GIRA: Gaussian Mixture Models for Inference and Robot Autonomy

Kshitij Goel and Wennie Tabib

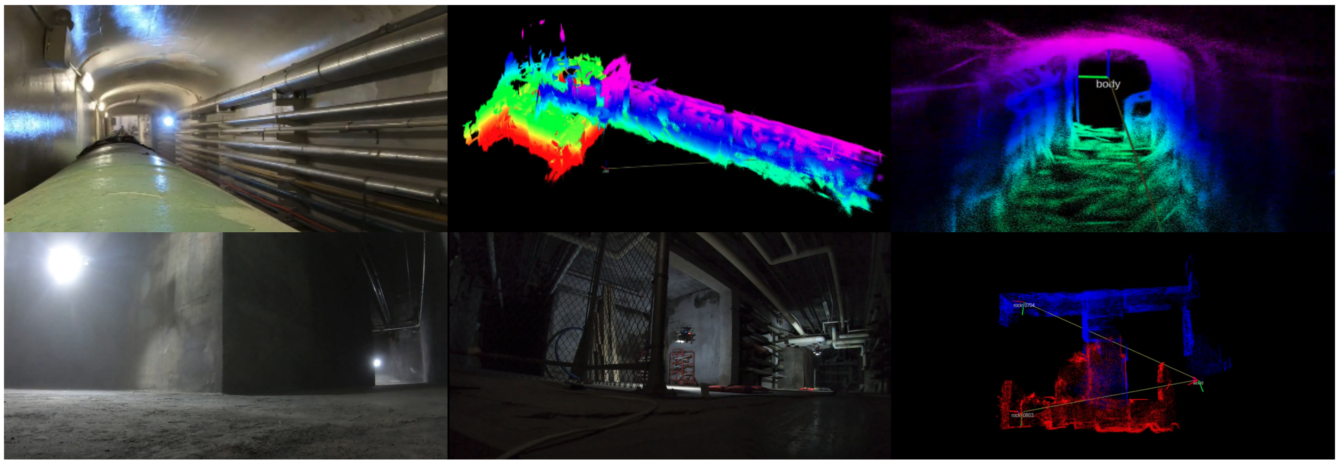


Fig. 1: GIRA has been deployed on size, weight, and power constrained aerial systems in real-world and unstructured environments. (Top left) A single aerial robot flies through an industrial tunnel and (top center) generates a high-fidelity Gaussian mixture model (GMM) map of the environment. (Top right) A close-up view of the reconstructed area around the robot. (Bottom left and bottom center) A team of two robots fly through a dark tunnel environment and produce a (bottom right) map, which is resampled from the underlying GMM and colored red or blue according to which robot took the observation. Videos of these experiments are available at: `https://youtu.be/qkbxfxgCoV0` and `https://youtu.be/t9iYd33oz3g`.

***Abstract*— This paper introduces the open-source framework, GIRA, which implements fundamental robotics algorithms for reconstruction, pose estimation, and occupancy modeling using compact generative models. Compactness enables *perception in the large* by ensuring that the perceptual models can be communicated through low-bandwidth channels during large-scale mobile robot deployments. The generative property enables *perception in the small* by providing high-resolution reconstruction capability. These properties address perception needs for diverse robotic applications, including multi-robot exploration and dexterous manipulation. State-of-the-art perception systems construct perceptual models via multiple disparate pipelines that reuse the same underlying sensor data, which leads to increased computation, redundancy, and complexity. GIRA bridges this gap by providing a unified perceptual modeling framework using Gaussian mixture models (GMMs) as well as a novel systems contribution, which consists of GPU-accelerated functions to learn GMMs 10-100x faster compared to existing CPU implementations. Because few GMM-based frameworks are open-sourced, this work seeks to accelerate innovation and broaden adoption of these techniques.**

I. Introduction

To navigate in and interact with the world, robots acquire, assimilate, and respond to sensor data. Models that enable perception in the large and small [1] are amenable to diverse robotics applications and have the potential to drastically increase robotic capabilities while addressing limitations in the way complex perception systems are developed today.

Recent large-scale robotic exploration deployments, like the DARPA Subterranean (Sub-T) Challenge [2], have highlighted the need for map compression to increase the exploration rate, an example of perception in the large, by facilitating information sharing. Further, state-of-the-art perception systems typically leverage separate concurrent perceptual processing pipelines, which increases computation, redundancy, and complexity [3]. For example, the highly sophisticated perception module of the NeBula system architecture [4] processes the same LiDAR data repeatedly (e.g., odometry, SLAM, terrain mapping, etc.), which is inefficient. Instead, what is needed is a unified framework for common perceptual processing elements, which is compact, generative, and amenable for deployment on low-power embedded systems [3].

