Direct LiDAR Odometry: Fast Localization with Dense Point Clouds

Kenny Chen[1], Brett T. Lopez[2], Ali-akbar Agha-mohammadi[3], and Ankur Mehta[1]

***Abstract*— Field robotics in perceptually-challenging environments require fast and accurate state estimation, but modern LiDAR sensors quickly overwhelm current odometry algorithms. To this end, this paper presents a lightweight frontend LiDAR odometry solution with consistent and accurate localization for computationally-limited robotic platforms. Our Direct LiDAR Odometry (DLO) method includes several key algorithmic innovations which prioritize computational efficiency and enables the use of dense, minimally-preprocessed point clouds to provide accurate pose estimates in real-time. This is achieved through a novel keyframing system which efficiently manages historical map information, in addition to a custom iterative closest point solver for fast point cloud registration with data structure recycling. Our method is more accurate with lower computational overhead than the current state-of-the-art and has been extensively evaluated in several perceptually-challenging environments on aerial and legged robots as part of NASA JPL Team CoSTAR's research and development efforts for the DARPA Subterranean Challenge.**

I. Introduction

Accurate state estimation and mapping in large, perceptually-challenging environments have become critical capabilities for autonomous mobile robots. Whereas typical visual SLAM approaches often perform poorly in dust, fog, or low-light conditions, LiDAR-based methods can provide more reliable localization due to the superior range and accuracy of direct depth measurements [1]. However, recent work on LiDAR odometry (LO) have revealed the challenges of processing the large number of depth returns generated by commercial LiDAR sensors in real-time for high-rate state estimation [2], [3]. This work presents several algorithmic innovations that make real-time localization with dense LiDAR scans feasible while also demonstrating the superiority of our method in terms of accuracy and computational complexity when compared to the state-of-the-art.

Current LO algorithms estimate a robot's egomotion in two stages: first, by performing a "scan-to-scan" alignment between adjacent LiDAR frames to recover an immediate motion guess, followed by a "scan-to-map" registration between the current scan and past environmental knowledge to increase global pose consistency. Unfortunately, the large number of data points per scan from modern LiDARs quickly overwhelms computationally-limited processors and bottlenecks performance during alignment, which can induce frame drops and ultimately cause poor pose estimation. More specifically, scan-to-scan alignment requires a registration of corresponding points between two clouds, but this process often involves a nearest-neighbor search which grows exponentially with the number of points per scan. Feature-based methods [2], [4]–[6] attempt to mitigate this by using only the most salient points, but these methods employ an often computationally-intensive feature extraction step and may accidentally discard data which could otherwise help improve the quality of downstream registration. Moreover, in scan-to-map alignment, keyed environmental history (which consists of all or a subset of past points) grows rapidly in size as new scans are acquired and stored in memory. While aligning with a submap (rather than the full history of scans) helps increase computational efficiency, the perpetual addition of points still significantly expands the nearest-neighbor search space for typical submap extraction methods. Tree-based data structures have been shown to decrease this nearest-neighbor search cost significantly [7], but the extraction of a local submap still involves too many points after just a few keyframes, thus preventing consistent performance for long-term navigation.

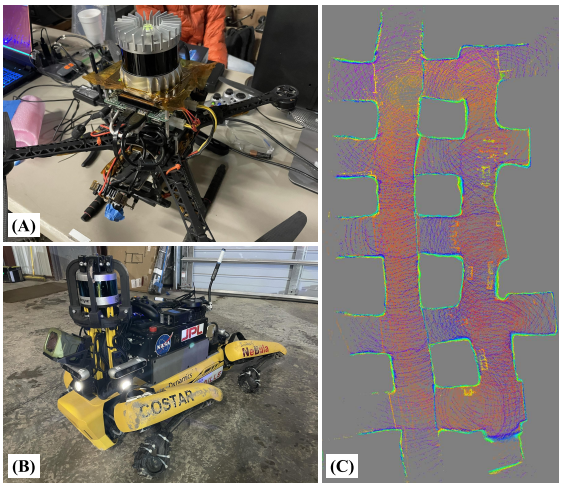

Fig. 1. **Fast and lightweight LiDAR odometry.** Two of Team CoSTAR's robotic platforms which have limited computational resources. (A) Our custom quadrotor platform which features an Ouster OS1 LiDAR sensor on top. (B) A Boston Dynamics Spot robot with a mounted custom payload and a Velodyne VLP-16 with protective guards. (C) Top-down view of a mapped limestone mine using our lightweight odometry method on these robots during testing and integration for the DARPA Subterranean Challenge.

In this letter, we present our Direct LiDAR Odometry (DLO) algorithm, a high-speed and computationally-efficient frontend localization solution which permits the direct use of dense point cloud scans without significant preprocessing. The main contribution of this work is a custom speed-first

[1]Department of Electrical and Computer Engineering, University of California Los Angeles, Los Angeles, CA, USA. {kennyjchen, mehtank}@ucla.edu

[2]Department of Mechanical and Aerospace Engineering, University of California Los Angeles, Los Angeles, CA, USA. btlopez@ucla.edu

[3]NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA. aliagha@jpl.nasa.gov

Work partially conducted at NASA JPL in support of the DARPA Subterranean Challenge.

