\quad d_h \leftarrow \text{unscentHellingerDistance}(r_g, c_g, q_g)$
24 $\quad\quad\quad\quad s_r \leftarrow \text{geometricSimilarity}(c_g, q_g)$
25 $\quad\quad\quad\quad$**if** $d_h \leq s_r \cdot \alpha_{h,free}$ **then**
26 $\quad\quad\quad\quad\quad q \leftarrow r$
27 $\quad\quad\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \setminus c$
28 $\quad\quad\quad\quad\quad \text{isFused} \leftarrow \text{true}$
29 $\quad\quad\quad$**if** isFused = false **then**
30 $\quad\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \setminus q$
31 $\quad\quad\quad\quad \mathcal{G}_{t,\text{free}} \leftarrow \mathcal{G}_{t,\text{free}} \cup q_g$
$\quad\quad\quad\quad$// Propagate fusion decision to subregion $\mathcal{B}_{i-1}$
32 $\quad\quad\quad\quad$**if** $i > 0$ **then**
33 $\quad\quad\quad\quad\quad \mathcal{B}_{i-1} \leftarrow \mathcal{B}_{i-1} \cup q_f$
34 $\quad$**return** $\mathcal{G}_{t,\text{free}}$

planes for each subregion $\mathcal{B}_i$ are computed as

$$d_{i,\max} = d_0 \sum_{k=0}^{k=i} (1 + \alpha_d \gamma_{\text{frum}})^i = \frac{d_0 \left((1 + \alpha_d \gamma_{\text{frum}})^{i+1} - 1 \right)}{\alpha_d \gamma_{\text{frum}}}, \tag{20a}$$

$$d_{i,\min} = \begin{cases} 0, & \text{if } i = 0, \\ d_{i-1,\max}, & \text{otherwise}, \end{cases} \tag{20b}$$

where α_d is a scaling parameter. We chose $\alpha_d = 0.5$ in all our experiments.

Although free Gaussians recovered from the basis can accurately model the free region, they are not as compact as the occupied GMM $\mathcal{G}_{t,\text{occ}}$. Thus, after recovery, free Gaussians are fused with each other in each subregion to further enhance the compactness of the map. Alg. 2 efficiently performs Gaussian recovery and fusion. After sorting each basis f into its associated subregion based on index i_f (Line 4 to 8), free Gaussians are constructed within each subregion (from Line 9 onward) starting from the one that is furthermost away from the sensor origin (see the green arrow in Fig. 4b). Using the bases, free Gaussians in each subregion $\mathcal{B}_i$ are initially recovered (Line 12 to 16) and then fused with each other one pair at a time in a region-growing approach (Line 17 to 33).

During region growing, we need to ensure that the fused Gaussian can still accurately represent the free region within each subregion $\mathcal{B}_i$. After fusing a free Gaussian q_g with its neighbor c_g (determined by whether their bounding boxes intersect in Line 19), we accept the fused Gaussian r_g (in Line 26) if it accurately represents its original components (*i.e.*, q_g and c_g). In prior works [14], the fused Gaussian r_g is accepted if the probabilistic distance d_h between two components q_g and c_g are below a pre-defined low threshold $\alpha_{h,\text{free}}$. Thus, only Gaussians that completely overlap the same region can be fused (see Fig. 5a). However, there exist many opportunities to fuse Gaussians that only partially overlap but accurately represent neighboring parts of the same region (see Fig. 5b). To also exploit these opportunities, our distance measure d_h is computed between the fused Gaussians r_g and its components $\{q_g, c_g\}$ using the Unscented Hellinger Distance [33] in Line 23. To maintain mapping accuracy, we scale the distance threshold $\alpha_{h,\text{free}}$ using the geometric similarity $s_r \in [0, 1]$ between Gaussians q_g and c_g in Line 25. The geometric similarity s_r between the two components is computed as the intersection over union ratio for the z dimension of their bounding boxes in Line 24.

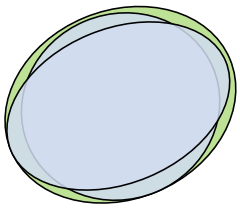
Even though free Gaussians are constructed to separately represent each subregion $\mathcal{B}_i$, the fusion decision made between Gaussians (Line 25) in the current subregion $\mathcal{B}_i$ can be propagated to reduce the number of computations in subsequent subregions. The Gaussians recovered by the same basis across most subregions are almost relatively similar in shape (*e.g.*, Gaussians $b_{2,1}$, $b_{1,1}$ and $b_{0,1}$ in Fig. 4b). Thus, the successful fusion between two Gaussians in the current subregion $\mathcal{B}_i$ (*e.g.*, between $b_{2,1}$ and $b_{2,2}$) implies the same for other subregions $\mathcal{B}_{i-1}, \ldots, \mathcal{B}_0$ (*e.g.*, between $b_{1,1}$ and $b_{1,2}$, $b_{0,1}$ and $b_{0,2}$). To automatically propagate the fusion decision from the current subregion $\mathcal{B}_i$, the fused basis q_f is simply transferred into the following subregions ($\mathcal{B}_{i-1}, \ldots, \mathcal{B}_0$) at Line 33 across multiple iterations of the outer loop (Line 9).

Construct Local Map (Lines 4 to 6 in Alg. 1): To enable the fusion between the local GMMs (occupied $\mathcal{G}_{t,\text{occ}}$ and free $\mathcal{G}_{t,\text{free}}$) and the global map $\mathcal{M}_{t-1}$ in Section IV-C, local GMMs need to transform into the world frame as follows:

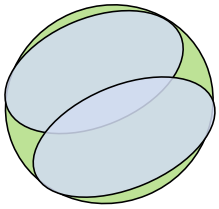
$$\mu_{\boldsymbol{X}} \leftarrow R_t \mu_{\boldsymbol{X}} + \epsilon_t, \ \ \Sigma_{\boldsymbol{X}} \leftarrow R_t \Sigma_{\boldsymbol{X}} R_t^\top, \tag{21}$$

where R_t and ϵ_t are the rotation and translation matrix associated with pose T_t. The mean $\mu_{\boldsymbol{X}}$ and covariance $\Sigma_{\boldsymbol{X}}$ for each Gaussian in the GMM are defined in Eqn. (3).

After the transformation, an R-tree is created for all Gaussians in Line 6 to form the local map $\mathcal{G}_t$. First, a bounding box is constructed for each Gaussian to enclose its ellipsoidal



(a) Fusion of two completely overlapping Gaussians (blue) into a single Gaussian (green).



(b) Fusion of two partially overlapping Gaussians (blue) into a single Gaussian (green).

Fig. 5. Our fusion criteria using the Unscented Hellinger Distance [33] allows for the creation of a single Gaussian (green) from two Gaussians (blue) when they (a) completely overlap to represent the same region or (b) partially overlap to represent neighboring parts of the same region in the environment.

bound as defined in Eqn. (12). Then, each Gaussian and its bounding box are inserted into the R-tree as shown in Fig. 2.

C. Globally-Consistent GMM Fusion

In this section, we present a novel memory-efficient procedure in Alg. 3 to directly update the global GMMap $\mathcal{M}_{t-1}$ in place using Gaussians from the local GMMap $\mathcal{G}_t$. When the robot obtains a new depth image Z_t, rays associated with a subset of pixels in the image traverse through a previously observed region $\mathcal{C}_t$ that is modeled in the global map $\mathcal{M}_{t-1}$. To retain the compactness of the map, these rays should be fused with the map to update the state of the region $\mathcal{C}_t$.

