

LOCUS 2.0: Robust and Computationally Efficient Lidar Odometry for Real-Time 3D Mapping

Andrzej Reinke^{1,2}, Matteo Palieri^{1,3}, Benjamin Morrell¹, Yun Chang⁴, Kamak Ebadi¹, Luca Carbone⁴, Ali-akbar Agha-mohammadi¹

Abstract—Lidar odometry has attracted considerable attention as a robust localization method for autonomous robots operating in complex GNSS-denied environments. However, achieving reliable and efficient performance on heterogeneous platforms in large-scale environments remains an open challenge due to the limitations of onboard computation and memory resources needed for autonomous operation. In this work, we present LOCUS 2.0, a robust and computationally-efficient lidar odometry system for real-time underground 3D mapping. LOCUS 2.0 includes a novel normals-based Generalized Iterative Closest Point (GICP) formulation that reduces the computation time of point cloud alignment, an adaptive voxel grid filter that maintains the desired computation load regardless of the environment’s geometry, and a sliding-window map approach that bounds the memory consumption. The proposed approach is shown to be suitable to be deployed on heterogeneous robotic platforms involved in large-scale explorations under severe computation and memory constraints. We demonstrate LOCUS 2.0, a key element of the CoSTAR team’s entry in the DARPA Subterranean Challenge, across various underground scenarios.

We release LOCUS 2.0 as an open-source library and also release a lidar-based odometry dataset in challenging and large-scale underground environments. The dataset features legged and wheeled platforms in multiple environments including fog, dust, darkness, and geometrically degenerate surroundings with a total of 11 h of operations and 16 km of distance traveled.

Index Terms—SLAM, Data Sets for SLAM, Robotics in Under-Resourced Settings, Sensor Fusion

I. INTRODUCTION

LIDAR odometry has emerged as a key tool for robust localization of autonomous robots operating in complex GNSS-denied environments. Lidar sensors do not rely on external light sources and provide accurate long-range 3D measurements by emitting pulsed light waves to estimate the range to surrounding obstacles, through time-of-flight-based techniques. For these reasons, lidar has been often

Manuscript received: February, 24, 2022; Revised May, 13, 2022; Accepted May, 19, 2022.

This paper was recommended for publication by Editor Javier Civera upon evaluation of the Associate Editor and Reviewers’ comments.

This work was supported by the Jet Propulsion Laboratory - California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004). This work was partially funded by the Defense Advanced Research Projects Agency (DARPA). ©2022 All rights reserved.

¹Reinke, Palieri, Morrell, Ebadi and Agha-mohammadi are with NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA benjamin.morrell@jpl.nasa.gov

²Reinke is with University of Bonn, Germany arein@uni-bonn.de

³Palieri is with the Department of Electrical And Information Engineering, Polytechnic University of Bari, IT matteo.palieri@poliba.it

⁴Chang and Carbone are with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, USA lcarbone@mit.edu

Digital Object Identifier (DOI): see top of this page.

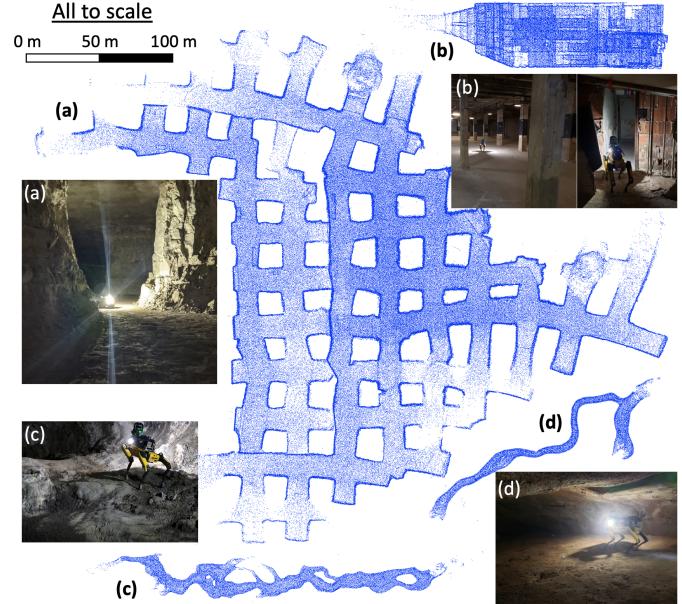


Fig. 1: Four examples from our large underground lidar-based SLAM dataset consisting of over 16 km distance traveled and 11 h of operation across diverse environments: (a) a large-scale Limestone Mine (Kentucky Underground), (b) A 3-level urban environment with both large, open spaces and tight passages (LA Subway), (c) lava tubes with large vertical changes, and (d) lava tubes with narrow passages. LOCUS 2.0 performed successfully in all these environments on computationally constrained robots.

preferred over visual sensors to achieve reliable ego-motion estimation in cluttered environments with significant illumination variations (e.g., search, rescue, industrial inspection and underground exploration).

Lidar odometry algorithms aim to recover the robot’s motion between consecutive lidar acquisitions using scan registration. Through repeated observations of fixed environmental features, the robot can simultaneously estimate its movement, construct a map of the unknown environment, and use this map to keep track of its position within it.

While many lidar odometry algorithms can achieve remarkable accuracy, their computational cost can still be prohibitive for computationally-constrained platforms, reducing their field of applicability in systems of heterogeneous robots, where some of the robots may have very limited computational resources. Moreover, many existing approaches maintain the global map in memory for localization purposes, making them unsuitable for large-scale explorations where the map size in memory would significantly increase.

Our previous work [1] presents LOCUS 1.0, a multi-sensor lidar-centric solution for high-precision odometry and 3D mapping in real-time featuring a multi-stage scan matching module, equipped with health-aware sensor integration

that fuses additional sensing modalities in a loosely-coupled scheme. While achieving remarkable accuracy and robustness in perceptually degraded settings, the previous version of LOCUS 1.0: *i*) had a more significant computational load, *ii*) maintained the global map in memory, *iii*) was less robust to more generalized sensor failures, e.g., failure of one of lidar sensor. LOCUS 2.0 presents algorithmic and system-level improvements to decrease the computational load and memory demand, enabling the system to achieve accurate and real-time ego-motion estimation in challenging perceptual conditions over large-scale exploration under severe computation and memory constraints.