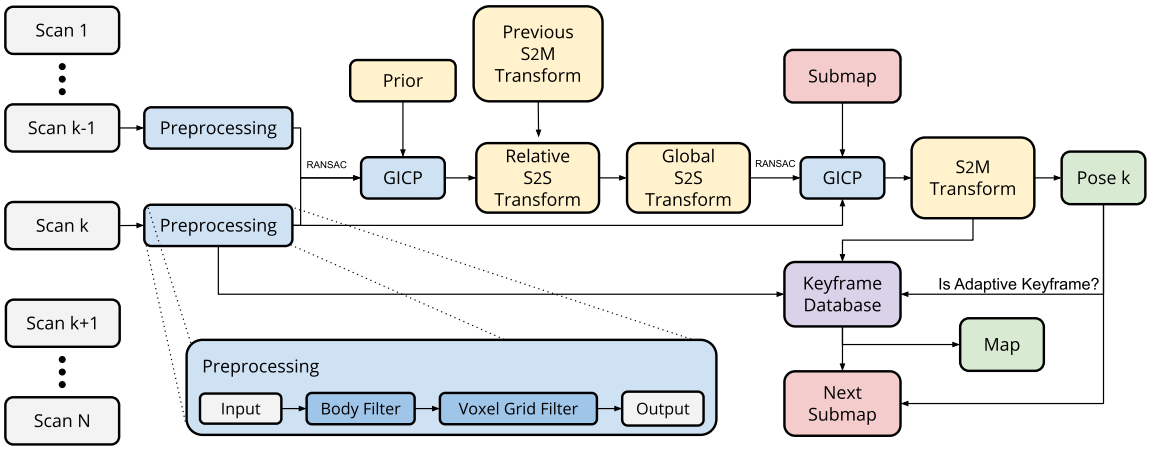

Fig. 2. **LiDAR odometry architecture.** Our system first retrieves a relative transform between two temporally-adjacent scans of times k and $k-1$ through scan-to-scan (S2S) matching with RANSAC outlier rejection and an optional rotational prior from IMU. This initial estimate is then propagated into the world frame and used as the initialization point for our secondary GICP module for scan-to-map optimization (S2M), which scan-matches the current point cloud $\mathcal{P}_k$ with a derived submap $\mathcal{S}_k$ consisting of scans from nearby and boundary keyframes. The output of this is a globally-consistent pose estimate which is subsequently checked against several metrics to determine if the current pose should be stored as a new keyframe.

pipeline which accurately resolves robot egomotion in real-time using minimally-preprocessed LiDAR scans and an optional IMU on consumer-grade processors. A key insight of our work is the link between algorithmic speed and accuracy, and our approach is comprised of three core innovations. First, an adaptive keyframing system which efficiently captures significant environmental information through a novel spaciousness metric. Second, a fast keyframe-based submapping approach via convex optimization which generates permissive local submaps for global pose refinement. Third, NanoGICP, a custom iterative closet point solver for lightweight point cloud scan-matching with data structure recycling to eliminate redundant calculations. Our method has been extensively evaluated in several challenging environments on computationally-limited robotic platforms as part of Team CoSTAR's research and development efforts for the DARPA Subterranean Challenge, and we have open-sourced our code for benefit of the community[1].

Related Work

LiDAR-based odometry is typically cast as a nonlinear optimization problem to calculate a best-fit homogeneous transform that minimizes the error across corresponding, i.e., matching, points and/or planes between two point clouds. Since correspondences are not known *a priori*, techniques such as the iterative closest point (ICP) algorithm [8] or other variants like Generalized ICP (GICP) [9] have become the standard to align two point clouds; however, searching over all data points can be computationally costly. *Feature*-based methods attempt to extract and use only the most salient points before scan-matching to reduce computation. Such features are found either via manually tuned methods [10] or learned networks [11] and might consist of planes [5], lines and edges [4], [6], or ground points [2]. These works aim to translate insights gained from visual odometry (VO) techniques into the 3D domain. However, adding this step increases computational overhead and risks discarding data points which could help with better correspondence matching for odometry accuracy. Alternatively, *direct* methods attempt to align dense point clouds but must heavily downsample to achieve computational tractability [12], [13]. More recently, a recursive filtering framework, e.g. Kalman filter, has been proposed [14], [15] to achieve real-time performance but at the potential expense of estimation accuracy.

A second stage immediately following scan alignment between adjacent clouds has been shown to reduce global drift by increasing pose estimation consistency with previous past scans [3], [13]. In the scan-to-map stage, a scan-to-scan transformation is further refined by aligning the current point cloud with an existing in-memory map; this submap is typically derived by retrieving nearby map points within some radius of the robot's current position. However, this search in "point-space" can quickly explode in computational expense due to the sheer number of operations needed to retrieve the nearest neighbor data points. While there exists techniques to mitigate this such as only incrementally storing map data at keyed locations [6], this search still involves thousands of calculations which can increase overall processor load and hence the potential to drop frames.

To address these issues, our DLO algorithm is built around a "speed-first" philosophy to permit the use of minimally-preprocessed point clouds and provide accurate pose estimates even for robots with limited computational resources (Fig. 2). The key contribution of our work lies in how we efficiently derive our submap for global refinement in scan-to-map matching. That is, rather than extracting points within a local vicinity of a robot's current position as most works do, DLO instead searches in keyframe-space by associating a scan's set of points with its corresponding keyframe position. The submap is subsequently constructed by concatenating the clouds from a subset of historic keyframes derived from nearby keyframes and those which make up the convex hull; this provides the current

[1] https://github.com/vectr-ucla/direct_lidar_odometry

scan with both nearby and distant points in the submap to anchor to. In addition, a custom GICP solver enables extensive reuse of data structures across multiple solver instantiations to eliminate redundant operations across the two-stage process. Our system also optionally accepts an initialization prior from an IMU in a loosely-coupled fashion to further improve accuracy during aggressive rotational motions. The reliability of our approach is demonstrated through extensive tests on several computationally-limited robotic platforms in multiple challenging environments. This work was part of Team CoSTAR's research and development efforts for the DARPA Subterranean Challenge in support of NASA JPL's Networked Belief-aware Perceptual Autonomy (NeBula) framework [16], in which DLO was the primary state estimation component for our fleet of autonomous aerial vehicles (Fig. 1A).

II. Methods

A. Notation

A point cloud, $\mathcal{P}$, is composed of a set of points $p \in \mathcal{P}$ with Cartesian coordinates $p_i \in \mathbb{R}^3$. We denote $\mathcal{L}$ as the LiDAR's coordinate system, $\mathcal{B}$ as the robot's coordinate system located at the IMU frame, and $\mathcal{W}$ as the world coordinate system which coincides with $\mathcal{B}$ at the initial position. Note that in this work we assume $\mathcal{L}$ and $\mathcal{B}$ reference frames coincide. Submap, covariance, and kdtree structures are denoted as $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{T}$, respectively. We adopt standard convention such that x points forward, y points left, and z points upward, and our work attempts to address the following problem: given adjacent point clouds scans $\mathcal{P}_k$ and $\mathcal{P}_{k-1}$ at time k, estimate the robot's current pose $\hat{\mathbf{X}}_k^{\mathcal{W}} \in \mathbb{SE}(3)$ and map $\mathcal{M}_k$ in $\mathcal{W}$.