In prior works [12], [13], [15], [20], the ray associated with each pixel in the image is cast into the global map to determine and update the region $\mathcal{C}_t$. Since these rays emanate outwards from the sensor origin, accessing the global map in memory along these rays often lacks spatial and temporal locality for effective cache usage. Thus, casting all rays (more than 300,000 in each 640×480 depth image) requires significant time and energy for accessing the DRAM where the map is stored. In GMMap, the rays from the depth image are accurately compressed into a local GMMap $\mathcal{G}_t$ whose geometric properties are exploited to *i)* quickly find the region $\mathcal{C}_t$ in the global map $\mathcal{M}_{t-1}$ with little memory access, and *ii)* directly update the region $\mathcal{C}_t$ using Gaussians in $\mathcal{G}_t$ to maintain the compactness and accuracy of the resulting global map $\mathcal{M}_t$.

Fig. 6 illustrates the entire procedure for fusing the local map $\mathcal{G}_t$ into the global map $\mathcal{M}_{t-1}$. Recall that from Section IV-B, Gaussians in the local map $\mathcal{G}_t$ are already transformed in the world frame and organized using an R-tree. Using the bounding box (at the root node of the R-tree) that encloses $\mathcal{G}_t$, Gaussians in the previously observed region $\mathcal{C}_t$ can be extracted from the previous global map $\mathcal{M}_{t-1}$ in a *single traversal* through its R-tree (Line 2) without ray casting. After extraction, Gaussians in the region $\mathcal{C}_t$ are *directly* fused with the local map $\mathcal{G}_t$ (Lines 5 to 25). Since the Gaussians in $\mathcal{C}_t$ and $\mathcal{G}_t$ are extremely compact for storage within the on-chip cache, the entire fusion process is expected to require little DRAM accesses. After completion, the fused local map (*i.e.*, $\mathcal{G}_t \cup \mathcal{C}_t$) is simply appended to the previous global map $\mathcal{M}_{t-1}$ in Line 26 to produce the updated global map $\mathcal{M}_t$.

Our fusion process (Lines 5 to 25) enhances the compactness of the local map $\mathcal{G}_t$ while maintaining its accuracy. For each Gaussian $c \in \mathcal{C}_t$, the R-tree in the local map $\mathcal{G}_t$ is used to efficiently search for the set of Gaussians $\mathcal{Q}$ that intersects with and represents the same type of region (*i.e.*, free or occupied) as c. In Line 14, the Gaussian c is fused with each neighbor $q \in \mathcal{Q}$ into a fusion candidate r using Eqn. (18). Similar to Alg. 2, the fusion candidate r is accepted in Line 18 if the Unscented Hellinger Distance [33] between candidate r and its components $\{c, q\}$ is less than a distance threshold α_h. Recall that our fusion criteria can exploit a wide range of scenarios (*i.e.*, Fig. 5a and 5b) to enhance the compactness of the map. To maintain accuracy, we scale the distance threshold α_h in Line 17 using the geometric similarity s_r (computed in Line 16) between components c and q. When c and q are free Gaussians that represent volumes, the similarity measure s_r is the intersection over union ratio of their 3D bounding boxes. When c and q are obstacle Gaussians that represent surfaces, the similarity measure s_r is the intersection over union ratio of the largest two dimensions of their 3D bounding boxes (*i.e.*, surface coverage similarity) multiplies by the dot product of their normal vectors (*i.e.*, orientation similarity).

Algorithm 3: Globally-Consistent GMM Fusion

Input: Local GMMap $\mathcal{G}_t$, previous global GMMap $\mathcal{M}_{t-1}$
Output: Updated global GMMap $\mathcal{M}_t$

```
1  function updateGlobalMap(M_{t-1}, G_t)
2      C_t ← findObservedRegion(G_t, M_{t-1})
3      M_{t-1} ← M_{t-1} \ C_t
4      M_{t-1} ← updateRtree(M_{t-1})
5      foreach c ∈ C_t do
6          if isFreeGaussian(c) then
7              Q ← findIntersectingFreeGaussians(G_t, c)
8              α_h ← α_{h,free}
9          else
10             Q ← findIntersectingObsGaussians(G_t, c)
11             α_h ← α_{h,occ}
12         isObserved ← false
13         foreach q ∈ Q do
14             r ← fuseGaussians(c, q)        // See Eqn. (18)
15             d_h ← unscentHellingerDistance(r, c, q)
16             s_r ← geometricSimilarity(c, q)
17             if d_h ≤ s_r · α_h then
18                 c ← r
19                 G_t ← G_t \ q
20                 G_t ← updateRtree(G_t)
21                 isObserved ← true
22         if isObserved then
23             C_t ← C_t \ c
24             G_t ← G_t ∪ c
25             G_t ← updateRtree(G_t)
26     M_t ← M_{t-1} ∪ C_t ∪ G_t
27     M_t ← updateRtree(M_t)
28     return M_t
```

V. Experimental Results & Analysis

In this section, we compare our GMMap against current state-of-the-art frameworks in terms of accuracy, throughput, memory footprint, and energy consumption. Specifically, we chose the following frameworks with open-source implementations and different types of occupancy representations: Oc-

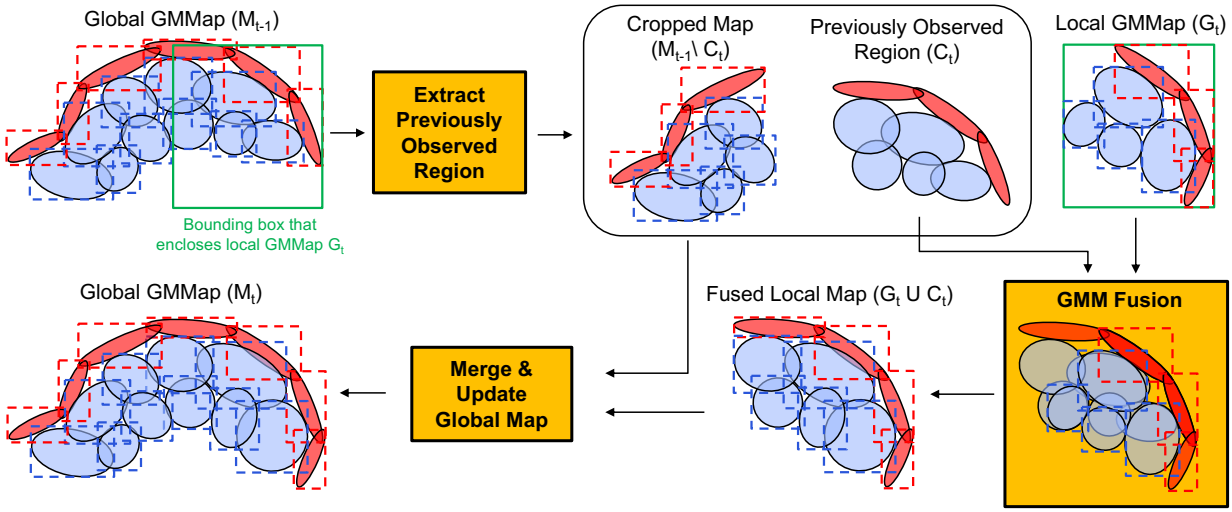


Fig. 6. **Globally-consistent GMM fusion**: Constructing the current global GMMap $\mathcal{M}_t$ by fusing the local GMMap $\mathcal{G}_t$ into the previous global GMMap $\mathcal{M}_{t-1}$. The bounding box (green rectangle) of local map $\mathcal{G}_t$ is used to determine the Gaussians $\mathcal{C}_t$ in the global map $\mathcal{M}_{t-1}$ that overlaps with $\mathcal{G}_t$. Occupied and free GMMs are illustrated with red and blue ellipsoids, respectively. Dotted rectangles represent the bounding boxes at the leaf nodes of the R-tree.

toMap[1] [12] (discrete), NDT-OM[2] [13] (semi-parametric), and BGKOctoMap-L[3] [20] (non-parametric). This comparison was performed using four diverse indoor and outdoor environments (*i.e.*, *Room*, *Warehouse*, *Soulcity*, and *Gascola*) generated from sequences of depth images and ground-truth poses.