Gaussian mixture models (GMMs) provide high-fidelity and communication-efficient point cloud modeling and inference [5] in real-world environments [6]. Recent works have demonstrated precise, high-fidelity representation of fine details required for perception in the small [7]. However, there are few open-source implementations, which poses a barrier to broad adoption by the general robotics community. To bridge this gap, this paper introduces GIRA, an open-source, unified framework (Fig. 1) for point cloud modeling, occupancy modeling, and pose estimation using GMMs based on [6, 8, 9]. In addition, GIRA includes a novel systems contribution, which consists of GPU-accelerated functions to learn GMMs 10-100x faster compared to existing CPU implementations. The software and associated datasets are open-sourced[1] to accelerate innovation and adoption of these

*The authors are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA (email: {`kshitij,wtabib`}`@cmu.edu`).

[1]Project webpage: `https://github.com/gira3d`

techniques.

II. Related Work

This section reviews open-source perception frameworks for compact, high-resolution point cloud modeling, pose estimation, and occupancy modeling for robotics applications. These works are compared and contrasted with GIRA.

The Normal Distributions Transform (NDT)[2] framework was introduced by Biber and Strasser [10] for scan registration and later extended to 3D registration [11] and occupancy modeling [12]. Goel et al. [7] demonstrate that NDTMap provides higher representation fidelity compared to Octomap[3] [13], but at the cost of increased disk storage requirements. While NDTMap provides distribution to distribution registration [14], Octomap does not provide analogous functionality. In contrast to these representations, GIRA provides higher memory-efficiency and surface reconstruction fidelity [7] as well as distribution to distribution registration [8, 9]. Further, NDTMap provides a CPU implementation, while GIRA provides both CPU and GPU implementations for multimodal environment modeling.

Oleynikova et al. [15] develop Voxblox, which uses Truncated Signed Distance Fields (TSDFs), for high-resolution reconstruction and occupancy mapping. The weights for the TSDFs are stored in a coarse fixed-resolution regular grid. Voxblox grows dynamically, but suffers from the same memory-efficiency limitation as the NDTMap. In contrast, the GIRA framework enables high-resolution surface reconstruction without a pre-specified size or a fixed-resolution discretization of the point cloud model. Like GIRA, Voxblox provides CPU[4] and GPU[5] implementations as well as a method to localize within the representation using submaps [16].

Duberg and Jensfelt [17] propose UFOMap, which improves upon Octomap by providing an explicit representation of unknown space and introduces Morton codes for faster tree traversal. An open-source CPU implementation of UFOMap[6] is available; however, the implementation does not provide functionality to localize within the map, like GIRA, Voxgraph, or NDTMap.

Vespa et al. [18] introduce the Supereight mapping framework, which consists of two dense mapping methods a TSDF-based implicit map and an explicit spatial occupancy map. In follow-on work [19], which leverages multi-resolution grids, the authors demonstrate that the TSDF-based method yields superior reconstruction compared to UFOMap. Supereight enables frame-to-model point cloud registration via Iterative-Closest-Point (ICP) alignment. However, Supereight uses RAM to assess memory efficiency, but does not provide statistics on space required to store the representation to disk. The CPU implementation is available for Supereight[7] as open-source software.

Reijgwart et al. [20] have recently proposed the Wavemap hierarchical volumetric representation, which uses wavelet compression for higher memory savings compared to Voxblox, Supereight, and Octomap. Like other discrete mapping methods, the highest resolution of the hierarchical map is set by the user and fixed during robot operation. In contrast, the SOGMM method in GIRA provides the ability to adapt the fidelity of the model according to the complexity of the scene, without utilizing a hierarchical approach [7]. The CPU implementation for Wavemap is available as open-source software[8] and includes some additional improvements (e.g., the use of OpenVDB [21] instead of the octree data structure used in the original paper). The approach, however, does not provide a method to estimate pose.

Doherty et al. [22] propose BGKOctomap, an extension to Octomap, which utilizes nonparametric Bayesian kernel inference for continuous-space occupancy modeling. The method is improved by modeling sensor rays as continuous free-space observations [23]. The CPU[9] implementations of these variants are available as open-source software.

Eckart et al. [24] present compact modeling of point clouds using GMMs to estimate pose. The approach is extended to a hierarchical formulation to improve memory-efficiency and accuracy [25]. Dhawale and Michael [26] present an alternative hierarchical approach, which equally weights the Guassian distributions, for high-resolution surface mapping. Srivastava and Michael [27] present another hierarchical approach but utilize the Expectation-Maximization algorithm to assign non-uniform weights depending on the local density of point cloud data. Common to these approaches is the need to set model complexity criteria such as image patch size [26], component splitting threshold [25], and model fidelity threshold [27]. Further, to the best of our knowledge, there are no open-source software implementations of existing GMM-based point cloud modeling methods. In contrast, we provide open-source CPU and GPU implementations of our work in [7], which leverages information-theoretic learning to adjust the model complexity. Finally, to demonstrate the occupancy modeling and pose estimation capabilities using GMM-based models, we provide CPU implementations based on [6, 8, 9].

III. Design

We envision the GIRA framework to be used in robotics research where 3D perception tasks must be executed in real-time and algorithms for these tasks should be easy to prototype.