B. Preprocessing

Our system assumes an input of 3D point cloud data gathered by a 360° LiDAR such as an Ouster OS1 (20Hz) or a Velodyne VLP-16 (10Hz). To minimize information loss from the raw sensor data, only two filters are used during preprocessing: first, we remove all point returns that may be from the robot itself through a box filter of size 1m^3 around the origin. This is especially important if an aerial robot's propellers (Fig. 1A) or protective guards (Fig. 1B) are in the LiDAR's field of view. The resulting cloud is then sent through a 3D voxel grid filter with a resolution of 0.25m to lightly downsample the data for subsequent tasks while maintaining dominate structures within the surrounding environment. Note that in this work we do not correct for motion distortion since non-rigid transformations can be computationally burdensome, and we directly use the dense point cloud rather than extracting features. On average, each cloud contains ∼10,000 points after preprocessing.

C. Scan Matching via Generalized-ICP

LiDAR-based odometry can be viewed as the process of resolving a robot's egomotion by means of comparing successive point clouds and point clouds in-memory to recover

Algorithm 1: Direct LiDAR Odometry

input: $\mathcal{P}_k$, $\hat{\mathbf{X}}_{k-1}^{\mathcal{W}}$; **initialize:** $\hat{\mathbf{X}}_{k-1}^{\mathcal{W}} = \mathbf{I}$ or gravityAlign()
output: $\hat{\mathbf{X}}_k^{\mathcal{W}}$, $\mathcal{M}_k$
while $\mathcal{P}_k \neq \emptyset$ **do**
  // preprocessing
  $\bar{\mathcal{P}}_k \leftarrow$ preprocessPoints($\mathcal{P}_k$) ;
  computeAdaptiveParameters($\bar{\mathcal{P}}_k$) ;
  // initialization
  if $k = 0$ **then**
    $\mathcal{T}_k^{\mathrm{t}_1}$, $\mathcal{C}_k^{\mathrm{t}_1} \leftarrow$ NanoGICP$_1$.build($\bar{\mathcal{P}}_k$) ;
    $\mathcal{K}_k \leftarrow$ updateKeyframeDatabase($\hat{\mathbf{X}}_{k-1}^{\mathcal{W}}$, $\bar{\mathcal{P}}_k$) ;
    continue;
  end
  // prior
  if *IMU* **then** $\tilde{\mathbf{X}}_k^{\mathcal{L}} \leftarrow \tilde{\mathbf{X}}_k^{\mathcal{B}}$; **else** $\tilde{\mathbf{X}}_k^{\mathcal{L}} \leftarrow \mathbf{I}$;
  // scan-to-scan
  $\mathcal{T}_k^{\mathrm{s}_1}$, $\mathcal{C}_k^{\mathrm{s}_1} \leftarrow$ NanoGICP$_1$.build($\bar{\mathcal{P}}_k$) ;
  $\hat{\mathbf{X}}_k^{\mathcal{L}} \leftarrow$ NanoGICP$_1$.align($\mathcal{T}_k^{\mathrm{s}_1}$, $\mathcal{T}_k^{\mathrm{t}_1}$, $\mathcal{C}_k^{\mathrm{s}_1}$, $\mathcal{C}_k^{\mathrm{t}_1}$, $\tilde{\mathbf{X}}_k^{\mathcal{L}}$) ;
  $\tilde{\mathbf{X}}_k^{\mathcal{W}} \leftarrow \hat{\mathbf{X}}_{k-1}^{\mathcal{W}}\ \hat{\mathbf{X}}_k^{\mathcal{L}}$;
  // scan-to-map
  $\mathcal{Q}_k \leftarrow$ getKeyframeNeighbors($\hat{\mathbf{X}}_{k-1}^{\mathcal{W}}$, $\mathcal{K}_k$) ;
  $\mathcal{H}_k \leftarrow$ getKeyframeHulls($\hat{\mathbf{X}}_{k-1}^{\mathcal{W}}$, $\mathcal{K}_k$) ;
  $\mathcal{S}_k \leftarrow \mathcal{Q}_k \oplus \mathcal{H}_k$;
  if $\mathcal{S}_k \neq \mathcal{S}_{k-1}$ **then** $\mathcal{T}_k^{\mathrm{t}_2} \leftarrow$ NanoGICP$_2$.build($\mathcal{S}_k$) ;
   else $\mathcal{T}_k^{\mathrm{t}_2} \leftarrow \mathcal{T}_{k-1}^{\mathrm{t}_2}$;
  $\mathcal{T}_k^{\mathrm{s}_2} \leftarrow \mathcal{T}_k^{\mathrm{s}_1}$; $\mathcal{C}_k^{\mathrm{s}_2} \leftarrow \mathcal{C}_k^{\mathrm{s}_1}$; $\mathcal{C}_k^{\mathrm{t}_2} \leftarrow \sum_n^N \mathcal{C}_{k,n}^{\mathcal{S}}$;
  $\hat{\mathbf{X}}_k^{\mathcal{W}} \leftarrow$ NanoGICP$_2$.align($\mathcal{T}_k^{\mathrm{s}_2}$, $\mathcal{T}_k^{\mathrm{t}_2}$, $\mathcal{C}_k^{\mathrm{s}_2}$, $\mathcal{C}_k^{\mathrm{t}_2}$, $\tilde{\mathbf{X}}_k^{\mathcal{W}}$) ;
  // update keyframe database and map
  $\mathcal{K}_k \leftarrow$ updateKeyframeDatabase($\hat{\mathbf{X}}_k^{\mathcal{W}}$, $\bar{\mathcal{P}}_k$) ;
  $\mathcal{M}_k \leftarrow \mathcal{M}_{k-1} \oplus \{\mathcal{K}_k \setminus \mathcal{K}_{k-1}\}$;
  // propagate data structures
  $\mathcal{T}_k^{\mathrm{t}_1} \leftarrow \mathcal{T}_k^{\mathrm{s}_1}$; $\mathcal{C}_k^{\mathrm{t}_1} \leftarrow \mathcal{C}_k^{\mathrm{s}_1}$;
  return $\hat{\mathbf{X}}_k^{\mathcal{W}}$, $\mathcal{M}_k$
end

an $\mathbb{SE}(3)$ transformation, which translates to the robot's 6-DOF motion between consecutive LiDAR acquisitions. This process is typically performed in two stages, first to provide a best instantaneous guess, which is subsequently refined to be more globally consistent with previous keyframe locations.