Table I summarizes the characteristics of all four environments. In particular, *Room* (from real-world TUM-RGBD datasets [34]) is a small structured environment that models crowded cubicles inside an office. *Warehouse* (from real-world TUM-RGBD datasets [34]) is a larger structured indoor environment captured using a longer and noisier range of the Kinect camera. In contrast, *Soulcity* (from synthetic TartanAir dataset [35]) is a large structured outdoor environment in a city containing several multi-story buildings with intricate sets of walkways. Finally, *Gascola* (from synthetic TartanAir dataset) is a large unstructured outdoor environment in a forest consisting of trees and a small hill.

To emulate an energy-constrained setting, all experiments were performed on the low-power NVIDIA Jetson TX2 platform in MAXP_CORE_ARM power mode [36]. All frameworks, implemented in C++, were compiled using the same settings. To reduce the memory overhead and map size, the floating point variables (and their associated operations) across all frameworks are stored as (and performed in) 32-bit single precision. Our single-core, multi-core, and GPU-accelerated GMMap implementations (visualized in Open3D [37]) can be obtained at https://github.com/mit-lean/GMMap.

Prior works achieve high mapping accuracy but are neither as computationally nor memory efficient as GMMap. After carefully selecting a set of hyperparameters for all frameworks (Section V-A), GMMap is evaluated across a diverse set of indoor and outdoor environments and achieves comparable accuracy as prior works (Section V-B). In addition, our GMMap is highly parallelizable and can be constructed in real-time at up to 81 images per second, which is $4\times$ to $146\times$ higher than prior works on the low-power Jetson TX2 platform (Section V-C). Due to single-pass depth image compression in SPGF* and directly operating on Gaussians during map construction, our GMMap is extremely memory efficient. Compared with prior works (in Section V-D), our CPU implementation reduces *i)* the map size by at least 56%, *ii)* the memory overhead for storing input and temporary variables by at least 88%, and *iii)* the number of DRAM accesses by at least 78% during map construction. Thus, in Section V-E, the computational and memory efficiency of our GMMap reduces energy consumption by at least 69% compared with prior works.

[1]https://github.com/OctoMap/octomap
[2]https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic
[3]https://github.com/RobustFieldAutonomyLab/la3dm

A. Selection of Hyperparameters

In this section, we discuss the selection of hyperparameters for the GMMap (Section V-A1). Then, we briefly summarize the hyperparameters selected for prior frameworks (*i.e.*, NDT-OM, BGKOctoMap-L, and OctoMap, in Section V-A2). The hyperparameters of all frameworks are presented in Table II and are manually tuned to reduce the size of the maps without significant deviation from their peak accuracy.

1) ***GMMap****:* For GMMap, the hyperparameters are the unexplored prior weight π_0 for the prior distribution in Eqn. (4),

TABLE I
PROPERTIES OF ALL FOUR ENVIRONMENTS USED FOR EVALUATION.

Environment	Dimensions (m)	Images	Depth Image Resolution	Avg. Sensor Range (m)
Room (freiburg1_room)	11.28 × 12.05 × 3.45	1311	640×480	0.97
Warehouse (freiburg2_ pioneer_slam)	23.52 × 17.90 × 4.29	2169	640×480	1.13
Soulcity	73.90 × 62.41 × 42.69	1083	640×480	10.85
Gascola	59.04 × 52.93 × 33.71	382	640×480	4.06

the initial distance d_0 between partitioning planes in Eqn. (20), and the distance thresholds ($\alpha_{h,\text{free}}$ and $\alpha_{h,\text{occ}}$) for fusing Gaussians in Alg. 2 and 3. These hyperparameters are mostly dependent on the sensor itself (*i.e.*, not on the environment). Recall that the Gaussian fusion thresholds ($\alpha_{h,\text{free}}$ and $\alpha_{h,\text{occ}}$) control the trade-off between compactness and the accuracy of the map. For environments (*i.e.*, *Room* and *Warehouse*) captured using a noisy sensor (*i.e.*, Kinect), we set both thresholds lower to better capture the surface details and the noise of the sensors at the expense of compactness, as shown in Table II. For environments (*i.e.*, *Soulcity* and *Gascola*) captured using an ideal noiseless sensor, Gaussians that represent the same region appear much more similar. Thus, we set both fusion thresholds higher to achieve better compactness without sacrificing accuracy.

Recall that from Eqn. (20), the (initial) partition plane distance d_0 affects the locations and number of free space subregions in each local map. Since the field of views for our cameras are quite similar across all environments, the distance d_0 is almost constant in Table II. Finally, the unexplored prior weight π_0 controls the evidence of the unexplored region. Thus, a larger π_0 would require *i)* more measurements for the occupancy probability to converge from unexplored state (*i.e.*, 0.5) to free (*i.e.*, 0) or occupied state (*i.e.*, 1), and *ii)* reduce the amount of free region spatially interpolated from the frontier (*i.e.*, the boundary between free and unexplored region) into the unexplored region. Since the number of pixels in the depth image is constant across all environments, the unexplored prior weight π_0 is unchanged in Table II to retain the same convergence rate of occupancy across environments.

*2) **Existing Frameworks**:* For NDT-OM, BGKOctoMap-L, and OctoMap, the environment is voxelized so that the minimum voxel size is the hyperparameter. In our experiments, we increase the voxel size with the size of the environment to maintain a good trade-off between accuracy and the compactness of the map in Table II. Since BGKOctoMap-L partitions the environment into equally-sized cubic blocks such that each block contains an octree (*i.e.*, defined as the test-data octrees [20]), the depth of the octree in each block is an additional hyperparameter which is kept constant across all environments. To generate compact training data representing free regions, BGKOctoMap-L samples points along each sensor ray at a free resolution interval. In Table II, we increase this interval with the size of the environment to maintain a good trade-off between accuracy and map construction throughput.

B. Accuracy of Occupancy Estimation

In this section, we compare the accuracy of the GMMap against NDT-OM, BGKOctoMap-L, and OctoMap. Specifically, we are interested in the accuracy of occupancy estimation in the occupied and free regions (Section V-B1) as well as the characteristics of the obstacle surfaces (at the free-to-occupied regions, in Section V-B2) and frontiers (at the free-to-unexplored regions, in Section V-B3). Recall that the hyperparameters in Table II are manually tuned to reduce the size of the maps without significant deviation from their peak accuracy for estimating occupancy probability. Depending on the downstream applications, these hyperparameters are also used to control two fundamental trade-offs within the GMMap, as discussed in Sections V-B1 and V-B3. For our experiments, we evaluated the occupancy probabilities at the end of all sensor rays for validating occupied regions, and along all sensor rays at 10 cm intervals for validating free regions. Fig. 7 and 8 illustrate each framework in *Room* (structured indoor) and *Gascola* (unstructured outdoor) environment, respectively.

