Memory-Efficient Gaussian Fitting for Depth Images in Real Time

Memory-Efficient Gaussian Fitting for Depth Images in Real Time

Peter Zhi Xuan Li, Sertac Karaman, Vivienne Sze

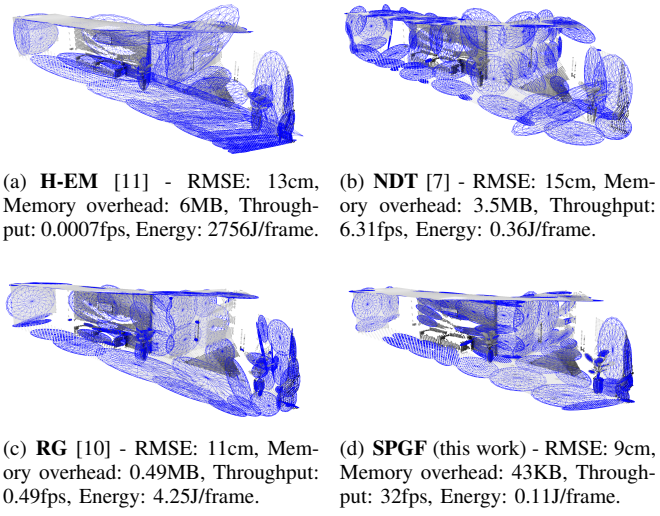
***Abstract*— Computing consumes a significant portion of energy in many robotics applications, especially the ones involving energy-constrained robots. In addition, memory access accounts for a significant portion of the computing energy. For mapping a 3D environment, prior approaches reduce the map size while incurring a large memory overhead used for storing sensor measurements and temporary variables during computation. In this work, we present a memory-efficient algorithm, named Single-Pass Gaussian Fitting (SPGF), that accurately constructs a compact Gaussian Mixture Model (GMM) which approximates measurements from a depthmap generated from a depth camera. By incrementally constructing the GMM one pixel at a time in a single pass through the depthmap, SPGF achieves higher throughput and orders-of-magnitude lower memory overhead than prior multi-pass approaches. By processing the depthmap row-by-row, SPGF exploits intrinsic properties of the camera to efficiently and accurately infer surface geometries, which leads to higher precision than prior approaches while maintaining the same compactness of the GMM. Using a low-power ARM Cortex-A57 CPU on the NVIDIA Jetson TX2 platform, SPGF operates at 32fps, requires 43KB of memory overhead, and consumes only 0.11J per frame (depthmap). Thus, SPGF enables real-time mapping of large 3D environments on energy-constrained robots.**

I. Introduction

Energy-constrained microrobots [1]–[3] have limited battery capacity, which limits the total amount of energy available for both actuation and computation. During computation, the energy cost of memory access can be quite significant. For instance, the energy cost of reading a 32-bit value from memory is more than performing a 32-bit multiplication [4]. The energy consumption of memory access increases with the size of memory and the distance of the memory from the processor. For instance, a CPU accessing data stored in a larger, off-chip memory such as DRAM (GBs of storage) requires orders-of-magnitude higher energy than smaller, on-chip (local) CPU caches (KBs to MBs of storage) [4]. In addition, lower-level L0 and L1 caches (a few KBs) require significantly lower energy to access than a L2 cache (a few MBs) [4]. Thus, algorithms designed for many robotics applications, especially the ones involving energy-constrained robots, should reduce memory overhead so that most data and variables used during computation can be stored in and accessed from lower-level caches.

Achieving memory efficiency is even more crucial for algorithms enabling 3D mapping on energy-constrained robots. During map construction, the memory usage is not limited to the storage of map itself, but also includes overheads for storing the sensor measurements and temporary variables. Mapping algorithms with significant memory overhead not only lead to higher energy consumption but also reduce the already limited memory available for map storage.

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. **Source code:** https://github.com/mit-lean/spgf



(a) **H-EM** [11] - RMSE: 13cm, Memory overhead: 6MB, Throughput: 0.0007fps, Energy: 2756J/frame.

(b) **NDT** [7] - RMSE: 15cm, Memory overhead: 3.5MB, Throughput: 6.31fps, Energy: 0.36J/frame.

(c) **RG** [10] - RMSE: 11cm, Memory overhead: 0.49MB, Throughput: 0.49fps, Energy: 4.25J/frame.

(d) **SPGF** (this work) - RMSE: 9cm, Memory overhead: 43KB, Throughput: 32fps, Energy: 0.11J/frame.

Fig. 1: Visualization of the GMMs (blue ellipsoids) constructed from a depthmap of a hallway from the TartanAir Office dataset [16]. Compared with prior approaches (a, b, c), SPGF (d) generates a more accurate GMM representation, requires significantly less memory overhead, executes in real time (*i.e.*, >30fps), and consumes much less energy using an ARM Cortex-A57 CPU on the NVIDIA Jetson TX2.