1) Scan-to-Scan: In the first stage, the scan-to-scan matching objective is to compute a relative transform $\hat{\mathbf{X}}_k^{\mathcal{L}}$ between a source $\mathcal{P}_k^{\mathrm{s}}$ and a target $\mathcal{P}_k^{\mathrm{t}}$ (where $\mathcal{P}_k^{\mathrm{t}} = \mathcal{P}_{k-1}^{\mathrm{s}}$) captured in $\mathcal{L}$ where

$$\hat{\mathbf{X}}_k^{\mathcal{L}} = \underset{\mathbf{X}_k^{\mathcal{L}}}{\arg\min}\ \mathcal{E}\left(\mathbf{X}_k^{\mathcal{L}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{P}_k^{\mathrm{t}}\right) . \tag{1}$$

The residual error $\mathcal{E}$ from GICP is defined as

$$\mathcal{E}\left(\mathbf{X}_k^{\mathcal{L}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{P}_k^{\mathrm{t}}\right) = \sum_i^N d_i^\top \left(\mathcal{C}_{k,i}^{\mathrm{t}} + \mathbf{X}_k^{\mathcal{L}}\mathcal{C}_{k,i}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{L}\top}\right)^{-1} d_i\,, \tag{2}$$

such that the overall objective for this stage is

$$\hat{\mathbf{X}}_k^{\mathcal{L}} = \underset{\mathbf{X}_k^{\mathcal{L}}}{\arg\min} \sum_i^N d_i^\top \left(\mathcal{C}_{k,i}^{\mathrm{t}} + \mathbf{X}_k^{\mathcal{L}}\mathcal{C}_{k,i}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{L}\top}\right)^{-1} d_i\,, \tag{3}$$

for N number of corresponding points between point clouds $\mathcal{P}_k^{\mathrm{s}}$ and $\mathcal{P}_k^{\mathrm{t}}$, where $d_i = p_i^{\mathrm{t}} - \mathbf{X}_k^{\mathcal{L}}p_i^{\mathrm{s}}$, $p_i^{\mathrm{s}} \in \mathcal{P}_k^{\mathrm{s}}, p_i^{\mathrm{t}} \in \mathcal{P}_k^{\mathrm{t}}, \forall i$, and $\mathcal{C}_{k,i}^{\mathrm{s}}$ and $\mathcal{C}_{k,i}^{\mathrm{t}}$ are the corresponding estimated covariance matrices associated with each point i of the source or target cloud, respectively. As will be further discussed in Section II-

D, we can initialize the above objective function with a prior supplied by external sensors in an attempt to push the convergence towards a global minimum. That is, for Eq. (3), if a prior $\tilde{\mathbf{X}}_k^{\mathcal{B}}$ is available by means of IMU preintegration, we can set the initial guess $\tilde{\mathbf{X}}_k^{\mathcal{L}} = \tilde{\mathbf{X}}_k^{\mathcal{B}}$ to create a loosely-coupled system. If a prior is not available however, the system reverts to pure LiDAR odometry in which $\tilde{\mathbf{X}}_k^{\mathcal{L}} = \mathbf{I}$ and relies solely on point cloud correspondence matching for this step.

2) Scan-to-Map: After recovering an initial robot motion estimate, a secondary stage of scan-to-map matching is performed and follows a similar procedure to that of scan-to-scan. However, rather than computing a relative transform between two instantaneous point clouds, the objective here is to further refine the motion estimate from the previous step to be more globally-consistent by means of matching with a local submap. In other words, the task here is to compute an optimal transform $\hat{\mathbf{X}}_k^{\mathcal{W}}$ between the current source cloud $\mathcal{P}_k^{\mathrm{s}}$ and some derived submap $\mathcal{S}_k$ such that

$$\hat{\mathbf{X}}_k^{\mathcal{W}} = \underset{\mathbf{X}_k^{\mathcal{W}}}{\arg\min}\ \mathcal{E}\left(\mathbf{X}_k^{\mathcal{W}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{S}_k\right) . \tag{4}$$

After similarly defining the residual error $\mathcal{E}$ from GICP as in Eq. (2), the overall objective function for scan-to-map is

$$\hat{\mathbf{X}}_k^{\mathcal{W}} = \underset{\mathbf{X}_k^{\mathcal{W}}}{\arg\min} \sum_{j}^{M} d_j^\top \left(\mathcal{C}_{k,j}^{\mathcal{S}} + \mathbf{X}_k^{\mathcal{W}}\mathcal{C}_{k,j}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{W}\top}\right)^{-1} d_j , \tag{5}$$

for M number of corresponding points between point cloud $\mathcal{P}_k^{\mathrm{s}}$ and submap $\mathcal{S}_k$, where $\mathcal{C}_{k,j}^{\mathcal{S}}$ is the corresponding scan-stitched covariance matrix for point j in the submap as defined later in Section II-F. Eq. (5) is initialized using the propagated result from scan-to-scan in the previous section from $\mathcal{L}$ to $\mathcal{W}$, i.e. $\tilde{\mathbf{X}}_k^{\mathcal{W}} = \hat{\mathbf{X}}_{k-1}^{\mathcal{W}}\ \hat{\mathbf{X}}_k^{\mathcal{L}}$, so that this prior motion can be compared against historical map data for global consistency. The output of this stage $\hat{\mathbf{X}}_k^{\mathcal{W}}$ is the final estimated robot pose used for downstream modules.

We note here that a key innovation of this work is how we derive and manage our submap for this stage. Whereas previous works create a submap by querying the locality of each individual point in a stored map, we associate scans to keyframes and search rather in keyframe-space to stitch point clouds together and create $\mathcal{S}_k$. The implications of this include a far faster and more consistent submap creation process each iteration, which is additionally more permissive as compared to a radius-based search and will be further discussed in Section II-E.