The new features and contributions of this work include (i) *GICP from normals*: a novel formulation of Generalized Iterative Closest-Point (GICP) that leverages point cloud normals to approximate the point covariance calculation for enhanced computational efficiency, (ii) *Adaptive voxel grid filter* that ensures deterministic and near-constant runtime, independently of the surrounding environment and lidars, (iii) improvement and evaluation of two *sliding-window map storage* data structures: multi-threaded octree, ikd-tree [2], and (iv) dataset release¹ including in challenging, real-world subterranean environments (urban, tunnel, cave), shown in Fig. 1, collected by heterogeneous robot platforms. All these features improve the computational and memory operation while maintaining accuracy at the same level. The source code of LOCUS 2.0 has been released as an open-source library².

The paper is organized as follows. Sec. II reviews related work on lidar odometry. Sec. III describes the proposed system architecture with a focus on the updates made to the framework to enhance its performance in terms of computational load and memory usage. Sec. IV provides an ablation study of the system on datasets collected by heterogeneous robots during the three circuits of the DARPA Subterranean Challenges, an international robotic competition where robots are tasked to explore complex GNSS-denied underground environments autonomously.

II. RELATED WORKS

Motivated by the need to enable real-time operation under computation constraints and large-scale explorations under memory constraints in perceptually-degraded settings, we review the current state-of-the-art to assess whether any solution can satisfy these requirements simultaneously.

A. Lidar Odometry

The work [3] proposes DILO, a lidar odometry technique that projects a three-dimensional point cloud onto a two-dimensional spherical image plane and exploits image-based odometry techniques to recover the robot ego-motion in a frame-to-frame fashion without requiring map generation. This results in dramatic speed improvements, however, the method does not fuse additional sensing modalities and is not open-source. The work [4] presents BALM, a lidar odometry

solution exploiting bundle adjustment over a sliding window of lidar scans for enhanced mapping accuracy. While the paper claims nearly real-time operation, the method does not fuse sensing modalities and maintains the entire map in memory.

B. Lidar-Inertial Odometry

General challenges encountered in pure lidar-based odometry estimators include degraded motion estimation in high-rate motion scenarios [5], and degenerate motion observability in geometrically-featureless areas (e.g. long corridors, tunnels) [6], [7]. For these reasons, lidars are commonly fused with additional sensing modalities to achieve enhanced accuracy in perceptually-degraded settings [8], [9]. The work [10] presents LIO-SAM, an accurate tightly-coupled lidar-inertial odometry solution via smoothing and mapping that exploits a factor graph for joint optimization of IMU and lidar constraints. Scan-matching at a local scale instead of a global scale significantly improves the real-time performance. The work [11] presents LILI-OM, a tightly-coupled lidar-inertial odometry solution with a lidar/IMU hierarchical keyframe-based sliding window optimization back-end. The work [12] presents LINS, a fast tightly-coupled fusion scheme of lidar and IMU with error-state Kalman filter to recursively correct the estimated state by generating new feature correspondences in each iteration. The work [13] presents RTLIO, a tightly-coupled lidar-inertial odometry pipeline that delivers accurate and high-frequency estimation for the feedback control of UAVs by solving a cost function consisting of lidar and IMU residuals. The work [14] presents FAST-LIO, a computationally-efficient lidar-inertial odometry pipeline that fuses lidar feature points with IMU data in a tightly-coupled scheme with an iterated extended Kalman filter. A novel formula for computing the Kalman gain results in a considerable decrease of computational complexity with respect to the standard formulation, translating into decreased computation time. The work [15] presents DLIO, a lightweight loosely-coupled lidar-inertial odometry solution for efficient operation over constrained platforms. The work provides efficient derivation of local submaps for global refinement constructed by concatenating point clouds associated with historical key-frames, along with a custom iterative closest point solver for fast and lightweight point cloud registration with data structure recycling that eliminates redundant calculations. While these methods are computationally efficient, the methods maintain a global map in memory, rendering it unsuitable for large-scale explorations over memory constraints.

C. Lidar-Visual-Inertial Odometry

The work [9] presents Super Odometry, a robust, IMU-centric multi-sensor fusion framework that achieves accurate operation in perceptually-degraded environments. The approach divides the sensor data processing into several sub-factor-graphs where each sub-factor-graph receives the prediction from an IMU pre-integration factor, recovering the motion from a coarse to fine manner and enhancing the real-time performance. The approach also adopts a dynamic octree data structure to organize the 3D points, making the scan-matching

¹<https://github.com/NeBula-Autonomy/nebula-odometry-dataset>

²<https://github.com/NeBula-Autonomy/LOCUS>

process very efficient and reducing the overall computational demand. However, the method maintains the global map in memory.

The work [8] presents LVI-SAM, a real-time tightly-coupled lidar-visual-inertial odometry solution via smoothing and mapping built atop a factor graph, comprising a visual-inertial subsystem (VIS) and a lidar-inertial subsystem (LIS). However, the method maintains the global map in memory, which is not unsuitable for large-scale explorations with memory limited processing units. The work [16] proposes R2LIVE, an accurate and computationally-efficient sensor fusion framework for lidar, camera, and IMU that exhibits extreme robustness to various failures of individual sensing modalities through filter-based odometry. While not explicitly mentioned in the paper, the open-source implementation of this method features the integration of an ikd-tree data structure for map storage which could be exploited to keep in memory only a robot-centered submap.

III. SYSTEM DESCRIPTION

LOCUS 2.0 provides an accurate Generalized Iterative Closest Point (GICP) algorithm [17] based multi-stage scan matching unit and a health-aware sensor integration module for robust fusion of additional sensing modalities in a loosely coupled scheme. The architecture, shown in Fig. 2, contains three main components: i) *point cloud preprocessor*, ii) *scan matching unit*, iii) *sensor integration module*. The *point cloud preprocessor* is responsible for the management of multiple-input lidar streams to produce a unified 3D data product that can be efficiently processed by the *scan matching unit*. The *preprocessor module* consists of Motion Distortion Correction (MDC) of the point clouds. This module corrects the distortion in the point cloud from sensor rotation during a scan due to robot movement using IMU measurements.