Popular robotics software packages like Bullet [28], Drake [29], and TensorFlow [30] provide low-level programming language support for high-performance, real-time operation and high-level programming language bindings for

[2]https://github.com/OrebroUniversity/perception_oru
[3]https://github.com/Octomap/octomap
[4]https://github.com/ethz-asl/voxblox
[5]https://github.com/nvidia-isaac/nvblox
[6]https://github.com/UnknownFreeOccupied/ufomap
[7]https://bitbucket.org/smartroboticslab/supereight2
[8]https://github.com/ethz-asl/wavemap
[9]https://github.com/RobustFieldAutonomyLab/la3dm

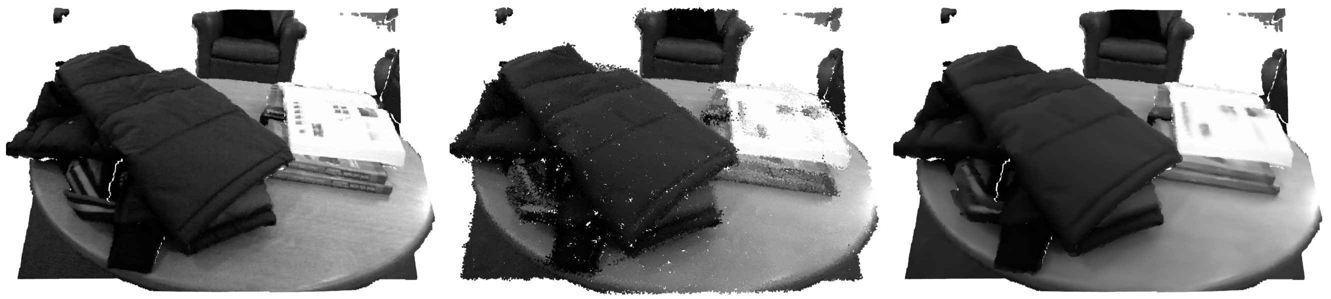

(a) Input (b) Resampled (c) Intensity Inference

Fig. 2: An example workflow for GIRA Reconstruction Section IV-A. The input is a depth-intensity point cloud shown in (a). The resulting model can be resampled to generate novel 4D points (b) or be used to infer expected intensity values at known 3D locations (c).

ease of prototyping. We follow the same model with GIRA where key algorithms are implemented in C/C++/CUDA with Python bindings. For C/C++, we use the C/C++17 standard. For GPU support, CUDA version 10.4 and above is required.

To enable message passing between different software systems, most robotics applications use the Robot Operating System (ROS) [31] and its successor ROS2 [32]. The GIRA framework is structured using Collective Construction (`colcon`) packages to help the robotics community easily integrate GIRA within their ROS and ROS2 workspaces. For low-level code and bindings, GIRA utilizes CMake for compilation support on both Linux and macOS. Python virtual environments are used to isolate executables.

For 3D perception tasks, visualization is an important capability for debugging research code. GIRA provides interfaces to the Open3D [33] visualization tools for this purpose. Furthermore, developers can leverage tools like RViz2 from ROS2 after integrating `colcon` packages from GIRA.

IV. GIRA Framework

The GIRA framework consists of three components: (1) GIRA Reconstruction, (2) GIRA Registration, and (3) GIRA Occupancy Modeling. This section provides an overview of these three components.

A. GIRA Reconstruction

Given time-synchronized depth and intensity images with known pose estimates, GIRA Reconstruction creates a Self-Organizing Gaussian Mixture Model (SOGMM) [7] that is:

1) **Continuous**, the point cloud is represented with a 4D GMM which is a linear combination of continuous functions (Gaussian distributions).
2) **Probabilistic**, the 4D GMM captures the variance and expected values for point locations and intensity values.
3) **Generative**, the 4D GMM enables fast sampling of point locations and intensity values from the model.
4) **Compact**, since the number of parameters required to represent the 4D GMM is much lower compared to the point cloud itself.
5) **Adaptive**, the number of Gaussian distributions within the 4D GMM are automatically estimated from the scene complexity of the underlying sensor data.

GIRA Reconstruction utilizes Open3D [33] for point cloud loading, writing, and visualization. We use NumPy [34] for interfacing with Eigen [35] via Pybind11 [36].

GIRA Reconstruction contains CPU and GPU implementations for SOGMM[10]. Both implementations can be accessed via a Python interface:

```
from sogmm_py.sogmm import SOGMM

# SOGMM of pointcloud on CPU
sg_cpu = SOGMM(bandwidth=0.015, compute='CPU')
mcpu = sg_cpu.fit(pointcloud)

# SOGMM of pointcloud on GPU
sg_gpu = SOGMM(bandwidth=0.015, compute='GPU')
mgpu = sg_gpu.fit(pointcloud)
```

where, `pointcloud` is a NumPy array and `bandwidth` is the bandwidth of the kernel used for the Gaussian Blurring Mean Shift (GBMS) within the SOGMM algorithm [7].