D. Optimization Prior

Eq. (3) describes the scan-to-scan nonlinear optimization problem and can be initialized with a prior to reduce the chances of converging into a sub-optimal local minima. This prior represents an initial guess of the relative motion between two LiDAR frames and can come from integrating angular velocity measurements from an inertial measurement unit (IMU). More specifically, angular velocity measurements $\hat{\boldsymbol{\omega}}_k$ is defined as $\hat{\boldsymbol{\omega}}_k = \boldsymbol{\omega}_k + \mathbf{b}_k^{\boldsymbol{\omega}} + \mathbf{n}_k^{\boldsymbol{\omega}}$ measured in $\mathcal{B}$ with static bias $\mathbf{b}_k^{\boldsymbol{\omega}}$ and zero white noise $\mathbf{n}_k^{\boldsymbol{\omega}}$ for convenience. After calibrating for the bias, a relative rotational motion of the robot's body between two LiDAR frames can be computed via gyroscopic propagation of the quaternion kinematics $\mathbf{q}_{k+1} = \mathbf{q}_k + (\frac{1}{2}\mathbf{q}_k \otimes \boldsymbol{\omega}_k)\Delta t$. Here, $\mathbf{q}_k$ is initialized to identity prior to integration, Δt is the difference in time between IMU measurements in seconds, and only gyroscopic measurements found between the current LiDAR scan and the previous one are used. Note that we are only concerned with a rotational prior during IMU preintegration and leave the retrieval of a translational prior via the accelerometer for future work. The resulting quaternion of this propagation is converted to an $\mathbb{SE}(3)$ matrix with zero translational component to be used as $\tilde{\mathbf{X}}_k^{\mathcal{B}}$, the scan-to-scan prior.

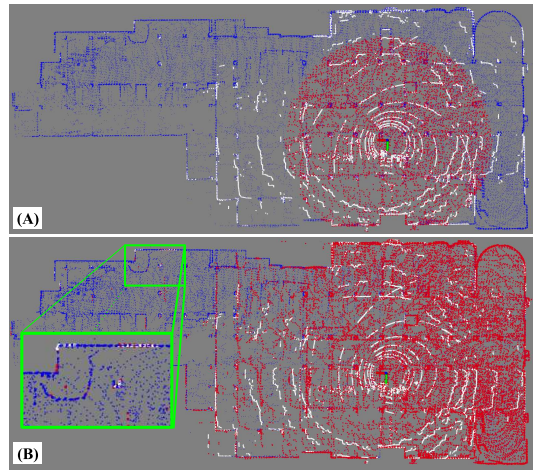

Fig. 3. **Keyframe-based submapping.** A comparison between the different submapping approaches, visualizing the current scan (white), the derived submap (red), and the full map (blue). (A) A common radius-based submapping approach of r = 20m retrieved in point cloud-space. (B) Our keyframe-based submapping approach, which concatenates a subset of keyed scans and helps anchor even the most distant points in the current scan (green box) during the scan-to-map stage.

E. Fast Keyframe-Based Submapping

A key innovation of this work lies in how our system manages map information and derives the local submap in scan-to-submap matching for global egomotion refinement. Rather than working directly with point clouds and storing points into a typical octree data structure, we instead keep a history of *keyframes* to search within, in which each keyframe is linked to its corresponding point cloud scan in a key-value pair. The resulting submap used for scan-to-submap matching is then created by concatenating the corresponding point clouds from a subset of the keyframes, rather than directly retrieving local points within some radius of the robot's current position.

The implication of this design choice is twofold: first, by searching in "keyframe-space" rather than "point cloud-space," a much more computationally tractable problem is obtained. Radius-based searches within a cumulative point cloud map can require distance calculations against hundreds of thousands of points — a process that quickly becomes infeasible even with an incremental octree data structure. Searching against keyframes, however, typically involves

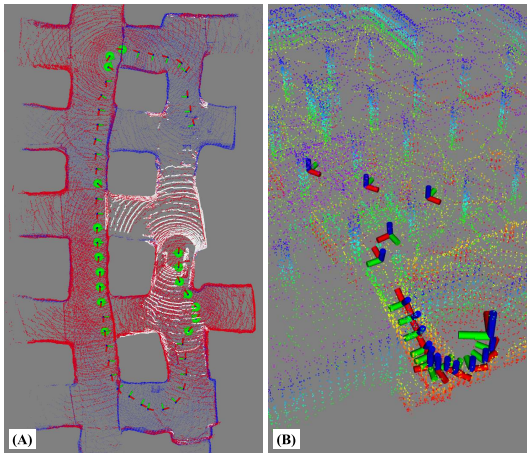

Fig. 4. **Keyframe selection and adaptive thresholds.** (A) Our method's submap (red) is generated by concatenating the scans from a subset of keyframes (green spheres), which consists of K nearest neighbor keyframes and those that construct the convex hull of the keyframe set. (B) An illustration of adaptive keyframing. In this scenario, the threshold decreases when traversing down a narrow ramp to better capture small-scale details.

only a few hundred points even after long traversals and provides much more consistent computational performance, reducing the chances of dropping frames. Additionally, a keyframe-based approach constructs a much more permissive submap as compared to range-based methods. That is, since the size of a submap derived from keyframe point clouds relies solely on the LiDAR sensor's range rather than a predetermined distance, the derived submap can have a larger overlap with the current scan; this is illustrated in Fig. 3. In this example, a submap of fixed radius r = 20m insufficiently overlaps with the current scan and can introduce drift over time due to containing only spatially-nearby points; however, a keyframe-based approach covers most of the current scan which helps with better scan-to-map alignment. Expanding the radius size may help increase this overlap for radius-based methods, but doing so would significantly slowdown subsequent tasks such as the GICP covariance calculations.

1) Keyframe Selection via kNN and Convex Hull: To construct the submap $\mathcal{S}_k$, we concatenate the corresponding point clouds from a selected subset of environmental keyframes. Let $\mathcal{K}_k$ be the set of all keyframe point clouds such that $\mathcal{S}_k \subseteq \mathcal{K}_k$. We define submap $\mathcal{S}_k$ as the concatenation of K nearest neighbor keyframe scans $\mathcal{Q}_k$ and L nearest neighbor convex hull scans $\mathcal{H}_k$ such that $\mathcal{S}_k = \mathcal{Q}_k \oplus \mathcal{H}_k$, where the indices which specify the convex hull are defined by the set of keyframes which make up the intersection of all convex sets containing the keyframes which compose $\mathcal{K}_k$.