Since popular mapping frameworks such as the occupancy grid map [5] and OctoMap [6] are not sufficiently compact for storage on energy-constrained robots, recent works [7]–[11] focused on building compact maps comprised of Gaussian Mixture Models (GMMs) generated using measurements obtained from a depth camera, which enable pose estimation [12], [13] and path planning [14], [15]. Since a depth camera generates depthmaps (frames) containing millions of measurements per second, constructing GMMs that accurately approximate each depthmap in real time (*i.e.,* 30fps) is necessary. Recent works [7], [10], [11] only focused on reducing the time complexity for constructing GMMs. Since energy-constrained robots might use only a low-power CPU (*i.e.*, no GPU) for computation, these works not only are unable to operate in real time on a CPU alone but also require significant memory overhead from the *multi-pass* processing of the depthmap or its intermediate representations.

GMMs are typically constructed using the computationally demanding Expectation Maximization (EM) algorithm [17].

To reduce time complexity, Eckart et al. [11] introduced the Hierarchical EM (H-EM) that re-arranges computation using a tree. However, the H-EM applies an optimization procedure derived from the traditional EM to the entire depthmap at each level of the tree, which is neither computationally nor memory efficient (*i.e.*, storing an entire depthmap and a correspondence matrix in memory). Thus, a 140W NVIDIA GTX660 GPU is required for real-time operation.

Saarinen et al. [7] proposed the Normal Distance Transform (NDT) that partitions the minimum bounding box of the environment into voxels, and uses a Gaussian to represent all measurements within each voxel. The minimum bounding box depends on the geometries in the environment which are often unknown *a-priori*. Thus, the authors' NDT implementation[1] determines the minimum bounding box via an additional pass through the entire depthmap stored in memory, which increases memory overhead.

Recently, Dhawale et al. [10] proposed the Region Growing (RG) algorithm that starts by discretizing a depthmap using grids. An intermediate GMM representation is constructed using pixels within each grid. Finally, this intermediate representation is refined at different fidelities by iteratively merging Gaussians in neighboring grids that represent the same surface in the environment. However, the repeated refinement and storage of such intermediate representation greatly reduces the computational and memory efficiency.

In this work, we propose the Single-Pass Gaussian Fitting (SPGF) algorithm that accurately constructs a GMM from a depthmap in real time with orders-of-magnitude lower memory overhead than prior approaches [7], [10], [11]. The computational and memory efficiencies are enabled by the incremental updates of GMM parameters one depth pixel at a time without storing any previously visited pixels (*i.e.*, in a *single pass*). By processing the depthmap *row-by-row*, SPGF exploits the intrinsic properties of the camera to efficiently infer surface geometries so that the accuracy and compactness of the GMM are maintained. With similar number of Gaussians, SPGF achieves superior accuracy than prior multi-pass approaches. To our best knowledge, SPGF is the first algorithm that constructs GMM at 32fps, requires just 43KB of memory overhead, and consumes only 0.11J per frame using a low-power ARM Cortex-A57 CPU, as shown in Fig. 1. Thus, SPGF enables the real-time 3D mapping of large environments on energy-constrained robots.

II. Proposed Algorithm

In this section, we describe the Single-Pass Gaussian Fitting (SPGF) algorithm that constructs a GMM from the *scanlines* (rows) of a depthmap so that surface geometries can be inferred accurately and efficiently in a *single pass*. As described in Alg. 1, SPGF executes the following two procedures for each scanline:

1) **Scanline Segmentation** (Line 4): Partitions the pixels from each scanline into segments that represent planar surfaces with distinct orientations.

[1]https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic

Algorithm 1: Single-Pass Gaussian Fitting (SPGF)

Input: Depthmap D containing an array of scanlines $[L_0, L_1, \ldots, L_{V-1}]$
Output: A set of Gaussians G

```
1  function constructGMM(D)
2      G ← ∅, G_prev ← ∅
3      for ( v = 0; v < V; v = v + 1 ) {
4          S ← segmentScanline(D[v])
5          if v = 0 then
6              G_prev ← S
7          else
8              G_prev, G_comp ← fuseSegments(G_prev, S)
9              G ← G ∪ G_comp
10     G ← G ∪ G_prev
11     return G
```

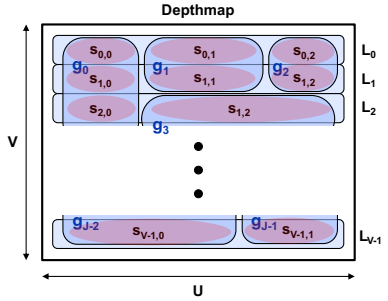


Fig. 2: Visualization of the notations used for describing the SPGF algorithm.