These models are continuous and generative. Three-dimensional points along with intensity values can be sampled from the model using:

```
# Sample 640*480 points from the model
rp = sg_gpu.joint_dist_sample(640*480)
```

A plot of the resampled point cloud is shown in Fig. 2b.

If the 3D point locations are known, the expected intensity values and variance can be inferred from the model at these locations:

```
# locs is a N x 3 numpy array
# E is N x 1 expected intensities
# V is N x 1 variance
_, E, V = mgpu.color_conditional(locs)
```

A plot of intensity values `E` is shown in Fig. 2c.

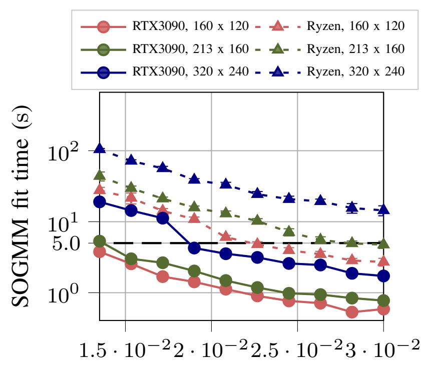
The SOGMM model is compact and its size can be computed as follows.

```
# computing memory usage
M = mgpu.n_components_

# 4 bytes per float
```

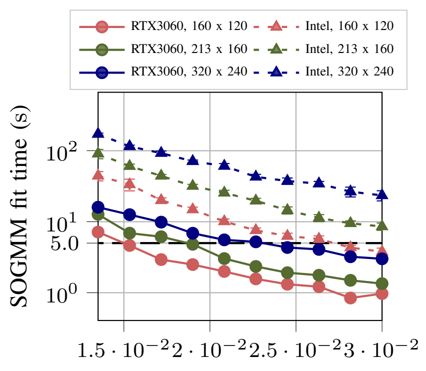
[10]Detailed tutorials are available at `https://gira3d.github.io/docs/index.html`.

Platform ID	Platform Type	CPU	GPU	Memory (CPU/GPU)
Ryzen/RTX3090	Desktop	AMD Ryzen Threadripper 3960X, 48 threads	NVIDIA GeForce RTX 3090	252 GB / 24 GB
Intel/RTX3060	Desktop	Intel Core i9-10900K, 20 threads	NVIDIA GeForce RTX 3060	32 GB / 12 GB
ARM-12c/Orin	Embedded	ARMv8 Processor rev 1 (v8l), 12 threads	NVIDIA Jetson Orin	32 GB
ARM-8c/Xavier	Embedded	ARMv8 Processor rev 0 (v8l), 8 threads	NVIDIA Jetson Xavier	16 GB
ARM-6c/TX2	Embedded	ARM Cortex-A57, 6 threads	NVIDIA Jetson TX2	8 GB