Next, the *Point Cloud Merger* enlarges the robot field-of-view by combining point clouds from different lidar sensors in the robot body frame using their known extrinsic transformation. To enable resilient merging of multiple lidar feeds, we introduce an external timeout-based health monitor that dynamically updates which lidars should be combined in the *Point Cloud Merger* (i.e. a lidar is ignored if its messages are too delayed). The health monitoring makes the submodule robust to lags and failures of individual lidars so that an output product is always provided to the downstream pipeline. Then, the *Body Filter* removes the 3D points that belong to the robot. Next, the *Adaptive Voxel Grid Filter* maintains a fixed number of voxelized points to manage CPU load and to ensure deterministic behavior. It allows the robot to have consistent computational load regardless of the size of the environment or the number of lidars (or if potential lidar failures). In comparison to LOCUS 1.0, the *Adaptive Voxel Grid Filter* changes the strategy of point cloud reduction from a blind voxelization strategy with fixed leaf size and random filter to an adaptive system (Sec. III-B). The *Normal Computation* module calculates normals from the voxelized point cloud. The *scan matching unit* performs a GICP scan-to-scan and scan-to-submap registration to estimate the 6-DOF motion of

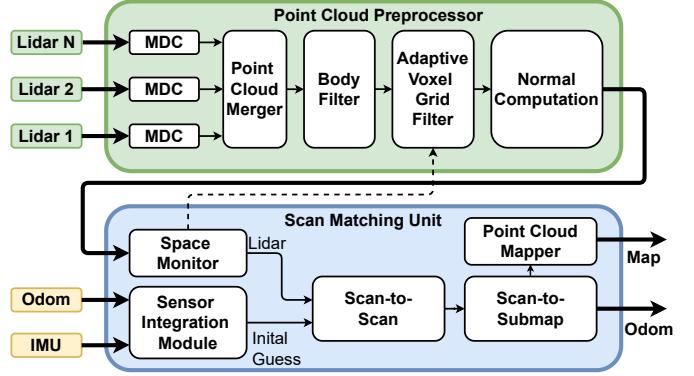


Fig. 2: LOCUS 2.0 architecture.

the robot. LOCUS 2.0, in comparison to its predecessor, does not recalculate covariances but instead leverages a novel GICP formulation to use normals, which only need to be computed once and stored in the map (Sec. III-A).

In robots with multi-modal sensing, when available, LOCUS 2.0 uses an initial estimate from a non-lidar source (from Sensor Integration Module) to ease the convergence of the GICP in the scan-to-scan matching stage, by initializing the optimization with a near-optimal seed that improves accuracy and reduces computation, enhancing real-time performance, as explained in [1].

LOCUS 2.0 also includes a more efficient technique for map storage. The system uses a sliding-window approach because large-scale areas are not feasible to be maintained in memory. For example, in one of the cave datasets presented here, a global map at 1 cm resolution requires 50 GB of memory, far exceeding the typically available memory on small mobile robots. This approach demands efficient computational solutions for insertion, deletion, and search.

A. GICP from normals

LOCUS 2.0 uses GICP for scan-to-scan and scan-to-submap matching. GICP generalizes the point-to-point and point-to-plane ICP registration by using a probabilistic model for the registration problem [17]. To do this, GICP requires the availability of covariances for each point in the point clouds to be aligned. Covariances are usually calculated based on the distributions of neighboring points around a given point. Segal *et al.* [17] presents plane-to-plane application with the assumption that real-world surfaces are at least locally planar. In this formulation, points on surfaces are locally represented by a covariance matrix, where the point is known to belong to a plane with high-confidence, but its exact location in the plane has higher uncertainty.

Here, we show how plane-to-plane covariance calculation is equivalent to calculating covariances from pre-computed normals. The fact that only the normal is needed is especially important for scan-to-submap alignment since the map would otherwise require recomputing point covariances, which is an expensive operation involving the creation of a kd-tree and nearest neighbors search. By instead using normals, the covariance computation is only performed once (since it is not influenced by the addition of extra points), and the result can be stored and reused.

The most common implementations of GICP [18], [19] rely on computing the covariances C_i^B and C_i^A for each point i in two scans. That calculation of covariances in GICP takes place as a pre-processing step whenever two scans have to be registered. In the following, we describe how to obtain the covariances, without recomputing them every time a scan is collected. Any well-defined covariance matrix \mathbf{C} can be eigendecomposed to eigenvalues and eigenvectors [20], $\mathbf{C} = \lambda_1 \mathbf{u}_1 \cdot \mathbf{u}_1^T + \lambda_2 \mathbf{u}_2 \cdot \mathbf{u}_2^T + \lambda_3 \mathbf{u}_3 \cdot \mathbf{u}_3^T$ where $\lambda_1, \lambda_2, \lambda_3$ are the eigenvalues of matrix \mathbf{C} , and $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ are eigenvectors of matrix \mathbf{C} . Two eigenvalues with the same value can be interpreted and visualized as eigenvectors that span equally 2D planar surface with the same distribution in each direction. This allows the covariance computation problem to be thought of geometrically.

In the plane-to-plane metric, for a point a_i from scan A we know the position along the normal with very high confidence, but we are less sure about its location in the plane. To represent this, we set $\mathbf{u}_1 = \mathbf{n}$, and assign $\lambda_1 = \epsilon$ as the variance in the normal direction, where ϵ is a small constant. We can then choose the other eigenvectors to be arbitrary vectors in the plane orthogonal to \mathbf{n} and assign them comparatively large eigenvalues $\lambda_2 = \lambda_3 = 1$, indicating that we are unsure about location in the plane.

Let us take any vector that lies on the plane and is perpendicular to the normal. The vector needs to satisfy the plane equation (that it is perpendicular to the normal vector) and cross the origin: $n_x x + n_y y + n_z z = 0$. Then $z = -\frac{n_x \cdot x + n_y \cdot y}{n_z}$, therefore the family of vectors on the plane is $\mathbf{u}_2 = \frac{(x, y, -(n_x \cdot x + n_y \cdot y)/n_z)}{\|(x, y, -(n_x \cdot x + n_y \cdot y)/n_z)\|}$, where n_z and n_x corresponds to the component z and x of a normal vector \mathbf{n} and $n_z = 0$ means the horizontal vector. The third vector \mathbf{u}_3 needs to simultaneously be perpendicular to \mathbf{u}_1 and \mathbf{u}_2 since eigenvectors need to span the whole 3D space. Therefore, $\mathbf{u}_3 = \mathbf{n} \times \mathbf{u}_2$. If we know the eigenvectors and eigenvalues of matrix \mathbf{C} , we have $\mathbf{C} = \epsilon \mathbf{u}_1 \cdot \mathbf{u}_1^T + 1.0 \mathbf{u}_2 \cdot \mathbf{u}_2^T + 1.0 \mathbf{u}_3 \cdot \mathbf{u}_3^T$. Substituting from above, we get:

$$\mathbf{C} = \epsilon \mathbf{n} \cdot \mathbf{n}^T + 1.0 \mathbf{u}_2 \cdot \mathbf{u}_2^T + 1.0 (\mathbf{n} \times \mathbf{u}_2) \cdot (\mathbf{n} \times \mathbf{u}_2^T) \quad (1)$$