2) **Segment Fusion** (Line 8): Fuses segments that represent the same surface across adjacent scanlines into Gaussians.

Since Scanline Segmentation dominates the amount of computations in SPGF and can concurrently execute along multiple rows using multiple cores, the SPGF algorithm is highly parallelizable. The rest of this section is organized as follows. In Section II-A, we present preliminary concepts. Then, we describe Scanline Segmentation in Section II-B, and Segment Fusion in Section II-C.

A. Preliminaries

Let the depthmap generated by a depth camera have a width U and height V. Each pixel within the depthmap has a depth measurement denoted by $d_{(u,v)}$, where u and v are the pixel coordinates. Using the pinhole camera model, the depth measurement $d_{(u,v)}$ has an associated coordinate $\mathbf{p} = [p_x, p_y, p_z]$ in the ambient space $\mathbb{R}^3$ given by

$$\mathbf{p} = \mathbf{\Phi}^{-1}(u, v, d_{(u,v)}), \tag{1}$$

where $\mathbf{\Phi}^{-1}(\cdot)$ is the inverse projection function defined using the camera's intrinsic parameters. Note that the x-axis is defined along the width of the depthmap, and the z-axis is perpendicular to the depthmap.

The depthmap is partitioned into a set of V scanlines $\{L_0, L_1, \ldots, L_{V-1}\}$, as shown in Fig. 2. Scanline Segmentation partitions each scanline into a set of segments $S = \{s_{v,0}, s_{v,1}, \ldots\}$ such that each segment represents a

Algorithm 2: Scanline Segmentation

Input: An array of depth pixels along a scanline L_v
Output: A set of line segments S

1 **function** segmentScanline(L_v)
2 $S \leftarrow \varnothing$, $M \leftarrow \varnothing$
3 **for** ($u = 0$; $u < U$; $u = u + 1$) {
4 $d \leftarrow L_v[u]$
5 $\mathbf{p} \leftarrow \mathbf{\Phi}^{-1}(u, v, d)$
6 $x_\mathrm{t}, z_\mathrm{t} \leftarrow$ adaptiveThresh(d, a, b)
7 $b_\mathrm{merge} \leftarrow$ false
8 **foreach** $s \in M$ **do**
9 **if** b_merge = *false* **then**
10 **if** numPixels$(s) < t_\mathrm{fit}$ **then**
11 **if** distZ$(s, \mathbf{p}) < z_\mathrm{t}$ ***and*** distX$(s, \mathbf{p}) < x_\mathrm{t}$ **then**
12 $s \leftarrow$ addPoint$(s, \mathbf{p})$
13 $b_\mathrm{merge} \leftarrow$ true
14 **else**
15 $s \leftarrow$ occluded(s)
16 **else**
17 **if** distLine$(s, \mathbf{p}) < z_\mathrm{t}$ **then**
18 $s \leftarrow$ addPoint$(s, \mathbf{p})$
19 $b_\mathrm{merge} \leftarrow$ true
20 **else**
21 $s \leftarrow$ occluded(s)
22 **else**
23 $s \leftarrow$ occluded(s)
24 **if** isOccluded(s) ***and*** numOccludedPixels$(s) > t_\mathrm{occ}$ **then**
25 $S \leftarrow S \cup s$, $M \leftarrow M \setminus s$
26 **if** b_merge = *false* **then**
27 $k \leftarrow$ createNewSegment$(u, v, \mathbf{p})$
28 $M \leftarrow M \cup k$
29 **if** $|M| > \beta$ **then**
30 $e \leftarrow$ earliestSegment(M)
31 $S \leftarrow S \cup e$, $M \leftarrow M \setminus e$
32 $S \leftarrow S \cup M$
33 **return** S

planar surface with a distinct orientation. Then, Segment Fusion merge segments representing the same surface across adjacent scanlines into a set of Gaussians $G = \{g_0, g_1, \dots\}$ which forms the final GMM model $\mathcal{M}$ defined as:

$$\mathcal{M}(\mathbf{p}) = \sum_{i=0}^{J-1} w_i g_i(\mathbf{p} \mid \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}), \tag{2}$$

where w_i is the mixture weight of the Gaussian g_i with mean $\boldsymbol{\mu_i}$ and covariance $\boldsymbol{\Sigma_i}$.