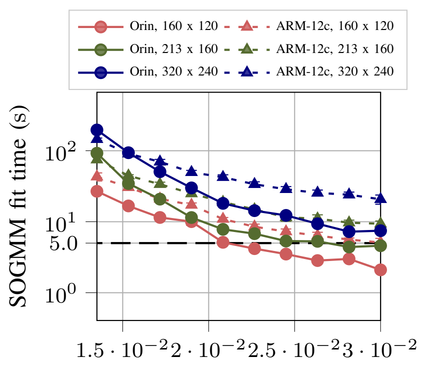
(a) Target Platforms



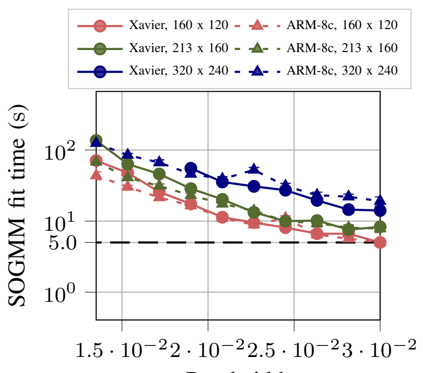
(b) Ryzen/RTX3090, SOGMM



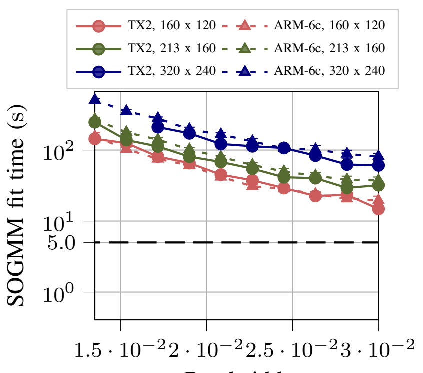
(c) Intel/RTX3060, SOGMM



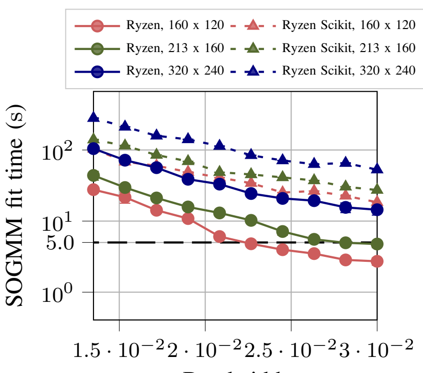
(d) ARM-12c/Orin, SOGMM



(e) ARM-8c/Xavier, SOGMM



(f) ARM-6c/TX2, SOGMM



(g) Comparison with `scikit-learn` on Ryzen

Fig. 3: Comparison of SOGMM computation time via GIRA Reconstruction on the target platforms listed in Fig. 3a. In (b) and (c) the GPU-accelerated case on the desktop platforms provides more than an order of magnitude improvement in timing compared to the CPU-only case for most image sizes. The results of the embedded platforms shown in (d), (e) and (f) demonstrate that the relative performance improvements seem to degrade with increasing SWaP constraints. In any case, (g) shows that our CPU implementation performs nearly an order of magnitude faster than a reference SOGMM implementation using `scikit-learn`.

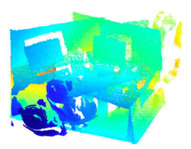
```
# 1 float value per weight
# 4 float values per mean
# 10 float values per covariance
mem_bytes = 4 * M * (1 + 10 + 4)
```

which is 69.78 kilobytes for the model learnt in Fig. 2.

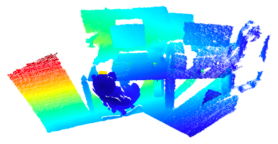
The time taken to learn a SOGMM is reported as a function of bandwidth parameter for a diverse set of platforms outlined in Fig. 3a. The input data for this experiment corresponds to frame 854, which was randomly selected, of the simulated `livingroom1` data from the Augmented ICL-NUIM datasets [37]. Ten equally spaced bandwidth values from 0.0135 to 0.0300 are used. Image sizes of 320×240, 213×160, and 160×120 are used (corresponding to $2\times$, $3\times$, and $4\times$ reduction along each axis of the original 640×480 image). Because there is randomness in the KInit step, each case is run ten times and averaged to obtain accurate timing results.

Figures 3b to 3f plot the results of these trials for the CPU-only (dashed lines with triangle markers) and GPU-accelerated (solid lines with circle markers) implementations. The y-axes of these plots use a base-10 log-scale and the observed standard deviation, which is plotted as error bars, is very low compared to the mean values. From Figs. 3b and 3c we observe over an order of magnitude faster performance when using the GPU-accelerated version of the system for all image sizes. Further, there is an overall decrease in performance from Ryzen/RTX3090 (Fig. 3b) to Intel/RTX3060 (Fig. 3c) for both CPU-only and GPU-accelerated versions. This is expected due to the decrease in computational capability for both the CPU and GPU.

Figure 3d provides results for ARM-12c/Orin platform. In this case, the gains for the GPU-accelerated version are lower than the desktop platforms. Notice that for image size



(a) Before registration



(b) After registration

Fig. 4: The point clouds in (a) are originally misaligned. (b) The code in Section IV-B estimates the SE(3) transform to align them.

320 × 240 the CPU starts performing better than GPU at low bandwidths. At low bandwidths, the number of estimated components are high.

Figures 3e and 3f suggest a further decrease in relative performance improvement in using the GPU-accelerated version as opposed to the CPU-only version of our system. Further, due to memory constraints the 320 × 240 image size fails for both platforms below certain bandwidths. Both ARM-8c/Xavier and ARM-6c/TX2 are SWaP-constrained platforms used on robots. For real-world usage of our framework, we recommend using the CPU-only version when CPU resources are not required by other subsystems (e.g., planning, control, and visual-inertial odometry) and using the GPU-accelerated version when CPU resources are in demand (which is often the case).

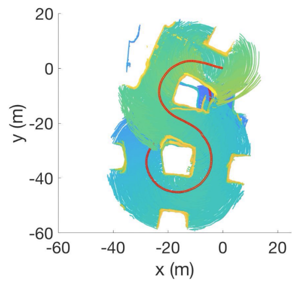
B. GIRA Registration

This module implements (1) registering a pair of GMMs [8] and (2) closing the loop using a pose graph optimization [9].

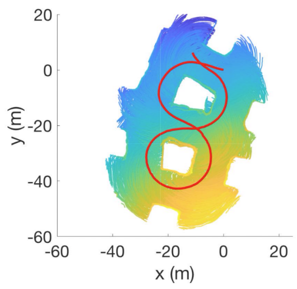
The *anisotropic*, *isoplanar*, and *isoplanar-hybrid* registration variants from [8] are implemented in this module. Python and MATLAB interfaces have been developed, but this document provide examples only for the Python interface. The isoplanar-hybrid registration approach first calls a coarse optimizing using the isoplanar registration function followed by a refinement optimization using the anisotropic registration. The source and target variables are paths to files containing GMMs.

```
from gmm_d2d_registration_py import
    isoplanar_registration
from gmm_d2d_registration_py import
    anisotropic_registration

# Initial registration guess
Tinit = np.eye(4)

# Isoplanar registration
Tiso = isoplanar_registration(Tinit, source,
    target)
Tout = anisotropic_registration(Tiso, source,
    target)

# Rotation and translation solutions
R = Tout[0:3, 0:3]
t = Tout[0:3, 3]
```

The result of registering a single pair of images may be seen in Fig. 4. In addition, a pose graph optimization example is provided, which uses GTSAM [38]. A comparison of the frame-to-frame registration with and without loop closure is shown in Fig. 5.