*1) **Occupied & Free Regions**:* We use the receiver operating characteristics (ROC) curve to compare the accuracy for estimating occupancy in both occupied and free regions across all frameworks. To generate the ROC curves, the occupancy probability is queried from each map at the locations of all sensor rays used to construct the map. By sweeping the thresholds for classifying occupied or free regions from each occupancy probability, the true positive rate (*i.e.*, the proportion of correct classifications during the prediction of occupied regions) of each map varies with the false positive rate (*i.e.*, the proportion of incorrect classifications during the prediction of the free regions). In addition, the area under the curve (AUC) represents the probability that the map estimates a higher occupancy for the occupied region than that of the free region [38]. Thus, a map with high accuracy should generate a ROC curve that tends towards the upper-left corner of the plot to achieve a large AUC close to one.

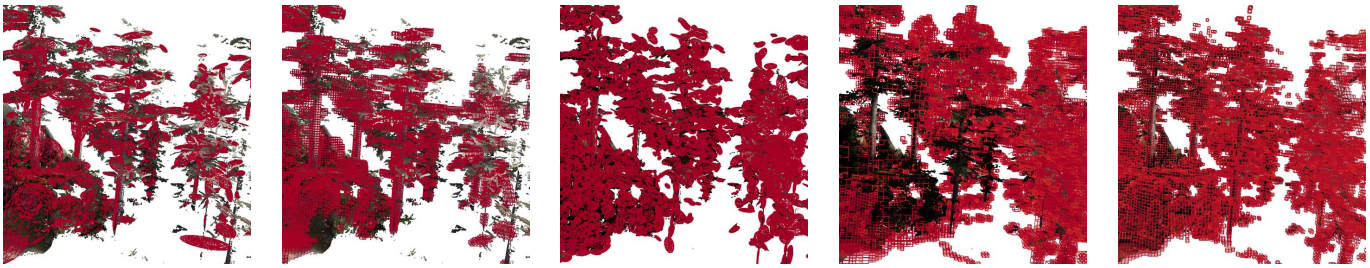
Fig. 9 illustrates the ROC curve for each framework across all environments. The (ROC) AUC for the GMMap is slightly higher than other frameworks in structured indoor (*i.e.*, *Room* and *Warehouse*) and outdoor (*i.e.*, *Soulcity*) environments. In addition, the ROC curves of other frameworks are mostly under the ROC curves of GMMap, which indicates Gaussians' ability to accurately model both volumetric free regions and thin obstacle surfaces with fewer false positives (*i.e.*, mistakenly predicting free regions as occupied) without sacrificing true positives (*i.e.*, correctly predicting occupied regions). In contrast, voxels in OctoMap and BGKOctoMap-L inflate the obstacle surfaces with cubic artifacts whose thickness is

TABLE II
HYPERPARAMETERS USED IN GMMAP, NDT-OM, BGKOCTOMAP-L, AND OCTOMAP ACROSS ALL FOUR ENVIRONMENTS.

Environment	GMMap				NDT-OM	BGKOctoMap-L			OctoMap
	Unexplored Prior Weight (π_0)	Partition Plane Distance (d_0)	Free Gaussian Fusion Threshold ($\alpha_{h,\text{free}}$)	Occupied Gaussian Fusion Threshold ($\alpha_{h,\text{occ}}$)	Voxel Size	Voxel Size	Free Resolution	Block Octree Depth	Voxel Size
Room	500,000	0.5m	0.26	0.70	0.4m	0.1m	0.3m	3	0.1m
Warehouse	500,000	0.5m	0.26	0.70	0.5m	0.1m	0.3m	3	0.2m
Soulcity	500,000	0.5m	0.63	1.41	1.2m	0.3m	3.0m	3	0.3m
Gascola	500,000	0.6m	0.63	1.41	1.2m	0.3m	3.0m	3	0.3m

(a) GMMap (Gaussians) (b) GMMap (Voxels) (c) NDT-OM (d) BGKOctoMap-L (e) OctoMap

Fig. 7. Visualization of the ground-truth *Room* (structured indoor) environment overlaid with the following mapping frameworks: (a) GMMap (obstacle Gaussians), (b) GMMap (occupied voxels from uniform voxel-grid sampling), (c) NDT-OM (obstacle Gaussians), (d) BGKOctoMap-L (occupied voxels), and (e) OctoMap (occupied voxels). Free regions are not illustrated for ease of visualization. Even though Gaussians are unbounded and continuous, each obstacle Gaussian is visualized at a Mahalanobis distance of two using an ellipsoidal wireframe in (a) and (c). For BGKOctoMap-L, OctoMap, and GMMap (from uniform voxel-grid sampling), wireframes of occupied voxels with an occupancy probability greater than 0.9 are visualized. For both GMMap and NDT-OM, the surface boundaries of the obstacles are smooth and often *extend beyond their ellipsoidal wireframes* (*e.g.*, compare (a) and (b) for GMMap).



(a) GMMap (Gaussians) (b) GMMap (Voxels) (c) NDT-OM (d) BGKOctoMap-L (e) OctoMap

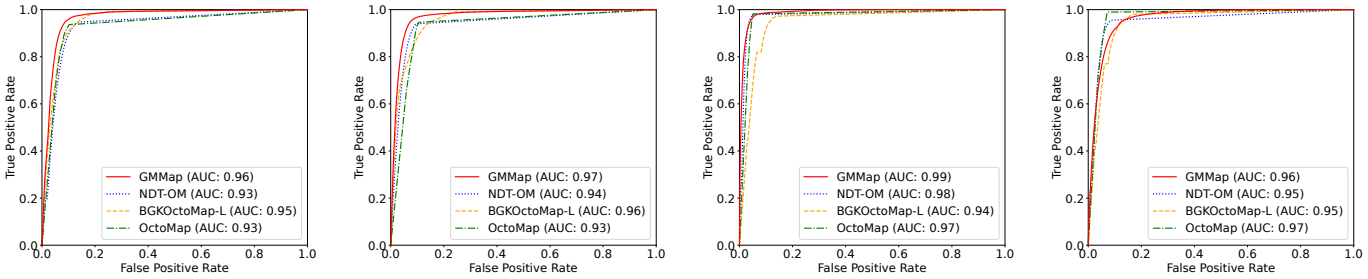
Fig. 8. Visualization of the ground-truth *Gascola* (unstructured outdoor) environment overlaid with the following mapping frameworks: (a) GMMap (obstacle Gaussians), (b) GMMap (occupied voxels from uniform voxel-grid sampling), (c) NDT-OM (obstacle Gaussians), (d) BGKOctoMap-L (occupied voxels), and (e) OctoMap (occupied voxels). Free regions are not illustrated for ease of visualization. Even though Gaussians are unbounded and continuous, each obstacle Gaussian is visualized at a Mahalanobis distance of two using an ellipsoidal wireframe in (a) and (c). For BGKOctoMap-L, OctoMap, and GMMap (from uniform voxel-grid sampling), wireframes of occupied voxels with an occupancy probability greater than 0.9 are visualized. For both GMMap and NDT-OM, the surface boundaries of the obstacles are smooth and often *extend beyond their ellipsoidal wireframes* (*e.g.*, compare (a) and (b) for GMMap).

at least equal to the minimum voxel size, which leads to higher false positives. Even though NDT-OM also utilizes Gaussians to represent obstacle surfaces, these Gaussians are constructed under the assumption that all measurements within each voxel belong to the same surface, and thus also suffer from voxelization artifacts when such assumption is invalid (*e.g.*, at corners of objects in Fig. 7c).