B. Scanline Segmentation

As described in Alg. 2, Scanline Segmentation (SS) groups measurements within the scanline into segments representing distinct planar surfaces. To achieve memory efficiency, SS computes along the scanline one pixel at a time in a single pass so that only one depth measurement is stored in memory at any time. For each measurement p starting from the left of the scanline (Line 3), SS needs to decide whether p corresponds to a new surface or a segment s representing a

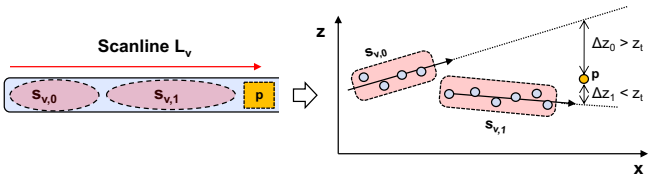
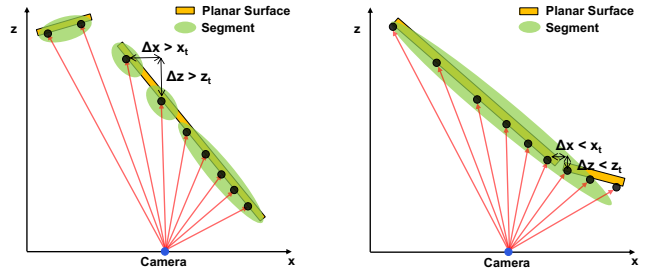


Fig. 3: Distances $\{\Delta z_0, \Delta z_1\}$ between measurement p and extrapolated lines from previous segments $\{s_{v,0}, s_{v,1}\}$ are computed. Since Δz_1 is less than the adaptive threshold z_t, measurement p should be merged with $s_{v,1}$.



(a) Using low thresholds (x_t, z_t) leads to the over-segmentation of measurements for a surface further away from the camera.

(b) Using high thresholds (x_t, z_t) leads to the under-segmentation of measurements for a surface closer to the camera.

Fig. 4: Undesirable results for using fixed thresholds (x_t, z_t) that are too low (a) or too high (b). Adaptive thresholds (defined in Eqn. (3)) that increase with the depth of each measurement are used to avoid under and over segmentation.

previously observed surfaces to the left of p (see Fig. 3). If p corresponds to segment s, parameters of s (mean, covariance, weight) can be incrementally updated with p as in [7].

Due to single-pass processing, SS cannot resolve inaccurate measurement-to-segment correspondences which are typically reduced with additional passes through the depthmap in prior works [10], [11]. These inaccurate correspondences are caused by unreliable estimates of the segment's parameters when such segment is initialized with a few noisy measurements (*i.e.*, less than a threshold t_fit). By the pinhole camera model, scanline measurements of the same planar surface form a *line* segment in the ambient space. Thus, SS exploits this property to efficiently reduce inaccurate correspondences for two cases: the number of measurements in s is 1) less than t_fit, and 2) equals to or greater than t_fit. Since the y coordinates for measurements within the same scanline do not vary significantly, SS is executed for the projection of the scanline on the xz plane.

Case 1 (Lines 11 to 15): If the line segment s is initialized with less than t_fit measurements, the parameters of the line cannot be reliably estimated. Instead of extrapolating the line towards p, the measurement p corresponds to s if the distances $(\Delta x, \Delta z)$ between p and the closest measurement in s are less than certain thresholds (x_t and z_t in Line 11). In prior work [10], these thresholds (x_t, z_t) are fixed hyperparameters, which could reduce the compactness (due to over-segmentation) or accuracy (due to under-segmentation) of the GMM (see Fig. 4 for an example). Because each

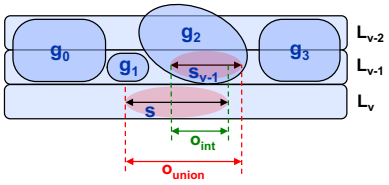


Fig. 5: Computation of the intersection to union ratio ($r = o_{\text{int}}/o_{\text{union}}$) between the number of pixels in segment s and segment s_{v-1} (contained within Gaussian g_2).

segment represents a *line*, SS is able to compute desirable thresholds that adapt to the depth of the measurement p, *i.e.*,

$$x_t = \frac{d^2}{fb}, \;\; z_t = ax_t, \tag{3}$$

where a and b are respectively the slope and z-intercept of the line formed from the intersection of the planar surface and the xz-plane, d is the value of the depth pixel for measurement p, and f is the focal length of the camera. The parameters a and b are set to predetermined values in Table II to accommodate a variety of surface orientations.

Case 2 (Lines 17 to 21): If the line segment s contains at least t_{fit} measurements, the parameters of the line are reliably estimated for extrapolation. Thus, as shown in Fig. 3, the measurement p corresponds with s if p is sufficiently close to the extrapolated line along the z direction (Lines 17). Processing along a scanline also allows SS to significantly reduce the amount of computation during parameter estimation. Since each segment s represents a line, the direction vector of s used for extrapolation can be incrementally updated with new measurements without re-computing from the eigen-decomposition of the covariance matrix of s (used in a prior approach [10]), which results in a $4\times$ higher throughput for SPGF on the ARM Cortex-A57 CPU.