(a) Without loop closure



(b) Loop closure

Fig. 5: The trajectories reconstructed using (a) frame-to-frame registration and (b) with loop closure is enabled are shown with the pointclouds plotted.

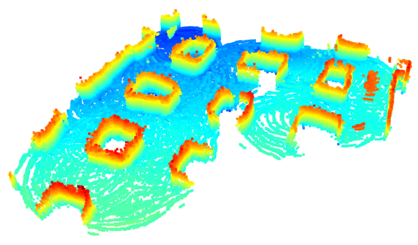


Fig. 6: Resampled points from a GMM are added to an occupancy grid map and the occupied voxels are queried and visualized.

C. GIRA Occupancy Modeling

This module implements occupancy reconstruction by sampling from a GMM and raytracing through an occupancy grid map. MATLAB and Python interfaces are provided, but only the Python interface is discussed in this document[11]. Like the registration module detailed in Section IV-B, this module is compatible with scikit-learn [39] GMMs and assumes GMMs are loaded from file.

```
# Create 3d occupancy grid with parameters p
grid = Grid3D(p)

# Nx3 sampled from GMM (assumed in world frame)
pts = gmm.sample(num_pts)

# Add the points to the grid
for i in range(0, num_pts):
  ray_end = Point(pts[i,0], pts[i,1], pts[i,2])

  # sensor_pose is in world frame
  # TRIMMED_MAX_RANGE set by user
  grid.add_ray(sensor_pose, ray_end,
      TRIMMED_MAX_RANGE)
```

Functions for querying occupied, free, and unknown voxels are provided through Python and MATLAB bindings of C++ code. The result of adding sampled points from the Mine dataset GMMs and querying the occupied voxels is shown in Fig. 6.

[11]Detailed documentation for both MATLAB and Python is provided at `https://gira3d.github.io/docs/index.html`.

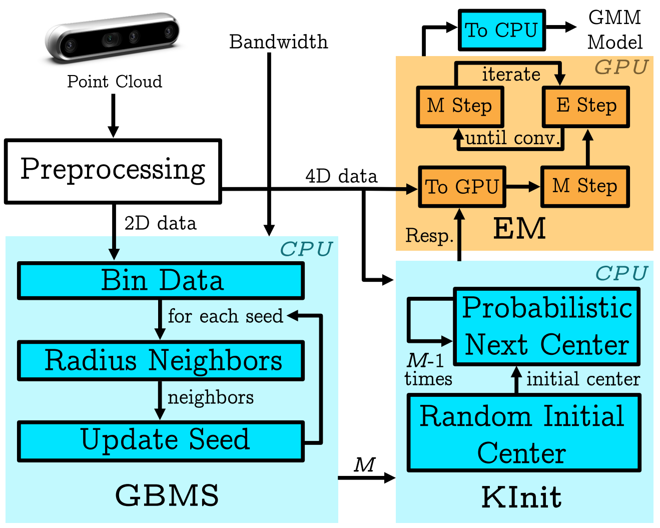


Fig. 7: Information flow for the GPU-accelerated adaptive point cloud modeling system. Given a bandwidth parameter and depth-intensity image pair, the Gaussian Blurring Mean Shift (GBMS) obtains the number of components $|\mathcal{B}|$. The number of components and the 4D data are used by KInit to calculate the responsibility matrix used by the EM algorithm. The result of the EM algorithm is the SOGMM model [7].

V. Implementation Details

This section details the GPU-accelerated software architecture of GIRA Reconstruction. Figure 7 provides an overview of the accelerated SOGMM components [7].

a) Gaussian Blurring Mean Shift: Comaniciu and Meer [40] leverage a binned estimator to determine seeds for the algorithm. A kd-tree [41] is used to query the neighbors in $\mathcal{Y}$. The points within the specified radius are averaged and the seed is updated to the new location. The algorithm terminates when either the number of maximum iterations is reached or there is no substantial change with respect to the previous seed position.

b) Expectation Maximization: The EM algorithm consists of the Expectation (E) and Maximization (M) steps. The E Step evaluates the responsibilities γ_{nb} using the current parameters $\boldsymbol{\mu}_b$, $\boldsymbol{\Sigma}_b$ and π_b via

$$\gamma_{nb} = \frac{\pi_b \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)}{\sum_{a=1}^{|\mathcal{B}|} \pi_a \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a)}. \tag{1}$$

To reduce the computational complexity of Eq. (1), the natural logarithm can be applied to convert the multiplications and divisions into sums and differences:

$$\begin{aligned}\ln \gamma_{nb} &= \ln \pi_b + \ln\left(\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)\right) \\ &\quad - \ln\left(\sum_{a=1}^{|\mathcal{B}|} \pi_a \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a)\right).\end{aligned} \tag{2}$$