Then, if we take arbitrarily $x = 1$ and $y = 0$ then $z = -\frac{n_x}{n_z}$, and the covariance simplifies to:

$$\begin{aligned} \mathbf{C} &= \epsilon \mathbf{n} \cdot \mathbf{n}^T + \frac{(1, 0, -n_x/n_z)}{\|(1, 0, -n_x/n_z)\|} \cdot \frac{(1, 0, -n_x/n_z)^T}{\|(1, 0, -n_x/n_z)\|} \\ &\quad + \mathbf{n} \times \frac{(1, 0, -n_x/n_z)}{\|(1, 0, -n_x/n_z)\|} \cdot \mathbf{n} \times \frac{(1, 0, -n_x/n_z)^T}{\|(1, 0, -n_x/n_z)\|} \end{aligned} \quad (2)$$

The results mean that the covariance can be purely expressed via precomputed normals at each point.

B. Adaptive Voxel Grid Filter

To manage the computation load of lidar odometry, regardless of the environment and lidar configuration (in terms of number of lidars and types), we propose an adaptive voxel grid filter. In this approach, the goal is to maintain the voxelized number of points at a fixed level (desired by the user) rather than specifying the voxel leaf size and exposing the system to

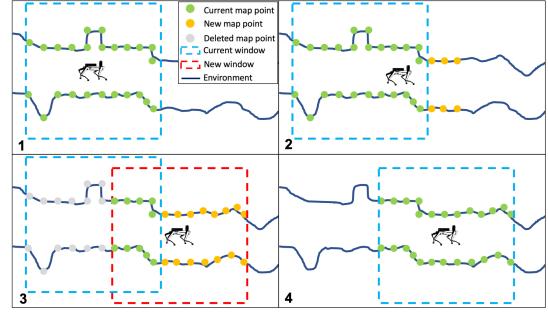


Fig. 3: Illustration of our sliding map approach. All points are maintained until the robot reaches the boundary of the original window (step 3). Then, a new window is set, and points outside that window are deleted.

the variability of the input points that stems from different sensors configurations and cross-sectional geometry of the environment. This design goal comes from the fact that almost all computations in the registration stages are dependent on a number of points N . Therefore the idea is to keep the voxelized number of 3D points fixed to have approximately fixed computation time per scan. The approach is as follows: let us take any size of the initial voxel size d_{init} and set $d_{leaf} = d_{init}$, where d_{leaf} is the size of the voxel leaf in the current time stamp. We propose the following control scheme: $d_{leaf_{t+1}} = d_{leaf_t} \frac{N_{scan}}{N_{desired}}$. The formula describes how much should the current voxel size change $d_{leaf_{t+1}}$ in comparison to what the current size is d_{leaf_t} based on the ratio on the number of points in the current input scan N_{scan} to the points that are desired for computation for given robot $N_{desired}$. This simple technique maintains the number of points on the fixed level, while avoiding any large jumps in the numbers of points, having too few points (e.g. a faulty scan) or having too many points. The result is an improvement in the efficiency and reduction of the computational load of the system.

C. Sliding-window Map

LOCUS 1.0 [1] stored the global map in memory through an octree data structure. The native octree implementation does not have an efficient way to prune data out. While a possible workaround is to filter the points around the robot and rebuild the octree accordingly, this might be computationally expensive and lead to long refreshing times.

To account for these challenges, and enable large-scale explorations under memory constraints, LOCUS 2.0 provides two map sliding-window approaches (Fig. 3): i) multi-threaded octree, ii) incremental k-dtree [21] (ikd-tree).

Multi-Threaded Octree approach maintains only a robot-centered submap of the environment in memory. Two parallel threads ($thread_a$ and $thread_b$) each working on dedicated data structures ($map_a/octree_a$ and $map_b/octree_b$) are responsible to dynamically filter the point cloud map around the current robot position through a box-filter, and rebuild the octree accordingly with the updated map, while accounting for robot motions between parallel worker processes.

Ikdtree [21] is a binary search tree that dynamically stores 3D points by merging new scans. Ikdtree does not maintain 3D points only in the leaf nodes: they have points in the

TABLE I: Dataset summary.

ID	Place	Robot	Distance (m)	Duration (min)	Characteristic	lidars
A	power plant Elma, WA (Urban)	Husky	631.53	59:56	feature-poor corridors, large open spaces	3
B	power plant Elma, WA (Urban)	Spot	664.27	32:26	2-level, stairs, feature-poor corridors, large & narrow spaces	1
C	power plant Elma, WA (Urban)	Husky	757.40	24:21	feature-poor corridors, large & narrow spaces	3*
D	Bruceton Mine Pittsburgh, PA (Tunnel)	Husky	1795.88	65:36	self-similar self-repetitive geometries	3*
E	Lava Beds National Monument, CA (Cave)	Spot	590.85	25:20	lava tubes and pools, non-uniform environment, degraded lightning	1
F	Bruceton Mine Pittsburgh, PA (Tunnel)	Husky	1569.73	49:13	self-similar self-repetitive geometries	3*
G	power plant Elma, WA (Urban)	Husky	877.21	93:10	feature-poor corridors, large & narrow spaces	3
H	Subway Station Los Angeles, CA (Urban)	Spot	1777.45	46:57	3-level, multiple stairs, feature-poor corridors, large & narrow spaces	3
I	Kentucky Underground Limestone Mine, KY (Cave)	Spot	768.82	19:28	large area, non-uniform environment, degraded lightning	1
J	Kentucky Underground Limestone Mine, KY (Cave)	Husky	2339.81	57:55	large area, non-uniform environment, degraded lightning	3

* For our experiments we use only two lidars.

internal nodes as well. This structure allows dynamic insertion and deletion capabilities and relies on lazy labels storage across the whole data structure. Initial building of an ikd-tree is similar to a kd-tree, where space is split at the median point along the longest dimension recursively. Points that are moved out of the boundaries of the ikd-tree data structure are not deleted immediately, but they are labeled as *deleted = True* and maintain information until a rebalancing procedure is triggered.