For unstructured outdoor environment (*i.e.*, *Gascola*), the (ROC) AUC of GMMap is comparable but slightly lower than OctoMap as shown in Fig. 9d. To achieve a compact representation and avoid modeling spurious measurements, GMMap prunes away small occupied Gaussians containing less than a certain number of measurements (*i.e.*, 200 in our experiments). While this pruning threshold does not affect the modeling of objects closer to the camera (typically containing more measurements), this threshold prunes away obstacle Gaussians for small objects (*i.e.*, leaves on the tree in Fig. 8a) that are far away from the camera (and robot).

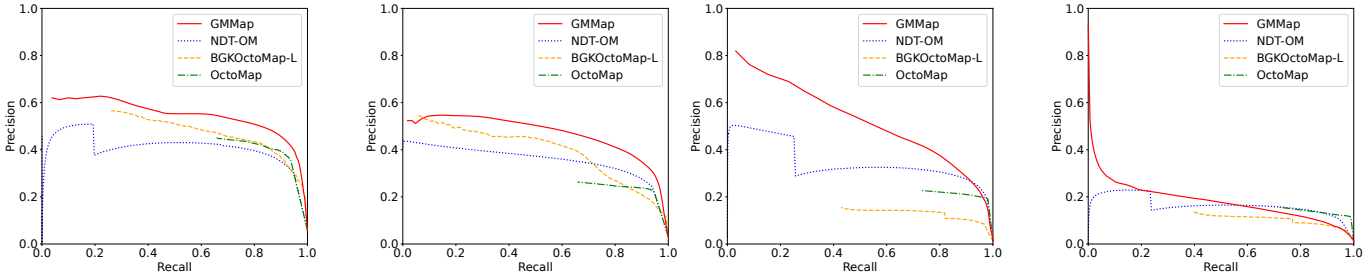
In fact, the Gaussian pruning threshold controls one of the fundamental trade-offs between the ability to model small distant objects and the compactness of the GMMap. For applications that require the mapping of distant small objects, the pruning threshold can be decreased to retain these objects at the expense of a larger map size. However, pruning these small occupied Gaussians does not degrade the safety of the map. In SPGF*, both the occupied regions associated with these pruned Gaussians and free regions near them will remain unexplored because free Gaussian bases leading up to (and associated with) these obstacle Gaussians are also pruned. This is reflected in Fig. 9d where the ROC curve of GMMap shifts to the right (*i.e.*, higher false positives for free regions near these pruned Gaussians) compared with other frameworks. If the robot recaptures these previously-neglected small objects at a closer distance later, these objects will be modeled by more measurements and thus retained in GMMap.

*2) **Obstacle Surfaces (at Free-to-Occupied Regions)**:* We used the precision-recall curves to compare the modeling of the obstacle surfaces among all frameworks at different occupancy decision thresholds. In Fig. 10, no precision-recall curve can reach the y-axis because precision becomes invalid when the occupancy decision threshold equals or exceeds the maximum occupancy probability in the map (*i.e.*, 0.97 for OctoMap and 1.0 for others). In addition, no framework achieves 100% precision due to the inflation/thickening of obstacle surfaces from either voxelization (in OctoMap and BGKOctoMap-L) or smooth decay of Gaussians (in GMMap and NDT-OM). To ensure safe navigation, we are interested in the precision of obstacle surfaces when the recall is high. In structured environments (*i.e.*, *Room*, *Warehouse*, and *Soulcity*), GMMap often achieves higher precision at similar recall compared with



(a) *Room* (b) *Warehouse* (c) *Soulcity* (d) *Gascola*

Fig. 9. Comparison of receiver operating characteristic (ROC) for classifying occupied and free regions among GMMap, OctoMap, NDT-OM, and BGKOctoMap-L in four environments: (a) *Room*, (b) *Warehouse*, (c) *Soulcity*, and (d) *Gascola*. The area under the ROC curve (AUC) equals to the probability that an occupied region is assigned a higher occupancy probability than the free region in the map. From the AUC of the ROC curves, the accuracy of occupancy estimation in occupied and free regions for the GMMap is comparable with other frameworks across all environments.



(a) *Room* (b) *Warehouse* (c) *Soulcity* (d) *Gascola*

Fig. 10. Comparison of precision-recall curves for modeling obstacle surfaces among GMMap, OctoMap, NDT-OM, and BGKOctoMap-L in four environments: (a) *Room*, (b) *Warehouse*, (c) *Soulcity*, and (d) *Gascola*. No precision-recall curve can reach the y-axis because precision becomes invalid when the occupancy decision threshold equals or exceeds the maximum occupancy probability in the map (*i.e.*, 0.97 of OctoMap and 1.0 for others). In addition, no framework achieves 100% precision due to the inflation of obstacle surfaces from voxelization (in OctoMap and BGKOctoMap-L) or smooth decay of Gaussians (in GMMap and NDT-OM). For NDT-OM and GMMap, the occupancy probability converges to one at the mean of each obstacle Gaussian. Since NDT-OM models all obstacle surfaces within each voxel by a single Gaussian, the mean of such Gaussian could be located in a free region between multiple surfaces which leads to a sharp decrease in both precision and recall at high occupancy decision threshold. Since Gaussians in the GMMap overlap and flexibly adapt to the thickness, location, and orientation of each surface, GMMap achieves higher precision at similar recall than other frameworks across most environments.

other frameworks because Gaussians in the GMMap overlap and flexibly adapt to the thickness, location, and orientation of each surface. In other frameworks, the precision is lower due to higher false positives caused by *i)* using Gaussian to model multiple distinct surfaces in each voxel (in NDT-OM) or *ii)* thickening of surfaces induced by the minimum size of the voxels (in OctoMap and BGKOctoMap-L). For unstructured outdoor environment (*i.e.*, *Gascola* in Fig. 10d), the precision of GMMap is lower than other frameworks at higher recall due to the pruning of Gaussians representing small distant objects far away from the robot. As explained in Section V-B1, the locations of these pruned Gaussians will remain unexplored and thus do not affect the safety of navigation.

*3) **Frontiers (at Free-to-Unexplored Regions)**:* The visualization of GMMap and the preservation of unexplored regions for *Warehouse* and *Soulcity* are illustrated in Fig. 11 and 12, respectively. Unlike discrete representations that assume occupancy is spatially independent, Gaussians allow the GMMap to spatially interpolate occupancy to achieve higher compactness. Thus, another fundamental trade-off of the GMMap is the total extent of free regions classified for safe navigation *v.s.* the extent of free regions interpolated by free Gaussians into the unexplored region at the frontiers (*i.e.*, boundaries between free and unexplored regions). This trade-off can be controlled using the unexplored prior weight (π_0) that dictates the contribution of unexplored regions during GMR. Specifically, if the application is more sensitive to the exact locations of the frontier, the prior weight (π_0) can be increased to reduce the total extent of free regions classified by GMMap for safe traversal. Even if the prior weight (π_0) is unchanged, the frontiers (and regions near obstacle surfaces) can be identified at locations of high occupancy variance computed using Eqn. (11).