Term 2 of Eq. (2) may be rewritten, as derived in [42–44],

$$\begin{aligned}&\ln\left(\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)\right) \\ &= -\frac{1}{2}\left(D\ln(2\pi) + \sum_{j=1}^{D} (\mathrm{P}_b(\mathbf{x}_n - \boldsymbol{\mu}_b))_j^2\right) + \left(\sum_{j=1}^{D} \ln\left(\mathrm{diag}(\mathrm{P}_b)\right)_j\right)\end{aligned} \tag{3}$$

where $\boldsymbol{\Sigma} = \mathrm{L}\mathrm{L}^\top$, $\mathrm{P} = \mathrm{L}^{-1}$, and L is a lower triangular matrix calculated using the Cholesky decomposition of the covariance matrix. Summing the logarithm of the diagonal entries of P_b (i.e., $\ln(\mathrm{diag}(\mathrm{P}_b))$) in Eq. (3) is equivalent to $\ln|\boldsymbol{\Sigma}_b|^{-1/2}$.

The GPU implementation leverages higher-order tensor representations (rank-3 and rank-4 tensors)[12]. The weights are represented as a rank-3 tensor of shape $(1, |\mathcal{B}|, 1)$, means are represented as a rank-3 tensor of shape $(1, |\mathcal{B}|, 4)$, and covariances are represented as a rank-4 tensor of shape $(1, |\mathcal{B}|, 4, 4)$. This implementation accelerates unary (e.g., logarithm and exponential of a matrix, reduction operations like summing along a dimension or taking a maximum along a dimension of a rank-2 or a rank-3 tensor) and binary (e.g., addition, subtraction, multiplication, and division of rank-2 and rank-3 tensors) operations via element-wise CUDA kernels with fixed block and grid sizes for all GPUs.

Rank-2 tensor multiplication is accelerated using the `cuBLAS`[13] `gemm` routine. Rank-3 tensor multiplication is accelerated via the `cuBLAS gemmStridedBatched` routine. The Cholesky decomposition of a rank-2 tensor is accelerated via the `cuSOLVER`[14] `potrf` routine. The Cholesky decomposition of a rank-3 tensor is accelerated using the `cuSOLVER potrfBatched` routine. Using the Cholesky decomposition, a linear system of equations involving rank-2 tensors is solved using the `cuSOLVER potrs` routine and for rank-3 tensors using the `potrsBatched` routine.

VI. Conclusion

GIRA is a set of tools and software for processing point cloud data into Gaussian mixture models for inference and robot autonomy. These tools and software are released open-source `https://github.com/gira3d`. Fundamental robotics capabilities from our prior works on point cloud modeling [7], pose estimation [8, 9], and occupancy modeling [6] are included in the open-source release. These fundamental capabilities have applications beyond exploration and aerial robotics. The adaptivity of the SOGMM representation has applicability to perception in the small and fine grained manipulation tasks. The variable resolution occupancy grid mapping and distribution to distribution registration software may be leveraged for high-speed mobile robot applications like off-road operations. By releasing this software, the authors hope to increase the accessibility of these formulations to technical experts.

References

[1] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, no. 2, pp. 177–196, 2018.

[2] T. H. Chung, V. Orekhov, and A. Maio, "Into the robotic depths: Analysis and insights from the darpa subterranean challenge," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 6, 2022.

[3] B. Eckart, "Compact generative models of point cloud data for 3d perception," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, October 2017.

[12]For the exposition of the GPU-accelerated components, tensor conventions from TensorFlow [30] are used.

[13]`cuBLAS https://docs.nvidia.com/cuda/cublas`

[14]`cuSOLVER https://docs.nvidia.com/cuda/cusolver`

[4] A. Agha, K. Otsu, B. Morrell *et al.*, "NeBula: TEAM CoSTAR's Robotic Autonomy Solution that Won Phase II of DARPA Subterranean Challenge," *FR*, vol. 2, no. 1, pp. 1432–1506, Mar. 2022.

[5] M. Corah, C. O'Meadhra, K. Goel *et al.*, "Communication-Efficient Planning and Mapping for Multi-Robot Exploration in Large Environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1715–1721, Apr. 2019.

[6] W. Tabib, K. Goel, J. Yao *et al.*, "Autonomous Cave Surveying With an Aerial Robot," *IEEE Trans. Robot.*, pp. 1–17, 2021.

[7] K. Goel, N. Michael, and W. Tabib, "Probabilistic Point Cloud Modeling via Self-Organizing Gaussian Mixture Models," *IEEE Robot. Autom. Lett.*, vol. 8, no. 5, pp. 2526–2533, May 2023.

[8] W. Tabib, C. O'Meadhra, and N. Michael, "On-Manifold GMM Registration," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3805–3812, Oct. 2018.