IV. EXPERIMENTAL RESULTS

A. Dataset

Over the last 3 years, Team CoSTAR [22] has intensively tested our lidar odometry system in real world environment such as caves, tunnels and abandoned factories. Each dataset (Tab. I) is selected to contain components that are challenging for lidar odometry. The dataset provides lidar scans, IMU and wheeled inertial odometry (WIO) measurements, as well cameras stream. All datasets have been recorded on different robotics platforms, e.g., Husky and Spot (Fig. 4) with vibrations and large accelerations as is characteristic of both a skid-steer wheeled robot traversing rough terrain and a legged robot that slips and acts dynamically in rough terrain. The Husky robot is equipped with 3 on-board VLP16 lidar sensors extrinsic calibrated (one flat, one pitched forward 30 deg, one pitched backward 30 deg). The Spot robot is equipped with one on-board lidar sensor extrinsic calibrated. Spot out-of-the-box implements (kinematic inertial odometry) *KIO* and (visual inertial odometry) *VIO*, therefore the data records those readouts as well. Lidar scans are recorded at 10 Hz. WIO and IMU are recorded at 50 Hz. To determine the ground truth of the robot in the environment, a survey-grade



(a) NeBula Spot robot



(b) NeBula Husky robot

Fig. 4: Type of robots for heterogeneous robotic system in Nebula framework for DARPA Subterranean Challenge

3D map (provided by DARPA in the Subterranean Challenge or produced by the team) is used. The ground-truth trajectory is produced by running LOCUS 1.0 against the survey-grade map (i.e. scan-to-map is scan-to-survey-map). In this mode, LOCUS 1.0 is tuned for maximum accuracy at the cost of computational efficiency, as it does not need to be run in real-time. The ground truth trajectory of the robot is determined based on LOCUS 1.0 and its multi-stage registration technique: scan-to-scan and scan-to-map (with high computational parameters and slower pace of data processing) and some manual post-processing work. These datasets have been made open-source to promote further research on lidar odometry and SLAM in underground environments: github.com/NeBula-Autonomy/nebula-odometry-dataset.

B. Metrics

For *CPU and memory profiling*, a cross-platform library for retrieving information on running processes and system utilization is used [23]. The library is used for system monitoring and profiling. The CPU represents the percentage value of the current system-wide CPU utilization, where 100% means 1 core is used. The memory represents statistics by summing different memory values depending on the platform. *Odometry delay* measures the difference between odometry message creation and the current timestamp of the system. The system is implemented in Robot Operating System (ROS) framework. This work considers maximum delay and mean delay since those two metrics more directly impact the performance of the modules using the odometry result, e.g., controllers and path planners. *Lidar callback time* measures the duration time for a scan at the time stamp t_k to go through a pipeline of processing from the queue of the lidar scans. *Scan-to-scan time* measures the duration time for a scan at time t_k to align with a scan at time t_{k-1} in GICP registration stage. *Scan-to-submap time* measures the duration time for a pre-aligned scan from scan-to-scan at time t_k to align with a reference local map in GICP registration.

C. Computation time

1) *GICP from normals*: The experiments presented in this section are designed to show the benefit of *GICP from normals* over GICP and support the claim that this reformulation yields better computation performance without sacrificing accuracy. For each dataset, we compute statistics over 5 runs. The GICP parameters for this experiment are chosen based on [24]. The parameters for scan-to-scan and scan-to-submap are

the same: optimization step 1e–10, maximum corresponding distances for associations 0.3, maximum number of iterations in optimization 20, rotational fitness score threshold 0.005. Husky computation runs in 4 threads, while Spot uses only 1 thread due to CPU limitations. The octree stores the map with a leaf size 0.001 m .

The Fig. 5.a-e present the comparison results between *GICP from normals* and GICP across datasets, while Fig. 5.f shows the average percentage change across all dataset for each metric with respect to the GICP method. *GICP from normals* reduces all the computational metrics in LOCUS 2.0: mean and max CPU usage, mean and max odometry delay, scan-to-scan, scan-to-submap, lidar callback duration and their maximum times. The computation burden is, on average, reduced by 18.57% for all those metrics and datasets. This reduction benefits the odometry update rate since frequency increases by 11.10% that is beneficial for another part of the system, i.e. for path planning and control algorithms. The lidar callback is generally higher for datasets I and J largely due to the consistently large cross-section of Kentucky Underground making GICP take longer. One drawback of this method leads to slight increase in the mean and max APE errors. The reason is that normals are calculated from sparse point clouds, and those normals are stored in the map without any further recomputation. In GICP, the covariances are recalculated from dense map point clouds. The mean and max APE error increases across all datasets on average 10.82%, but the increase is not for all datasets. Without including the tunnel dataset (F) the average APE error is only 5.23%. The rotational APE errors do not change much since max APE decreases 0.94%, while mean APE increases 0.1%.

2) *Adaptive Voxel Grid Filter*: The second experiment presented in this section shows LOCUS 2.0 adaptive behavior. The experiments are run across all datasets with *GICP from normals* and the same parameters as in Sec. IV-C1 with an ikd-tree data structure for map maintenance with a box size 50 m , and $N_{desired}$ ranging from 1000 to 10000. Fig. 6.a shows how the adaptive voxel grid filter keeps the number of points relatively consistent across a 1 hour dataset, no matter what the input $N_{desired}$ is.

There is still some variability in the computation time across a dataset, though. Nonetheless, as shown in Fig. 8, the approach produces a consistent average computation time across different environments and sensor configurations, without any large spikes in computation time. This performance gives more predictable computational loads, regardless of robot or environment, as further reinforced in Fig. 7, where the average callback time and CPU load are similar for all datasets at the same adaptive voxelization setting.

D. Memory

1) *Map maintenance*: The third experiment presented in this section presents the benefit of sliding-window maps in real-time systems compared to classic, static octree-based structures. In these experiments, LOCUS 2.0 uses: ikd-tree and multi-threaded octree (mto). The octree with leaf size

TABLE II: Relative memory and CPU change.

	ikd-tree	mto 0.001	mto 0.01	mto 0.1	octree 0.001
Memory	–68.09%	–38.88%	–62.15%	–87.76%	X
CPU	9.36%	50.42%	44.36%	19.61%	X

0.001 m is the baseline used in LOCUS 1.0 to maintain full map information. To assess different parameters mto runs with leaf size 0.1 m , 0.01 m , and 0.001 m .