Efficient extrapolation of line segments also allows SS to address the over-segmentation of surfaces caused by object occlusions (ignored in prior works to enhance throughput). The surface of a large object (*e.g.*, a wall) can be occluded by the surface of a smaller object (*e.g.*, a chair) that is closer to the camera. Thus, the segment that represents the occluded surface will encounter a temporary discontinuity along the scanline, which causes over-segmentation. To avoid such discontinuity, we select up to $\beta = 4$ closest segments (represented by a set M in Alg. 2) as candidates for determining the correspondence of each measurement p. A segment s representing a previously observed surface is transferred from M to the output set S if s does not correspond to t_{occ} consecutive new measurements (Lines 24 and 25).

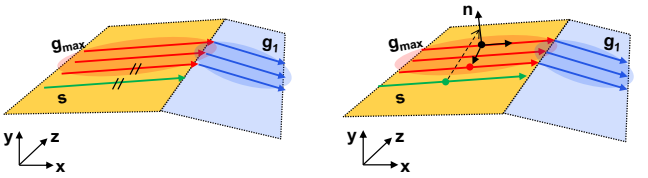
C. Segment Fusion

As described in Alg. 3, Segment Fusion (SF) fuses each line segment $s \in S$ (from scanline L_v) with a Gaussian $g_{\max} \in G_{\text{prev}}$ (from prior scanlines) if s corresponds to (*i.e.*, lies on) the same plane represented by $g_{\max}$. Thus, verifying segment-to-Gaussian correspondences dominates the amount of computations in SF. By incrementally constructing each

Algorithm 3: Segment Fusion

Input: A set of line segments S and incomplete Gaussians G_{prev} from previous scanlines
Output: A set of incomplete Gaussians G_{incomp} and completed Gaussians G_{comp}

1 **function** fuseSegments(G_{prev}, S)
2 $G_{\text{comp}} \leftarrow \varnothing$, $G_{\text{incomp}} \leftarrow \varnothing$
3 **foreach** $s \in S$ **do**
4 $g_{\max} \leftarrow$ maxIntersectionToUnion(s, G_{prev})
5 **if** cosineAngle($g_{\max}, s$) $> t_{\cos}$ ***and*** distPlane($g_{\max}, s$) $< n_{\min}$ **then**
6 $g_{\max} \leftarrow$ addSegment($g_{\max}, s$)
7 **else**
8 $G_{\text{incomp}} \leftarrow G_{\text{incomp}} \cup s$
9 **foreach** $g \in G_{\text{prev}}$ **do**
10 **if** notUpdated(g) **then**
11 $G_{\text{comp}} \leftarrow G_{\text{comp}} \cup g$
12 **else**
13 $G_{\text{incomp}} \leftarrow G_{\text{incomp}} \cup g$
14 **return** G_{incomp}, G_{comp}



(a) Criteria 1: The direction vector of the segment s should be parallel with the direction vector of s_{v-1} in the Gaussian $g_{\max}$.

(b) Criteria 2: The mean of the segment s should be sufficiently close to the plane represented by the Gaussian $g_{\max}$.

Fig. 6: Two criteria for verifying if a segment s corresponds to a planar Gaussian $g_{\max}$.

Gaussian using *line segments*, SF exploits geometric properties within each Gaussian so that each segment-to-Gaussian correspondence is verified more efficiently than the generic Gaussian-to-Gaussian correspondences from [10].

To reduce the amount correspondence verification for each segment $s \in S$, SF exploits the property of the depthmap to efficiently select the best candidate $g_{\max} \in G_{\text{prev}}$ that could correspond to s. Note that every Gaussian in the set G_{prev} contains a segment s_{v-1} from the previous scanline L_{v-1}. In the depthmap, measurements of the same surface appears in the same pixel region. Thus, in Line 4, the best candidate $g_{\max}$ is chosen to obtain the largest intersection to union ratio between pixels of segments s_{v-1} and s (see Fig. 5).

Once $g_{\max}$ is chosen, we verify if segment s corresponds to the same plane represented by $g_{\max}$. This verification is completed using two criteria in Line 5. Firstly, segment s should be sufficiently parallel to segment s_{v-1} in $g_{\max}$ (see Fig. 6(a)). Since both s and s_{v-1} are line segments, their parallelism is efficiently verified using a dot product of their direction vectors. Secondly, the mean of the segment s should be sufficiently close to the plane represented by $g_{\max}$, which requires the normal vector typically computed using eigen-

decomposition of the covariance matrix of $g_{\max}$ [10]. By constructing each Gaussian using *line segments*, SF easily obtains two planar vectors in $g_{\max}$ to efficiently compute its normal vector via a cross product. As shown in Fig. 6(b), one vector is derived using a line segment in $g_{\max}$. The other vector is derived using the means of all segments in $g_{\max}$. If eigen-decomposition were computed instead of cross product, the throughput of the SPGF algorithm would be reduced by around 10% on the ARM Cortex-A57 CPU.