C. Construction & Query Throughput

In this section, we compare the computational efficiency of our GMMap against other frameworks using the NVIDIA Jetson TX2 platform. The computational efficiency is evaluated in terms of the throughput for constructing the map (*i.e.*, depth images per second, in Section V-C1) and also querying the map (*i.e.*, locations per second, in Section V-C2). Table III summarizes these metrics for all frameworks.

*1) **Map Construction**:* The NVIDIA Jetson TX2 platform contains a low-power ARM Cortex A57 CPU with four cores and a Pascal GPU with two Streaming Multiprocessors (SMs). Due to computationally efficient GMM generation and fusion, our GMMap can be constructed at a throughput of 11 to 18 images per second using only one CPU core, which is $4\times$ to

(a) *Warehouse* point cloud (b) GMMap (c) Occupancy distribution at a cross section

Fig. 11. Visualization of (a) the point cloud overlaid with its (b) GMMap for *Warehouse* (structured indoor) environment. For ease of visualization, only occupied Gaussians are shown for the GMMap. In (c), the distribution of occupancy is visualized in the free regions (blue), unexplored regions (yellow), and occupied regions (red). The locations of the unexplored regions are well-preserved. Depending on the application requirements, the unexplored prior weight (π_0) of the GMMap can be used to increase (*i.e.*, smaller π_0) or decrease (*i.e.*, larger π_0) the extent of free regions interpolated into the unexplored region during Gaussian Mixture Regression.

(a) *Soulcity* point cloud (b) GMMap (c) Occupancy distribution at a cross section

Fig. 12. Visualization of (a) the point cloud overlaid with its (b) GMMap for *Soulcity* (structured outdoor) environment. For ease of visualization, only occupied Gaussians are shown for the GMMap. In (c), the distribution of occupancy is visualized in the free regions (blue), unexplored regions (yellow), and occupied regions (red). The locations of the unexplored regions are well-preserved. Depending on the application requirements, the unexplored prior weight (π_0) of the GMMap can be used to increase (*i.e.*, smaller π_0) or decrease (*i.e.*, larger π_0) the extent of free regions interpolated into the unexplored region during Gaussian Mixture Regression.

$36\times$ higher than other frameworks. Since Scanline Segmentation (Line 12 in Alg. 1) dominates the amount of computation during map construction and can be concurrently executed across multiple rows of the depth image, our construction throughput can be significantly increased via parallelization. By using all four CPU cores, our multi-core implementation reaches a throughput of 31 to 60 images per second. Multi-core implementations of existing frameworks are either not publicly available or highly experimental. Even if these frameworks can be effectively parallelized with four cores, their throughputs are expected to be $4\times$ higher, which are still much lower than our multi-core implementation. By concurrently executing Scanline Segmentation across four images, our GPU implementation of GMMap offers the highest construction throughput of 44 to 81 images per second, which is up to $2\times$ higher[4] than our CPU multi-core implementation.

*2) **Occupancy Query**:* Table III also compares the query throughput of our GMMap against existing frameworks. To emulate an energy-constrained setting during path planning, each map is queried at locations throughout all observed regions (*i.e.*, no unexplored regions) in the environment using only a single CPU core. Recall that each map consists of geometric primitives (*e.g.*, Gaussians or voxels) stored using a spatial data structure (*e.g.*, grid, R-tree, or octree). For existing frameworks, either traversing the spatial data structure (*i.e.*, accessing a voxel from a grid in NDT-OM) or inferring occupancy from primitives (*i.e.*, reading occupancy probability in BGKOctoMap-L and OctoMap) require little compute, which leads to high query throughputs ranging from 9.3×10^5 to 4.2×10^6 locations per second. However, in our GMMap, both R-tree traversal and Gaussian Mixture Regression (GMR) require more computation. Thus, the query throughput is lower than other frameworks but still sufficiently high (ranging from 4.6×10^5 to 7.9×10^5 locations per second). If needed, the query throughput can be increased by accessing the map with multiple cores and/or partitioning the query locations into more localized sets for batch processing.

[4]Even though we are processing four images at the same time, the throughput is not four times higher because other sequential procedures of the GMMap construction (*i.e.*, segment and GMM fusion) start to dominate.

D. Memory Footprint

In this section, we compare the memory efficiency of our GMMap against other frameworks when executing on the NVIDIA Jetson TX2 platform. In addition to the map size (Section V-D1), we are interested in the memory overhead (for storing input and temporary variables, in Section V-D2) and the amount of DRAM access per pixel (which dictates DRAM energy consumption, in Section V-D3) during the map construction. Table III summarizes our results.

*1) **Map Size**:* We compare the size of the map that includes the geometric primitives (*e.g.*, Gaussians and/or voxels) and the spatial data structure (*i.e.*, R-tree, grid and/or octree) among all frameworks. Due to the compactness and strong representational power of the Gaussians, NDT-OM achieves comparable accuracy while reducing the map size by 61% to 96% compared with BGKOctoMap-L and OctoMap. However, the extent of each Gaussian in NDT-OM is restricted by the constant voxel size across the entire environment. Thus, all Gaussians appear similarly sized as shown in Fig. 7c and 8c. By using SPGF* to construct Gaussians that appropriately adapt to the geometries of occupied and free regions in the environment (see Fig. 7a and 8a), our GMMap achieves comparable accuracy while reducing the map size by 56% to 73% compared with NDT-OM. Across all frameworks, GMMap requires the least amount of memory (167KB to 850KB) across all four environments.

*2) **Memory Overhead**:* We are interested in the memory overhead[5] (defined as the peak memory usage minus the map size) for storing input and temporary variables during map construction. For a memory-efficient framework, its memory overhead should be insignificant compared with the map size. Unfortunately, existing frameworks are not memory efficient. For NDT-OM, the memory overhead mostly comprises the point cloud associated with each depth image (up to 3.6MB) for supporting a variety of edge cases during recency-weighted covariance update [13]. For BGKOctoMap-L, the memory overhead mostly is comprised of subsampled measurements in free and occupied regions (up to 21MB) for performing multi-pass BGK inference. For OctoMap, the memory overhead is mostly comprised of pointers to a large number of voxels intersected by sensor rays from each depth image (up to 1MB).

In contrast, our GMMap requires very little memory overhead. Since SPGF* processes the depth image one pixel at a time in a single pass, only one (for single-core implementation) or four pixels (for multi-core implementation) are stored in memory at any time. Thus, the memory overhead associated with map construction is mostly comprised of compact line segments $\mathcal{S}$ generated from Scanline Segmentation in SPGF* and the local GMMap $\mathcal{G}_t$ generated at the output of Alg. 1. From Table III, the memory overhead of our single-core implementation is only 31KB to 106KB, which is at least 90% lower than other frameworks. Since four scanlines are segmented concurrently in our multi-core implementation, the memory overhead increases and ranges from 41KB to 128KB, which is at least 88% lower than other frameworks. However, our GPU implementation requires much larger memory overhead (around 24MB) due to the allocations of large GPU-accessible buffers for transferring four depth images and their Scanline Segmentation outputs to and from the GPU.