For sliding-window approaches, the map size is 50 m since it is the maximum range of lidars. For scan-to-scan and scan-to-submap stage *GICP from normals* is used with the parameters chosen based on previous experiments. Fig. 9 presents the maximum memory use for F and I dataset and how memory occupation evolves over time. The largest memory occupancy is for octree and mto version for 0.001 m leaf size. The ikd-tree achieves similar performance in terms of memory and CPU usage as the mto with leaf size 0.01 m . Tab. II shows how sliding-window map approaches reduce the memory usage while increasing the CPU usage in comparison to the reference method from LOCUS 1.0.

Fig. 6.b shows the deletion, insertion and searching procedure timings for different mapping strategies. Ikdtree has the most time-consuming procedures for searching, deletion, and insertion across all datasets. For insertion, ikd-tree uses on average 222% more computation time than the octree data structure as new points need to be stored in a particular manner. For search, ikd-tree gives on average 140% more computation than the octree data structure.

2) *Map size*: These experiments show that the size of the sliding map is an important parameter to consider while trading off the computational, memory load and accuracy of the result. The sliding-window map allows the system to be bounded by the maximal memory that the robot allocates, following the paradigms of Real-Time Operating System requirements. Fig. 10 shows the max APE, CPU and memory metrics for ikd-tree and mto in terms of map size. The smaller map size gives the robot a lower upper bound for the memory, but on the other hand, instances with larger maps have lower APE as there is more overlap between scan and map. Other than memory, these larger maps also see larger the mean and max CPU load.

E. Comparison to the state-of-the-art

Tab. III shows the comparison study for LOCUS 2.0 against the state-of-the-art methods FAST-LIO [14] and LINS [12] for different environment domains: urban, tunnel, cave (A,C,F,H,I,J). The table shows that LOCUS 2.0 presents a state-of-the-art performance in terms of max and mean APE error metrics and achieves the smallest errors in 5 out of 6 presented datasets. In addition, LOCUS 2.0 is the only method that does not fail in the tunnel type of the environment (dataset F) where lidar slip occurs. In terms of computation, LOCUS 2.0 achieves equivalent performance to FAST-LIO. Memory usage for LOCUS 2.0 is slightly larger, yet this is likely related to the resolution of the map chosen by default in all the systems.

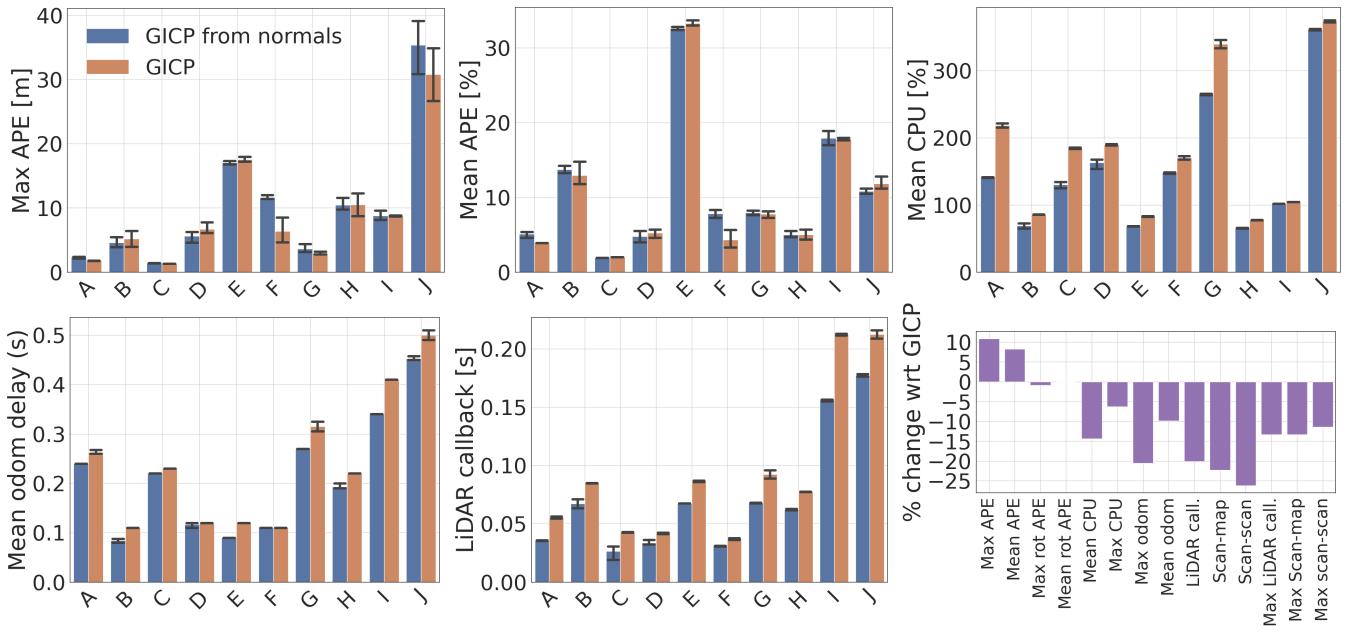


Fig. 5: Results of GICP from normals and GICP comparison in LOCUS 2.0. For the meaning of the labels A-J see Tab. I

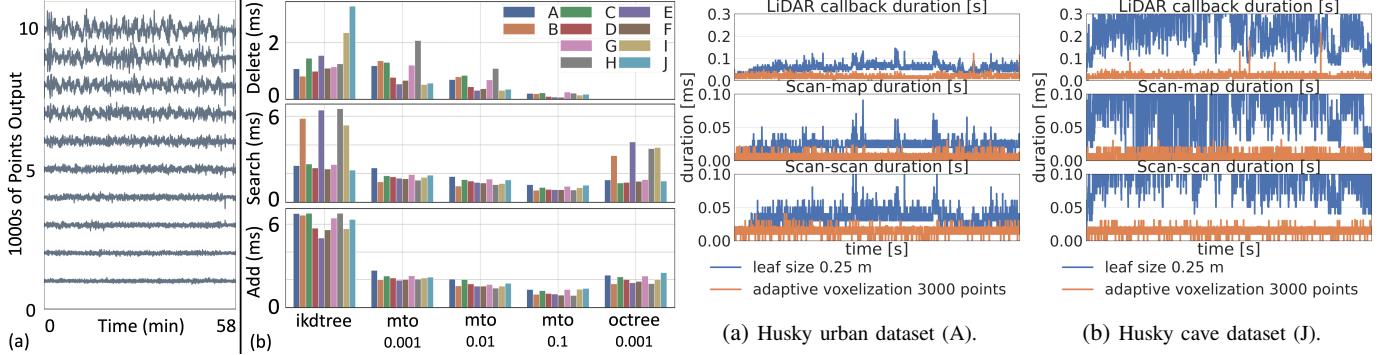


Fig. 6: (a) Number of points after the adaptive voxel grid filter for different set-points (on dataset I). (b) Timeplots for deleting, adding, searching for different map storage mechanisms.