If the segment s corresponds with $g_{\max}$, parameters of s (*i.e.,* mean, covariance, and weight) are fused with those from $g_{\max}$ using the technique in [7] (Line 6). Otherwise, segment s represents a newly observed surface and is appended to G_{incomp} (Line 8) which is the input to SF for the next scanline. If a Gaussian in G_{prev} does not correspond with any segment in S, such Gaussian is appended to G_{comp} (Line 11) which belongs to the final GMM G in Line 9 of Alg. 1.

III. Results

In this section, we compare the accuracy, throughput, memory overhead and energy consumption of the Single-Pass Gaussian Fitting (SPGF) algorithm against the following state-of-the-art algorithms: Hierarchical EM (H-EM) [11], Normal Distance Transform (NDT) [7] and Region Growing (RG) [10]. To emulate an energy-constrained setting, we obtained our results using *only* the ARM Cortex-A57 CPU on the NVIDIA Jetson TX2 platform. All algorithms were tested using both synthetic (TartanAir [16]) and real-world (TUM RGB-D [18]) datasets to emulate diverse environments and sensor properties (see Table I).

A. Accuracy of Representation

We are interested in how accurate GMMs generated from SPGF and existing approaches can model the measurements from a depthmap. For fairness, we chose the parameters such that the *average* number of Gaussians per depthmap generated across all algorithms is similar. For H-EM, we created a GMM tree with 4 levels and 4 Gaussians per node. For NDT, we chose a grid size of 2.0m and 0.5m for TartanAir Office and TUM Room dataset, respectively. For RG, we altered the threshold λ parameters based on the sensor properties and obtained all other parameters from [10]. The parameters used in SPGF are presented in Table II.

We reconstructed the original point cloud from each depthmap by randomly sampling the corresponding GMM. The Root Mean Squared Error (RMSE) associated with the *precision* of such reconstruction was calculated from each point in the reconstructed point cloud to the closest point in the original point cloud. The RMSE associated with the *recall* of such reconstruction was calculated in the opposite direction. A GMM with low precision error and high recall error means that each Gaussian models its corresponding surface accurately, but not all surfaces from the original point cloud are modeled.

TABLE I: Properties of the datasets used for evaluation.

Dataset	**TartanAir** [16]	**TUM RGB-D** [18]
Environment	Office	Freiburg1 Room
Dimensions	$37.23\text{m} \times 30.04\text{m} \times 6.32\text{m}$	$11.30\text{m} \times 11.94\text{m} \times 3.41\text{m}$
Depth Range	$0.37\text{m} \rightarrow 20.00\text{m}$	$0.47\text{m} \rightarrow 9.04\text{m}$
Depthmap Resolution	640×480	640×480
Number of Depthmaps	1395	1311

From Table III, SPGF achieves higher precision than prior approaches even though SPGF processes the depthmap in a single pass. Furthermore, SPGF adjusts the number of Gaussians in the GMM solely based on the complexity of environment so that the precision of the GMM representation is maintained for each depthmap (*i.e.*, lower standard deviation for precision RMSE). On the other hand, the number of Gaussians produced by other approaches is constrained by properties that increase throughput at the expense of precision, such as the number of nodes in the tree (H-EM), voxelization (NDT), and image-plane discretization (RG).

To eliminate the modeling of spurious measurements in SPGF, we pruned away Gaussians containing less than 200 measurements. Since small surfaces that are further away from the camera contain fewer measurements, SPGF occasionally treats these measurements as spurious which leads to a slightly larger recall error for the synthetic TartanAir Office dataset that contains measurements up to 20m. However, the recall error of SPGF is comparable with prior works in the TUM Room dataset where the range of the Kinect sensor is lower (up to 9m). A visualization of the GMMs generated from SPGF and prior approaches is shown in Fig. 1.

B. Throughput

Table III summarizes the throughput of the SPGF and prior approaches which were implemented and optimized similarly in C++. The throughputs for all algorithms are higher for the TUM Room dataset due to the presence of invalid measurements which were ignored during computation. Due to computationally efficient scanline processing, SPGF achieves superior throughput compared with prior multi-pass approaches. Using just one CPU core, the SPGF operates at 8fps for TartanAir Office dataset and 12fps for the TUM Room dataset, which are at least 32% faster than prior approaches. Using all four cores, SPGF is effectively

TABLE II: Parameters used in the SPGF algorithm.