The result of this is illustrated in Fig. 4A, in which the keyframes highlighted in green are those that compose the extracted submap, indicated in red. Intuitively, extracting nearest neighbor keyframes aims to help with overlap of nearby points in the current scan, while those from the convex hull — which contain boundary map points — increase the overlap with more distant points in the scan. This combination reduces overall trajectory drift by maximizing scan-to-map overlap and provides the system with multiple scales of environmental features to align with. Note that

TABLE I

SUMMARY OF DATA STRUCTURE RECYCLING

Element	Scan-to-Scan	Scan-to-Map
$\mathcal{T}_k^{\text{source}}$	build	$\xrightarrow{\text{reuse from S2S}}$
$\mathcal{T}_k^{\text{target}}$	$\mathcal{T}_{k-1}^{\text{source}}$	build when $\mathcal{S}_k \neq \mathcal{S}_{k-1}$
$\mathcal{C}_k^{\text{source}}$	compute	$\xrightarrow{\text{reuse from S2S}}$
$\mathcal{C}_k^{\text{target}}$	$\mathcal{C}_{k-1}^{\text{source}}$	$\sum_n^N \mathcal{C}_{k,n}^{\mathcal{S}}$

keyframes which are classified as both a nearest neighbor and a convex hull index are only used once in the submap.

2) Adaptive Keyframing: The location of keyframes affects the derived submap and can subsequently influence accuracy and robustness of the odometry. Keyframe nodes are commonly dropped using fixed thresholds (e.g., every 1m or 10° of translational or rotational change) [4], [6], [13], but the optimal position can be highly dependent on a surrounding environment's structure. More specifically, in large-scale settings, features captured by the point cloud scan are much more prominent and can be depended on for longer periods of time. Conversely, for narrow or small-scale environments, a smaller threshold is necessary to continually capture the small-scale features (i.e., tight corners) in the submap for better localization. Thus, we choose to scale the translational threshold for new keyframes according to some notion of "spaciousness" in the instantaneous point cloud scan, defined as $m_k = \alpha m_{k-1} + \beta M_k$, where M_k is the median Euclidean point distance from the origin to each point in the preprocessed point cloud, $\alpha = 0.95$, $\beta = 0.05$, and m_k is the smoothed signal used to scale keyframe distance threshold th_k at time k such that

$$th_k = \begin{cases} 10\text{m} & \text{if } m_k > 20\text{m} \\ 5\text{m} & \text{if } m_k > 10\text{m} \,\&\, m_k \leq 20\text{m} \\ 1\text{m} & \text{if } m_k > 5\text{m} \,\&\, m_k \leq 10\text{m} \\ 0.5\text{m} & \text{if } m_k \leq 5\text{m} \end{cases} \tag{6}$$

with rotational threshold held fixed at 30°. Fig. 4B illustrates the effects of this adaptive thresholding, which helps with robustness to changing environmental dimension.

F. Algorithmic Implementation

1) Scan-Stitched Submap Normals: Generalized-ICP involves minimizing the plane-to-plane distance between two clouds, in which these planes are modeled by a computed covariance for each point in the scan. Rather than computing the normals for each point in the submap on every iteration (which can be infeasible for real-time operation), we assume that the set of submap covariances $\mathcal{C}_k^{\mathcal{S}}$ can be approximated by concatenating the normals $\mathcal{C}_{k,n}^{\mathcal{S}}$ from N keyframes which populate the submap such that $\mathcal{C}_k^{\mathcal{S}} \approx \sum_n^N \mathcal{C}_{k,n}^{\mathcal{S}}$. As a consequence, each submap's set of normals need not be explicitly computed, but rather just reconstructed by stitching together those calculated previously.

2) Data Structure Recycling: Expanding on the above, several algorithmic steps in current LiDAR odometry pipelines can benefit from data structure sharing and reuse,

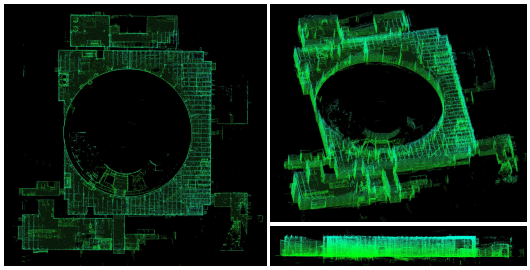

Fig. 5. **Alpha course map.** Different views and angles of the dense 3D point cloud map generated using our DLO algorithm on the Urban Alpha dataset. Estimated positions at each timestamp were used to transform the provided scan into a world frame; this was performed for all scans across the dataset and concatenated / voxel filtered to generated the above images.

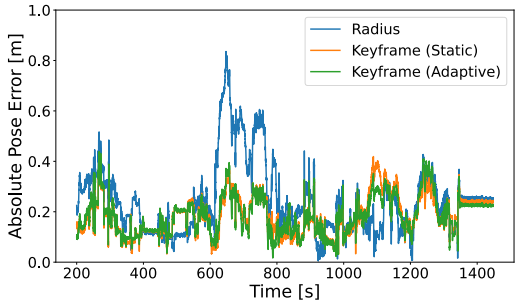

Fig. 6. **Error comparison.** The absolute pose error plotted across a 1200s window of movement, showing the difference between radius and keyframe submapping schemes. Keyframe-based approaches do not have the range restriction that radius-based approaches inherently contain, which directly translates to a lower error in odometry due to more perceptive submapping. Note that adaptive keyframing primarily helps with reliability and robustness to changes in environmental dimension (Fig. 9).

drastically reducing overall system overhead by removing unnecessary and redundant operations. As summarized in Table I, the system requires eight total elements to successfully perform scan-to-scan and scan-to-map matching. This includes kdtrees $\mathcal{T}_k$ used to search for point correspondences and covariance matrices $\mathcal{C}_k$ for GICP alignment for both source and target clouds in each scan-matching process.

Out of the four required kdtrees data structures, only two need to be built explicitly. That is, the tree for the source (input) cloud $\mathcal{T}_k^{\text{source}}$ can be built just once per scan acquisition and shared between both modules (as the same scan is used for both sources). For the scan-to-scan target tree $\mathcal{T}_k^{\text{target}}$, this is simply just the previous iteration's scan-to-scan source tree $\mathcal{T}_{k-1}^{\text{source}}$ and thus can be propagated. The scan-to-map target tree needs to be built explicitly, but since the submap is derived from a set of keyframes, this build only needs to be performed when the set of selected keyframes via our kNN and convex hull strategy changes from one iteration to the next, such that $\mathcal{S}_k \neq \mathcal{S}_{k-1}$. Otherwise, the data structure can just be reused again for additional computational savings. Point covariances $\mathcal{C}_k$ needed for GICP, on the other hand, only need to be computed once per scan aquisition, and its data can be shared directly in the other three instances.