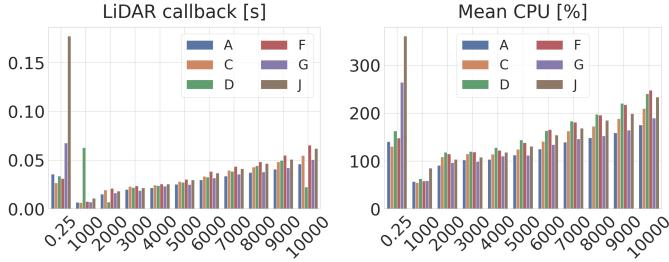


Fig. 7: The figure presents comparison between adaptive voxelization 1000 – 10000 points and constant leaf size 0.25 (our previously used static voxel size). For 0.25 voxel the average number points for each dataset (A, C, D, F, G, J) is 8423, 5658, 2967, 2368, 8511, 15901.

V. CONCLUSIONS

This work presents LOCUS 2.0, a robust and computational efficient lidar odometry system for real-time, large-scale explorations under severe computation and memory constraints suitable to be deployed over heterogeneous robotic platforms. This work reformulates GICP covariance calculations from precomputed normals that improves the computational performance of GICP. LOCUS 2.0 uses an adaptive voxel grid filter and makes

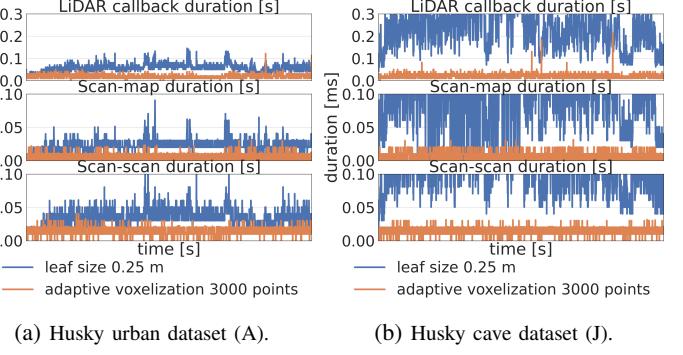


Fig. 8: A comparison of the consistency of computation time for adaptive voxelization with 3000 points and constant leaf size 0.25 (our previously used static voxel size). a) Urban dataset using 2 lidars. b) Cave dataset using 3 lidars.

TABLE III: Comparison of LOCUS 2.0 to the state-of-the art methods FAST-LIO [14] and LINS [12] for different environment domains.

Dataset	Algorithms	APE		CPU [%]		max memory
		max [m]	mean [%]	max	mean	
A	LOCUS 2.0	0.19	0.09	102.38	185.50	1.06
	FAST-LIO	0.79	0.30	89.11	126.40	0.36
	LINS	0.43	0.18	40.84	81.50	0.42
C	LOCUS 2.0	0.16	0.24	114.79	198.00	1.30
	FAST-LIO	2.21	4.22	76.46	307.20	0.99
	LINS	0.43	0.60	38.43	75.30	0.47
F	LOCUS 2.0	0.67	0.45	119.00	229.20	1.98
	FAST-LIO	48555.33	9268.71	156.73	401.30	11.31
	LINS	52.73	23.35	28.10	52.30	0.47
H	LOCUS 2.0	0.57	0.23	61.05	169.90	2.42
	FAST-LIO	5.92	5.69	75.15	160.80	0.62
	LINS	12.11	8.05	39.19	97.90	0.61
I	LOCUS 2.0	1.39	1.95	72.11	141.60	1.01
	FAST-LIO	0.99	1.44	117.87	167.80	0.80
	LINS	0.86	0.85	75.90	101.40	0.85
J	LOCUS 2.0	2.42	3.88	107.72	185.00	2.13
	FAST-LIO	1.72	2.60	126.72	332.50	2.54
	LINS	3.56	5.79	73.76	176.50	1.85

the computational load independent on the environment and sensor configuration. Adaptive behavior keeps the number of points from the lidar consistent while keeping the voxelized

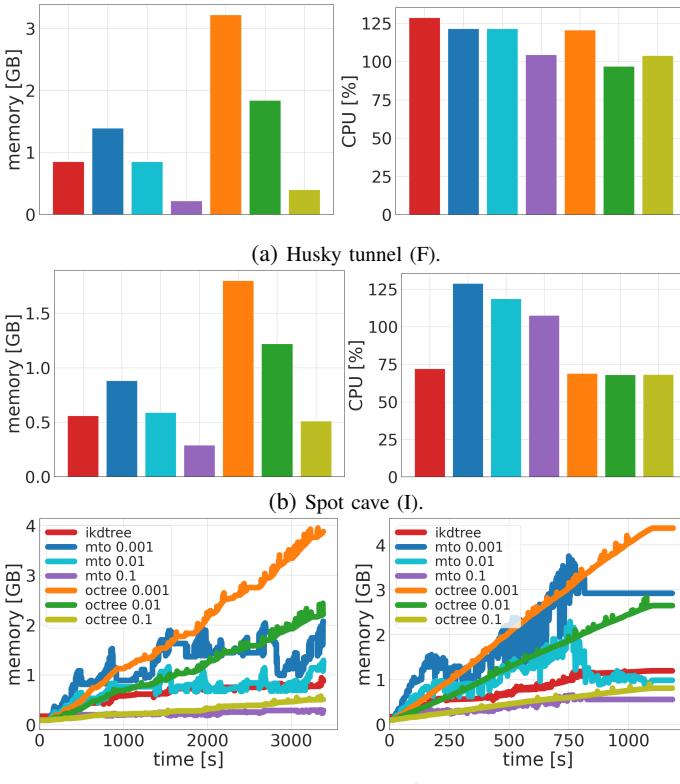


Fig. 9: The plots show the metrics for different datasets relative to the number of fixed voxelized points.

structure of the environment, which stabilizes and improves the computational load. We evaluate two sliding map strategies for reducing memory use: multi-threaded octree and ikd-tree, and show both their computational cost, and improvement in memory usage. We open-source both LOCUS 2.0, and our dataset for challenging and large-scale underground environments that features various real-world conditions such as fog, dust, darkness, and geometrically degenerate environments that restrict mobility. Overall the datasets include 11 h of operations and 16 km distance traveled.