Dataset	$n_{\min}$	β	a	b	t_{fit}	t_{occ}	$t_{\cos}$
TartanAir Office	0.05	4	6	1.43	16	10	0.5
TUM Room	0.08	4	6	0.42	16	10	0.5

TABLE III: Performance of SPGF vs. existing algorithms evaluated on the ARM Cortex-A57 CPU. The number of CPU cores is indicated in brackets.

Algorithm	**Precision RMSE (m)**	**Recall RMSE (m)**	**Number of Gaussians**	**Throughput (fps)**
TartanAir Office				
H-EM (1C)	0.13 ± 0.08	0.03 ± 0.04	50 ± 6	0.0007 ± 0.0002
NDT (1C)	0.15 ± 0.04	$\mathbf{0.03 \pm 0.01}$	65 ± 42	6.31 ± 0.13
RG (1C)	0.11 ± 0.04	0.03 ± 0.02	59 ± 23	0.49 ± 0.02
SPGF (1C)	$\mathbf{0.09 \pm 0.01}$	0.06 ± 0.05	59 ± 34	8.33 ± 0.11
SPGF (4C)				$\mathbf{32.31 \pm 1.03}$
TUM RGB-D Room				
H-EM (1C)	0.035 ± 0.014	0.008 ± 0.007	52 ± 5	0.0008 ± 0.0002
NDT (1C)	0.045 ± 0.008	0.009 ± 0.002	47 ± 26	8.37 ± 0.71
RG (1C)	0.043 ± 0.013	$\mathbf{0.008 \pm 0.003}$	52 ± 15	0.63 ± 0.04
SPGF (1C)	$\mathbf{0.033 \pm 0.004}$	0.012 ± 0.014	45 ± 16	11.57 ± 0.75
SPGF (4C)				$\mathbf{44.08 \pm 3.43}$

parallelized and executes in real time at 32fps and 44fps on the TartanAir Office and TUM Room datasets, respectively. Multi-core implementations of prior works are not publicly available. Even if these works can be parallelized, their throughputs are expected to be $4\times$ higher, which are still much lower than the multi-core implementation of SPGF.

C. Memory Overhead

In Fig. 7, we compare the *maximum* memory overhead for storing the input (*i.e.*, depth pixels) and temporary variables (only used during computation) in SPGF against prior works. Across all works, pixels in each depthmap are stored as float32 and most other variables are stored as float64.

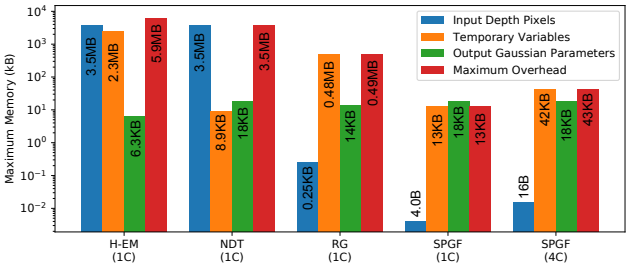
Since Scanline Segmentation in SPGF processes each row of the depthmap one pixel at a time in a single pass, SPGF only requires one depth pixel in memory (4B for one core) at any time, which is at least 98% lower than prior approaches. In particular, H-EM and NDT require the storage of all valid measurements from entire depthmap (up to 3.5MB) in memory due to the multi-pass optimization procedure (in H-EM) or the determination of minimum bounding box for the environment (in NDT). RG requires the storage of a subset of depthmap pixels (0.25KB) for initializing the Gaussians.

The amount of temporary variables in SPGF (up to 13KB for one core) is dictated by the maximum number of segments generated by Scanline Segmentation across all scanlines, which is at least 97% lower than H-EM and RG. In particular, the amount of temporary variables in H-EM (up to 2.3MB) is dictated by the pixel-to-Gaussian correspondence matrix used for optimizing GMM parameters. The amount of temporary variables for RG (up to 0.48MB) is dominated by a large intermediate representation of the depthmap used to refine the GMM parameters. The amount of temporary variables in NDT (up to 11KB) is slightly lower than SPGF and is dominated by a voxel-based data structure that partitions the minimum bounding box of the environment.