[9] W. Tabib and N. Michael, "Simultaneous Localization and Mapping of Subterranean Voids with Gaussian Mixture Models," in *Field Serv. Robot.*, ser. Springer Proceedings in Advanced Robotics, G. Ishigami and K. Yoshida, Eds. Singapore: Springer, 2021, pp. 173–187.

[10] P. Biber and W. Strasser, "The normal distributions transform: A new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, Oct. 2003, pp. 2743–2748 vol.3. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1249285

[11] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3D-NDT," *J. Field Robot.*, vol. 24, no. 10, pp. 803–827, 2007.

[12] J. Saarinen, H. Andreasson, T. Stoyanov *et al.*, "Normal Distributions Transform Occupancy Maps: Application to large-scale online 3D mapping," in *2013 IEEE Int. Conf. Robot. Autom.*, May 2013, pp. 2233–2238.

[13] A. Hornung, K. M. Wurm, M. Bennewitz *et al.*, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013.

[14] T. Stoyanov, M. Magnusson, H. Andreasson *et al.*, "Fast and accurate scan registration through minimization of the distance between compact 3d ndt representations," *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1377–1393, 2012.

[15] H. Oleynikova, Z. Taylor, M. Fehr *et al.*, "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning," in *2017 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*, Sep. 2017, pp. 1366–1373.

[16] V. Reijgwart, A. Millane, H. Oleynikova *et al.*, "Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps," *IEEE Robot. Autom. Lett.*, vol. 5, no. 1, pp. 227–234, Jan. 2020.

[17] D. Duberg and P. Jensfelt, "UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6411–6418, Oct. 2020.

[18] E. Vespa, N. Nikolov, M. Grimm *et al.*, "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1144–1151, Apr. 2018.

[19] N. Funk, J. Tarrio, S. Papatheodorou *et al.*, "Multi-Resolution 3D Mapping With Explicit Free Space Representation for Fast and Accurate Mobile Robot Motion Planning," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3553–3560, Apr. 2021.

[20] V. Reijgwart, C. Cadena, R. Siegwart *et al.*, "Efficient volumetric mapping of multi-scale environments using wavelet-based compression," in *Robotics: Science and Systems XIX*, vol. 19, Jul. 2023.

[21] K. Museth, J. Lait, J. Johanson *et al.*, "OpenVDB: An open-source data structure and toolkit for high-resolution volumes," in *ACM SIGGRAPH 2013 Courses*, ser. SIGGRAPH '13. New York, NY, USA: Association for Computing Machinery, Jul. 2013, p. 1.

[22] K. Doherty, J. Wang, and B. Englot, "Bayesian generalized kernel inference for occupancy map prediction," in *2017 IEEE Int. Conf. Robot. Autom. ICRA*, May 2017, pp. 3118–3124.

[23] K. Doherty, T. Shan, J. Wang *et al.*, "Learning-Aided 3-D Occupancy Mapping With Bayesian Generalized Kernel Inference," *IEEE Trans. Robot.*, vol. 35, no. 4, pp. 953–966, Aug. 2019.

[24] B. Eckart, K. Kim, A. Troccoli *et al.*, "Accelerated Generative Models for 3D Point Cloud Data," in *2016 IEEE Conf. Comput. Vis. Pattern Recognit. CVPR*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 5497–5505.

[25] B. Eckart, K. Kim, and J. Kautz, "HGMR: Hierarchical Gaussian Mixtures for Adaptive 3D Registration," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 705–721.

[26] A. Dhawale and N. Michael, "Efficient Parametric Multi-Fidelity Surface Mapping," in *Robot. Sci. Syst. XVI*. Robotics: Science and Systems Foundation, Jul. 2020.

[27] S. Srivastava and N. Michael, "Efficient, Multifidelity Perceptual Representations via Hierarchical Gaussian Mixture Models," *IEEE Trans. Robot.*, vol. 35, no. 1, pp. 248–260, Feb. 2019.

[28] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[29] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[30] M. Abadi, P. Barham, J. Chen *et al.*, "Tensorflow: a system for large-scale machine learning." in *Osdi*, vol. 16, no. 2016. Savannah, GA, USA, 2016, pp. 265–283.

[31] M. Quigley, B. Gerkey, K. Conley *et al.*, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[32] S. Macenski, T. Foote, B. Gerkey *et al.*, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[33] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," Jan. 2018.

[34] C. R. Harris, K. J. Millman, S. J. van der Walt *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[35] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[36] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – seamless operability between c++11 and python," 2017, https://github.com/pybind/pybind11.

[37] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust Reconstruction of Indoor Scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5556–5565.

[38] F. Dellaert, V. Agrawal, A. Jain *et al.*, "Gtsam," *URL: https://borg. cc. gatech. edu*, 2012.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[40] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[41] J. L. Blanco and P. K. Rai, "nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees," https://github.com/jlblancoc/nanoflann, 2014.

[42] L. Buitinck, G. Louppe, M. Blondel *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[44] W. Tabib, "Approximate Continuous Belief Distributions for Exploration," CMU-CS-TR-19-108, Carnegie Mellon University, Pittsburgh, PA, USA, May 2019.