REFERENCES

- [1] M. Palieri, B. Morrell, A. Thakur, K. Ebadi, J. Nash, A. Chatterjee, C. Kanellakis, L. Carbone, C. Guaragnella, and A.-a. Agha-Mohammadi, “Locus: A multi-sensor lidar-centric solution for high-precision odometry and 3d mapping in real-time,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 421–428, 2020.
- [2] Y. Cai, W. Xu, and F. Zhang, “ikd-tree: An incremental kd tree for robotic applications,” *arXiv preprint arXiv:2102.10808*, 2021.
- [3] S.-J. Han, J. Kang, K.-W. Min, and J. Choi, “Dilo: Direct light detection and ranging odometry based on spherical range images for autonomous driving,” *ETRI Journal*, vol. 43, no. 4, pp. 603–616, 2021.
- [4] Z. Liu and F. Zhang, “BALM: Bundle adjustment for lidar mapping,” *IEEE RAL*, vol. 6, no. 2, pp. 3184–3191, 2021.
- [5] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” in *Robotics: Science and Systems*, vol. 2, 2014, p. 9.
- [6] A. Tagliabue, J. Tordesillas, X. Cai, A. Santamaría-Navarro, J. P. How, L. Carbone, and A.-a. Agha-mohammadi, “Lion: Lidar-inertial observability-aware navigator for vision-denied environments,” *arXiv preprint arXiv:2102.03443*, 2021.
- [7] K. Ebadi, M. Palieri, S. Wood, C. Padgett, and A. akbar Aghamohammadi, “DARE-SLAM: Degeneracy-aware and resilient loop closing in perceptually-degraded environments,” *Journal of Intelligent Robotic Systems*, vol. 102, no. 1, pp. 1–25, 2021.
- [8] T. Shan, B. Englot, C. Ratti, and D. Rus, “Lvi-sam: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping,” *arXiv preprint arXiv:2104.10831*, 2021.
- [9] S. Zhao, H. Zhang, P. Wang, L. Nogueira, and S. Scherer, “Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments,” *arXiv preprint arXiv:2104.14938*, 2021.
- [10] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, “Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5135–5142.
- [11] K. Li, M. Li, and U. D. Hanebeck, “Towards high-performance solid-state-lidar-inertial odometry and mapping,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5167–5174, 2021.
- [12] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang, and M. Liu, “Lins: A lidar-inertial state estimator for robust and efficient navigation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8899–8906.
- [13] J.-C. Yang, C.-J. Lin, B.-Y. You, Y.-L. Yan, and T.-H. Cheng, “Rtlio: Real-time lidar-inertial odometry and mapping for uavs,” *Sensors*, vol. 21, no. 12, p. 3955, 2021.
- [14] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, “Fast-lio2: Fast direct lidar-inertial odometry,” *arXiv preprint arXiv:2107.06829*, 2021.
- [15] K. Chen, B. T. Lopez, A.-a. Agha-mohammadi, and A. Mehta, “Direct lidar odometry: Fast localization with dense point clouds,” *arXiv preprint arXiv:2110.00605*, 2021.
- [16] J. Lin, C. Zheng, W. Xu, and F. Zhang, “R2live: A robust, real-time, lidar-inertial-visual tightly-coupled state estimator and mapping,” *arXiv preprint arXiv:2102.12400*, 2021.
- [17] A. Segal, D. Haehnel, and S. Thrun, “Generalized-icp,” in *In Robotics: science and systems*, vol. 2, no. 4, 2009, p. p. 435.
- [18] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” in *ICRA*. IEEE, 2011, pp. 1–4.
- [19] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, “Voxelized gicp for fast and accurate 3d point cloud registration,” EasyChair Preprint no. 2703, EasyChair, 2020.
- [20] G. Strang, *Introduction to Linear Algebra*, 4th ed. Wellesley, MA: Wellesley-Cambridge Press, 2009.
- [21] Y. Cai, W. Xu, and F. Zhang, “ikd-tree: An incremental k-d tree for robotic applications,” 02 2021.
- [22] A. Agha et al., “Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge,” *ArXiv*, vol. abs/2103.11470, 2021.
- [23] “psutil,” <https://psutil.readthedocs.io/en/latest/>.
- [24] A. Reinke, “Advanced lidar odometry for exploration of unknown underground environments,” Master’s thesis, University of Bonn, 2022.

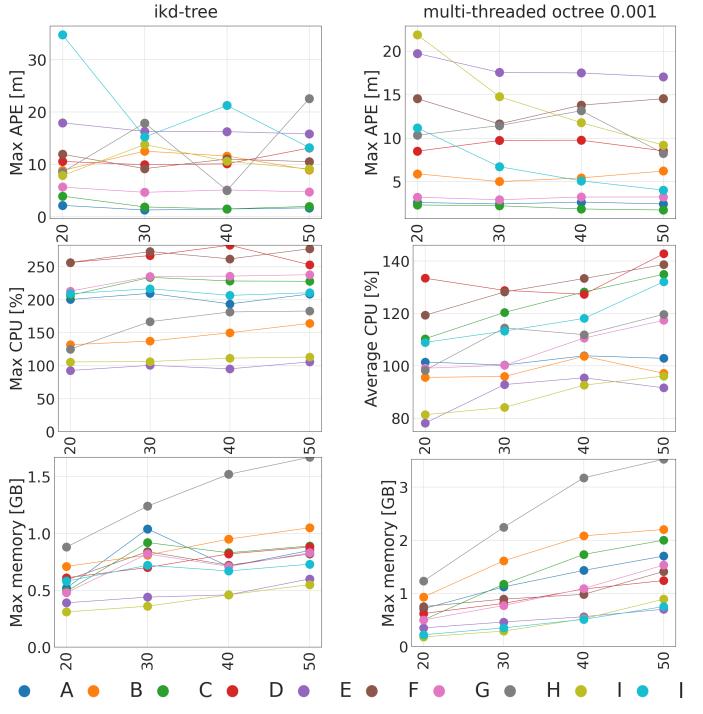


Fig. 10: Results based on size of the map for ikd-tree and mto 0.001 across all datasets (A-J).