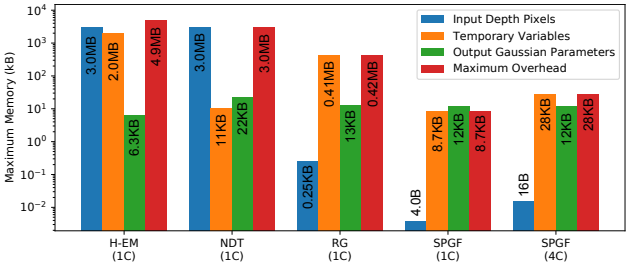
Since Scanline Segmentation for each row of depthmap can be executed independently on a CPU core, the memory overhead associated with input and temporary variables for our multi-core SPGF implementation scales with the number of cores. From Fig. 7, SPGF implemented with *four cores* requires up to 43KB of total memory overhead which is only $2.4\times$ higher than the maximum size of the output GMM (up to 18KB). In contrast, the *single-core* implementation for each prior work requires a total memory overhead that is $33\times$ to $959\times$ higher than the maximum size of the GMM.

D. Energy Consumption

Table IV summarizes the average energy consumption per frame (depthmap) for SPGF and prior approaches. Across all algorithms, the energy consumption of the DRAM is comparable to that of the CPU, which underscores the importance of minimizing memory overhead in addition to computation. Since SPGF is more computationally efficient, its single core implementation consumes at least 16% less CPU energy compared with prior approaches. From Fig. 7, the total memory overhead of SPGF's single core implementation



(a) TartanAir Office dataset [16]



(b) TUM RGB-D Room dataset [18]

Fig. 7: Maximum memory usage for the input depth pixels, temporary variables and output Gaussian parameters during the execution of SPGF and existing algorithms. The number of cores used in each algorithm is also annotated.

is at most 13KB which is less than the size of L1 data cache (32KB per core [19]) in the ARM Cortex-A57 CPU. Thus, SPGF seeks to reduce the amount of DRAM accesses. In fact, SPGF's single core implementation requires least 27% less DRAM energy than those from prior approaches. SPGF's multi-core implementation is more energy efficient as the power consumption of CPU and DRAM is amortized across four CPU cores. Using four cores, the total energy consumption of SPGF is only 0.11J/frame, which is at least 69% less than prior approaches across both datasets.

IV. Conclusion

In this work, we proposed the Single-Pass Gaussian Fitting (SPGF) algorithm that processes the depthmap *row-by-row* to construct a GMM while using only 43KB of memory. Due to its computational and memory efficiencies, the SPGF operates at 32fps on a low-power ARM CPU and consumes at least 69% less energy compared with prior approaches. Thus, SPGF finally enables large-scale 3D mapping for energy-constrained robots in real time. The superior performance of SPGF demonstrates the importance of memory-efficient algorithms for enabling autonomy on these robots.

TABLE IV: Average energy consumption per frame for SPGF vs. existing algorithms evaluated using the ARM Cortex-A57 CPU on the NVIDIA Jetson TX2 platform.

Algorithm	**TartanAir Office**			**TUM RGB-D Room**		
	CPU	DRAM	Total	CPU	DRAM	Total
H-EM (1C)	1529J	1227J	2756J	1402J	1125J	2527J
NDT (1C)	0.18J	0.18J	0.36J	0.14J	0.13J	0.27J
RG (1C)	2.36J	1.89J	4.25J	1.81J	1.45J	3.26J
SPGF (1C)	0.15J	0.13J	0.28J	0.11J	0.09J	0.20J
SPGF (4C)	**0.07J**	**0.04J**	**0.11J**	**0.05J**	**0.03J**	**0.08J**

REFERENCES

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

[2] 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*, p. 588, 2012.

[3] R. He, S. Sato, and M. Drela, "Design of single-motor nano aerial vehicle with a gearless torque-canceling mechanism," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, p. 1417, 2008.

[4] 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)*, pp. 10–14, 2014.

[5] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, no. 6, pp. 46–57, 1989.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] A. Dhawale and N. Michael, "Efficient parametric multi-fidelity surface mapping," in *Robotics: Science and Systems (RSS)*, vol. 2, p. 5, 2020.

[11] 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*, pp. 5497–5505, 2016.

[12] A. Dhawale, K. S. Shankar, and N. Michael, "Fast monte-carlo localization on aerial vehicles using approximate continuous belief representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5851–5859, 2018.

[13] W. Tabib, C. O'Meadhra, and N. Michael, "On-manifold gmm registration," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3805–3812, 2018.

[14] A. Dhawale, X. Yang, and N. Michael, "Reactive collision avoidance using real-time local gaussian mixture model maps," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3545–3550, IEEE, 2018.

[15] W. Tabib, K. Goel, J. Yao, M. Dabhi, C. Boirum, and N. Michael, "Real-time information-theoretic exploration with gaussian mixture model maps.," in *Robotics: Science and Systems*, 2019.

[16] 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)*, pp. 4909–4916, IEEE, 2020.

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[18] 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.

[19] "Jetson Download Center." Available: https://developer.nvidia.com/jetson-tx2-nx-system-module-data-sheet.