3) Dual NanoGICP: To facilitate the cross-talking between scan-matching modules, we developed NanoGICP, a custom iterative closest point solver which combines the FastGICP [17] and NanoFLANN [18] open-source packages with additional modifications for data structure sharing as

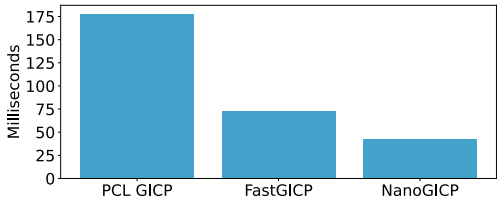

Fig. 7. **Average convergence time.** A comparison of average convergence times across 100 benchmark alignments for each algorithm, including our NanoGICP solver and two other open-source GICP packages.

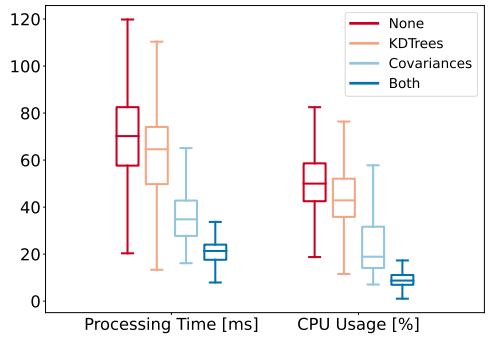

Fig. 8. **Ablation study of data recycling schemes.** Box plots of the processing time and CPU usage for four different data recycling schemes, ranging from no data structure reuse to partial reuse and full reuse.

TABLE II

DROPPED LIDAR SCANS PER RECYCLING SCHEME

	None	KDTrees	Covariances	Both
% Scans	9.37%	4.51%	0.00%	0.00%

described before. In particular, NanoGICP uses NanoFLANN to efficiently build kdtree data structures, which are subsequently used for point cloud correspondence matching by FastGICP. In practice, data structure sharing is performed between two separate NanoGICP instantiations with different hyperparameters — one to target each scan-matching problem — and done procedurally as detailed in Algorithm 1.

III. RESULTS

A. Component Evaluation

To investigate the impact of our system's components, including keyframe-based submapping, submap normal approximation, and the reuse of data structures, we compare each component with its counterpart using the Alpha Course dataset from the Urban circuit of the DARPA Subterranean Challenge. This dataset contains LiDAR scans from a Velodyne VLP-16 sensor, in addition to IMU measurements from a VectorNav VN-100, collected across 60 minutes in an abandoned powerplant located in Elma, WA which contains multiple perceptual challenges such as large or self-similar scenes (Fig. 5). For these component-wise evaluations, data was processed using a 4-core Intel i7 1.30GHz CPU.

1) Keyframe-Based Submapping: We compared the absolute pose error (APE), processing time, and CPU load across three submapping schemes, including: radius-based (r = 10m), keyframe-based with a 1m static threshold, and keyframe-based with adaptive thresholding. For keyframe-based variants, we used 10 nearest-neighbor and 10 convex

TABLE III

COMPARISON ON BENCHMARK DATASETS

Method	Alpha Course (757.4m)				Beta Course (631.5m)				CPU Usage	
	APE [m]			ME [m]	APE [m]			ME [m]	No. of Cores	
	max	mean	std	rmse	max	mean	std	rmse	max	mean
BLAM [12]	3.44	1.01	0.94	0.43	3.89	2.27	0.89	1.27	1.14	0.93
Cartographer [19]	5.84	2.91	1.60	1.05	2.64	1.37	0.67	0.31	1.75	0.88
LIO-Mapping [5]	2.12	0.99	0.51	0.45	1.60	1.18	0.22	0.61	1.80	1.53
LOAM [10]	4.33	1.38	1.19	0.60	2.58	2.11	0.44	0.99	1.65	1.41
LOCUS [13]	0.63	0.26	0.18	0.28	1.20	0.58	0.39	0.48	3.39	2.72
DLO	**0.40**	**0.18**	**0.06**	**0.19**	**0.50**	**0.16**	**0.09**	**0.19**	**0.92**	**0.62**

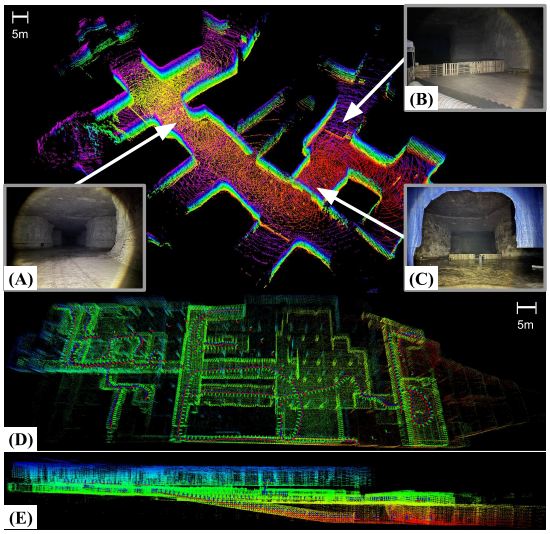

Fig. 9. **Extreme environments.** *Top:* A section of an underground mine in Lexington, KY mapped autonomously using our custom drone while running DLO. This environment contained challenging conditions such as: (A) low illuminance, (B) object obstructions, and (C) wet and muddy terrain. *Bottom:* Top-down (D) and side (E) views of the three levels of an abandoned subway located in Downtown Los Angeles, CA mapped via DLO using a Velodyne VLP-16 on a quadruped. In this run, we manually tele-operated the legged robot to walk up, down, and around each floor for a total of 856m.

hull keyframes for submap derivation. From Fig. 6, the influence of our approach is clear: submapping in keyframe-space can significantly reduce positional error by considering more distant points that would otherwise be outside the scope of a radius-based approach. These additional points influence the outcome of the GICP optimization process as they are considered during error minimization for the optimal transform; this is especially important in purely frontend-based odometry, since any additional error in pose can quickly propagate over time due to drift. Processing time and CPU load showed similar trends: radius-based processed each scan notably slower at 74.2ms per scan with an average of 37.5% CPU load as compared to 21.6ms / 10.2% and 19.1ms / 9.1% for static and adaptive schemes, respectively.