*3) **DRAM Access**:* We compare the average amount of DRAM access required for integrating each measurement (*i.e.*, pixel in the depth image) into the map among all frameworks. The amount of DRAM access correlates with the energy consumption of the DRAM and is computed by multiplying the number of last-level cache misses[6] with the size of the cache line. Recall that existing frameworks update the map by incrementally casting each measurement ray (more than 300,000 rays in a 640×480 depth image) into the current map. Since these rays diverge away from the sensor origin, memory

[5]Measured using our own memory profiler that automatically tracks memory allocation in the constructors and deconstructors of relevant C++ objects.

[6]Obtained by reading hardware counters of the CPU using the `perf_event_open()` system call on Linux. For GPU, the amount of DRAM accesses can be obtained directly using NVIDIA NSight Systems.

TABLE III
COMPARISON OF THE GMMAP AGAINST PRIOR WORKS USING THE NVIDIA JETSON TX2. ALL FRAMEWORKS ACHIEVE COMPARABLE ACCURACY.

Environment	Framework	Compute Resource (C = CPU core)	Throughput: Construction (images/s)	Throughput: Query* (10^6 locations/s)	Memory Footprint: Map Size (KB)	Memory Footprint: Overhead (KB)	Memory Footprint: DRAM Access (bytes/pixel)	Energy Consumption: CPU & GPU (mJ/image)	Energy Consumption: DRAM (mJ/image)	Energy Consumption: Total (mJ/image)
Room	GMMap	GPU & 4 C	**81**	0.79	**167**	24,563**	477	41	17	58
		4 C	60	0.79	176	41	27	**36**	**16**	**52**
		1 C	18	0.79	176	**31**	**14**	59	51	110
	NDT-OM	1 C	5.0	3.5	426	3,146	160	202	157	359
	BGKOctoMap-L	1 C	2.8	0.93	4,935	7,101	242	352	272	624
	OctoMap	1 C	3.6	**4.0**	2,190	629	164	298	209	507
Warehouse	GMMap	GPU & 4 C	**73**	0.52	**268**	24,596**	492	43	16	59
		4 C	58	0.51	269	56	30	**37**	**14**	**51**
		1 C	18	0.51	269	**41**	**20**	59	41	100
	NDT-OM	1 C	3.7	3.7	614	3,436	199	273	209	482
	BGKOctoMap-L	1 C	0.5	1.4	13,811	21,265	940	1,888	1,463	3,351
	OctoMap	1 C	4.3	**4.2**	1,590	606	143	256	176	433
Soulcity	GMMap	GPU & 4 C	**60**	0.46	850	24,740**	625	**56**	**23**	**79**
		4 C	31	0.47	**838**	128	76	73	25	98
		1 C	11	0.47	**838**	**106**	**44**	92	66	158
	NDT-OM	1 C	3.1	3.9	1,925	4,391	372	324	248	572
	BGKOctoMap-L	1 C	0.8	1.0	23,265	5,502	596	1,204	926	2,130
	OctoMap	1 C	2.1	**4.1**	10,452	1,068	644	485	373	858
Gascola	GMMap	GPU & 4 C	**44**	0.62	362	24,644**	1,048	**69**	36	105
		4 C	32	0.62	**361**	79	78	73	**29**	**102**
		1 C	11	0.62	**361**	**63**	**54**	97	81	178
	NDT-OM	1 C	2.6	**3.9**	1,339	4,392	358	383	291	674
	BGKOctoMap-L	1 C	0.4	1.1	16,736	9,993	899	2,407	1,840	4,248
	OctoMap	1 C	1.6	**3.9**	9,376	760	1,136	634	494	1,129

* Unlike other metrics, query throughput is computed using a single CPU core. ** High memory overhead due to the necessary allocation of large GPU-accessible buffers (used to store input images and output results of Scanline Segmentation) for concurrent processing of four images. Allocations of these buffers are not required for CPU-only implementations.

accesses along these rays often lack spatial and temporal locality for effective cache usage (especially if the map is too large to fit within on-chip caches). Thus, the single-core implementations of existing frameworks require a significant number of DRAM accesses ranging from 160 bytes to more than 1KB per pixel.

In contrast, our GMMap avoids ray casting by directly fusing Gaussians from a compact local map $\mathcal{G}_t$ with Gaussians from the previously observed region $\mathcal{C}_t$ in the global map $\mathcal{M}_{t-1}$ (see Fig. 6). Since both the local map $\mathcal{G}_t$ and the previously observed region $\mathcal{C}_t$ can be compactly cached, our single-core implementation reduces DRAM access by at least 85% (compared to existing frameworks) by accessing only 14 bytes to 54 bytes per pixel. Since multiple cores share the last-level cache, the number of cache misses increases for our multi-core implementation which requires slightly higher DRAM accesses ranging from 27 bytes to 78 bytes per pixel (at least 78% lower than existing frameworks). Our GPU implementation requires much larger DRAM accesses due to the higher amount of cache misses from the concurrent segmentation of all scanlines in four images. However, most DRAM accesses from our GPU implementation are coalesced (*i.e.*, multiple accesses can be serviced with a single transaction). Thus, the energy consumption of the DRAM slightly increases compared with our multi-core CPU implementation.

E. Energy Consumption

Table III summarizes the average energy consumption[7] per depth image during map construction. For all frameworks, the energy consumption of the DRAM is significant compared with that of CPU and GPU, which underscores the importance of reducing memory overhead and access. Due to computationally efficient single-pass GMM creation and fusion, our single-core implementation reduces the energy consumption of the CPU by at least 71% compared with other frameworks. By avoiding ray casting (and its associated DRAM accesses), our single-core implementation reduces the energy consumption of the DRAM by at least 68% compared with other frameworks. Our multi-core and GPU implementations are even more energy efficient. Recall that energy equals the product of power and latency. For both implementations, the decrease in average latency per image significantly outweighs the increase in power consumption. Thus, the energy consumption per image decreases compared with single-core implementation. In all, our CPU single-core, multi-core, and GPU implementations reduce the average energy consumption by at least 69%, 83%, and 84% compared with other frameworks, respectively.

VI. Conclusion

In this work, we proposed the GMMap that uses a compact Gaussian Mixture Model to accurately model the continuous distribution of occupancy in 3D environments. Occupancy probability is inferred with Gaussian Mixture Regression which is extended to retain unexplored regions. Across multiple indoor and outdoor environments, we analyzed GMMap's ability to accurately model free/occupied regions as well as boundaries from free-to-obstacle, and free-to-unexplored regions. Furthermore, we explained how to tune the parameters of the GMMap for controlling two fundamental trade-offs (*i.e.*, modeling of distant small objects *v.s.* compactness of the map, and the total extent of free regions *v.s.* extension of free regions into unexplored regions) to ensure safe navigation of robots.

Due to single-pass processing of depth images and directly operating on Gaussians, GMMap can be constructed in real-time on energy-constrained platforms while significantly reducing memory overhead and access. When benchmarked on the low-power NVIDIA Jetson TX2 platform across a diverse set of environments, GMMap can be constructed at a throughput of up to 60 images per second using the CPU and up to 81 images per second using the GPU, which is $4\times$ to $146\times$ higher than prior works. While achieving comparable accuracy as prior works, our CPU implementation of GMMap reduces map size by at least 56%, memory overhead by at least 88%, DRAM access by at least 78%, and energy consumption by at least 69%. Thus, to the best of our knowledge, GMMap not only enables real-time large-scale 3D mapping for energy-constrained robots for the first time but also illustrates the significance of memory-efficient algorithms for enabling low-power autonomy on these robots.

[7]Measured directly from the power monitors embedded on the Jetson TX2.

References

[1] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*. Springer, 2015, vol. 2077.

[2] Q. Tao, J. Wang, Z. Xu, T. X. Lin, Y. Yuan, and F. Zhang, "Swing-reducing flight control system for an underactuated indoor miniature autonomous blimp," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 4, pp. 1895–1904, 2021.

[3] Y. M. Chukewad, J. James, A. Singh, and S. Fuller, "Robofly: An insect-sized robot with simplified fabrication that is capable of flight, ground, and water surface locomotion," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2025–2040, 2021.

[4] R. Wood, R. Nagpal, and G.-Y. Wei, "Flight of the robobees," *Scientific American*, vol. 308, no. 3, pp. 60–65, 2013.

[5] S. H. Suhr, Y. S. Song, S. J. Lee, and M. Sitti, "Biologically inspired miniature water strider robot." in *Robotics: Science and Systems*, vol. 2005, 2005, pp. 319–326.

[6] M. Keennon, K. Klingebiel, and H. Won, "Development of the nano hummingbird: A tailless flapping wing micro air vehicle," in *50th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2012, p. 588.

[7] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[8] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997, pp. 146–151.

[9] Z. Zhang, T. Henderson, S. Karaman, and V. Sze, "Fsmi: Fast computation of shannon mutual information for information-theoretic mapping," *The International Journal of Robotics Research*, vol. 39, no. 9, pp. 1155–1177, 2020.

[10] T. Henderson, V. Sze, and S. Karaman, "An efficient and continuous approach to information-theoretic exploration," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8566–8572.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[12] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[13] 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," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, 2013.

[14] V. Guizilini and F. Ramos, "Towards real-time 3d continuous occupancy mapping using hilbert maps," *The International Journal of Robotics Research*, vol. 37, no. 6, pp. 566–584, 2018.

[15] S. Srivastava and N. Michael, "Efficient, multifidelity perceptual representations via hierarchical gaussian mixture models," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 248–260, 2018.

[16] B. Eckart, K. Kim, A. Troccoli, A. Kelly, and J. Kautz, "Accelerated generative models for 3d point cloud data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5497–5505.

[17] C. O'Meadhra, W. Tabib, and N. Michael, "Variable resolution occupancy mapping using gaussian mixture models," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2015–2022, 2018.

[18] A. Dhawale and N. Michael, "Efficient parametric multi-fidelity surface mapping," in *Robotics: Science and Systems (RSS)*, vol. 2, no. 3, 2020, p. 5.

[19] K. Goel, N. Michael, and W. Tabib, "Probabilistic point cloud modeling via self-organizing gaussian mixture models," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2526–2533, 2023.

[20] K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference," *IEEE Transactions on Robotics*, pp. 1–14, 2019. [Online]. Available: https://doi.org/10.1109/tro.2019.2912487

[21] P. Z. X. Li, S. Karaman, and V. Sze, "Memory-efficient gaussian fitting for depth images in real time," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8003–8009.

[22] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[23] N. Funk, J. Tarrio, S. Papatheodorou, M. Popović, P. F. Alcantarilla, and S. Leutenegger, "Multi-resolution 3d mapping with explicit free space representation for fast and accurate mobile robot motion planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3553–3560, 2021.

[24] D. Duberg and P. Jensfelt, "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6411–6418, 2020.

[25] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012.

[26] J. Wang and B. Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1003–1010.

[27] W. Zhi, L. Ott, R. Senanayake, and F. Ramos, "Continuous occupancy map fusion with fast bayesian hilbert maps," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4111–4117.

[28] Y. Gao and W. Dong, "An integrated hierarchical approach for real-time mapping with gaussian mixture model," *IEEE Robotics and Automation Letters*, 2023.

[29] F. Ramos and L. Ott, "Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[30] H. G. Sung, *Gaussian mixture regression and classification*. Rice University, 2004.

[31] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47–57.

[32] D. W. Scott and W. F. Szewczyk, "From kernels to mixtures," *Technometrics*, vol. 43, no. 3, pp. 323–335, 2001.

[33] M. Kristan and A. Leonardis, "Multivariate online kernel density estimation," in *Computer Vision Winter Workshop*, 2010, pp. 77–86.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

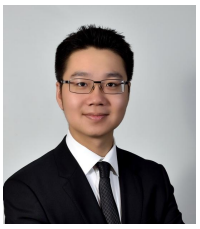
[35] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "Tartanair: A dataset to push the limits of visual slam," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4909–4916.

[36] "Jetson Download Center," NVIDIA Developer, available: https://developer.nvidia.com/jetson-tx2-nx-system-module-data-sheet.

[37] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[38] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

VII. Biography



Peter Zhi Xuan Li (Student Member, IEEE) received the B.A.Sc. in Engineering Science from the University of Toronto, Canada, in 2018. Between 2016 and 2017, he worked in the High-Speed Converters Group at Analog Devices, Toronto, as an integrated circuit engineer. His research focuses on the co-design of memory-efficient algorithms and specialized hardware for localization, mapping, and path-planning on energy-constrained devices such as AR/VR headsets, smartphones, and micro-robots.

Sertac Karaman (Member, IEEE) received the B.S. degrees in mechanical engineering and computer engineering from the Istanbul Technical University, Istanbul, Turkey, in 2007, the S.M. degree in mechanical engineering and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2009 and 2012, respectively. He is currently a Professor of Aeronautics and Astronautics at MIT. His research interests include the broad areas of robotics and control theory. In particular, he is focusing on the applications of probability theory, stochastic processes, stochastic geometry, formal methods, and optimization for the design and analysis of high-performance cyber-physical systems. The application areas of his research include driverless cars, unmanned aerial vehicles, distributed aerial surveillance systems, air traffic control, certification and verification of control systems software, and many others.

Dr. Karaman was the recipient of the IEEE Robotics and Automation Society Early Career Award, in 2017, the Office of Naval Research Young Investigator Award, in 2017, the Army Research Office Young Investigator Award, in 2015, the National Science Foundation Faculty Career Development (CAREER) Award, in 2014, the AIAA Wright Brothers Graduate Award, in 2012, and the NVIDIA Fellowship, in 2011.

Vivienne Sze (Senior Member, IEEE) received the B.A.Sc. (Hons) degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2004, and the S.M. and Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010 respectively. In 2011, she received the Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT.

She is an Associate Professor at MIT in the Electrical Engineering and Computer Science Department. Her research interests include computing systems that enable energy-efficient machine learning, computer vision, and video compression/processing for various applications, including autonomous navigation, digital health, and the Internet of Things. Prior to joining MIT, she was a Member of the Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designed low-power algorithms and architectures for video coding. She also represented TI in the Joint Collaborative Team on Video Coding (JCT-VC).

Dr. Sze was a recipient of the Air Force Young Investigator Research Program Award, the DARPA Young Faculty Award, the Edgerton Faculty Award, several faculty awards from Google, Facebook, and Qualcomm, the 2021 University of Toronto Engineering Mid-Career Achievement Award, and the 2020 ACM-W Rising Star Award, and a co-recipient of the 2018 Symposium on VLSI Circuits Best Student Paper Award, the 2017 CICC Outstanding Invited Paper Award, and the 2016 IEEE Micro Top Picks Award. She was a member of the JCT-VC team that received the Primetime Engineering Emmy Award for the development of the HEVC video compression standard.