2) Data Structure Recycling: To evaluate the effectiveness of data reusage, we measured and compared the processing time and CPU usage between different recycling schemes via a box plot (Fig. 8) and percentage of dropped scans over the dataset (Table II). In a naive system which explicitly calculates each kdtree and cloud covariance, computation time exceeded LiDAR rate (10Hz for Velodyne) with a high average of 69.8ms per scan and nearly 10% of scans dropped due to high processing time. Recycling kdtrees but not covariances provides a slight improvement in processing time and CPU percentage, while recycling covariances but not kdtrees provides a more prominent performance boost; this is reasonable since our covariance recycling scheme is more aggressive than kdtree reusage. Finally, using the full scheme as detailed in Table I significantly decreases both metrics, with an average processing time of 21.9ms and 9.5% CPU load, which prevents any LiDAR frames from dropping.

3) NanoGICP: To compare NanoGICP with the state-of-the-art, we use FastGICP's [17] benchmark alignment code found in the authors' open-source repository. This benchmark measures the average convergence time to align two LiDAR scans across 100 runs, and we compare against PCL's [20] GICP implementation as well as FastGICP's multithreaded implementation. Note that we do not compare against the voxelized FastGICP variant, since this method approximates planes with groups of planes and decreases overall accuracy. All tested algorithms were initialized with an identity prior, and as shown in Fig. 7, we observed that NanoGICP converged faster on average (42.53ms) when compared to FastGICP (72.88ms) and PCL's GICP (178.24ms).

B. Benchmark Results

The odometry accuracy and CPU load of DLO was compared to several LiDAR and LiDAR-IMU odometry methods — including BLAM [12], Cartographer [19], LIO-Mapping [5], LOAM [10], and LOCUS [13] — using the Alpha and Beta course dataset from the Urban Circuit of the Subterranean Challenge (numbers and ground truth retrieved from [13]). We note that LIO-SAM [6] and LVI-SAM [4], two state-of-the-art tightly-coupled approach, could not be tested at the time of this work due to their sensitive calibration procedure and strict input data requirements. We observed that our method's CPU load was measured to be far lower than any other algorithm, using less than one core both on average and at its peak. This is likely a result how our system derives its submap, in addition to the extensive reuse of internal data structures. This observation can also explain DLO's much lower absolute pose error (APE) and mean error (ME), with similar trends in the relative pose error. With this faster processing time, our method outperformed all other methods in both Alpha and Beta courses, having more than twice the accuracy in the Beta course for max,

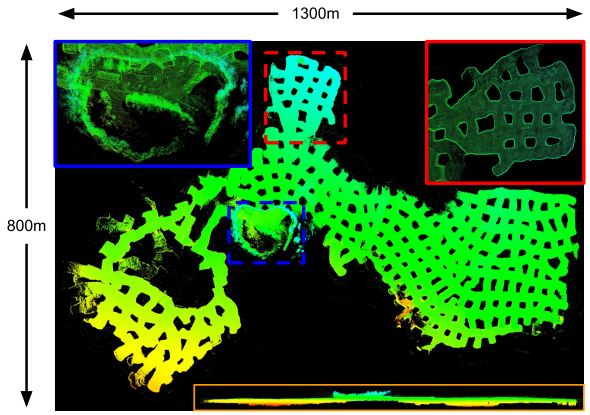

Fig. 10. **Mega Cavern.** Different views of the Mega Cavern in Louisville, KY mapped by our DLO algorithm, with a total estimated trajectory of 9057.66m. Data is courtesy of Team Explorer.

mean and standard deviation, even without motion distortion correction. In addition to our more permissive submapping approach, we are less likely to drop frames than other methods and have the processing capital to match the dense point clouds at a higher resolution.

C. Field Experiments

We additionally tested and implemented our solution on several custom robotic platforms for real-world field operation. Specifically, we integrated DLO onto an aerial vehicle (Fig. 1A) with an Ouster OS1 and a Boston Dynamics Spot (Fig. 1B) with a Velodyne VLP-16. Both systems contained a VectorNav VN-100 IMU rigidly mounted below the base of the LiDAR and processed data on an Intel NUC Board NUC7i7DNBE 1.9GHz CPU. We conducted both manual and autonomous traversals in two perceptually-challenging environments: in an underground limestone cave in Lexington, KY and at an abandoned subway in Los Angeles, CA (Fig. 9). Both locations contained environmental properties which often challenge perceptual systems, including poor lighting conditions, featureless corridors, and the presence of particulates such as dust or fog. Despite traversing over 850m across three different levels in the abandoned subway, our system reported only a 10cm end-to-end drift, largely owing to DLO's robust keyframing scheme which adapted to large and small spaces. Our tests in the underground mine showed similar promise: while this environment lacked any external lighting deep within the cave, DLO could still reliably track our aerial vehicle across 348m of autonomous flight. These results demonstrate the real-world reliability of our method.

IV. Conclusion

This work presented Direct LiDAR Odometry (DLO), a lightweight and accurate frontend localization solution with minimal computational overhead for long-term traversals in extreme environments. A key innovation which distinguishes our work from others is how we efficiently derive a local submap for global pose refinement using a database of keyframe-point cloud pairs. This in turn permits a substantial number of solver data structures to be shared and reused between system modules, all of which is facilitated using our custom NanoGICP cloud registration package. We demonstrate the reliability of our approach through benchmarks and extensive field experiments on multiple platforms operating in large-scale perceptually-challenging environments, and we invite others to use and evaluate our open-source code. DLO was developed for and used on NASA JPL's Team CoSTAR's fleet of quadrotors in the DARPA Subterranean Challenge (Fig. 10), and in the future we are interested in tighter IMU integration as well as motion distortion correction.

Acknowledgements: The authors would like to thank Team CoSTAR teammates and colleagues, including Amanda Bouman, Luca Carlone, Micah Corah, Kamak Ebadi, Seyed Fakoorian, David Fan, Sung Kim, Benjamin Morrell, Joshua Ott, Andrzej Reinke, Toni Rosinol, and Patrick Spieler, for their valuable insight and productive discussions.

References

[1] C. Cadena, L. Carlone *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, 2016.
[2] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *International Conference on Intelligent Robots and Systems*, 2018.
[3] K. Ebadi, Y. Chang *et al.*, "Lamp: Large-scale autonomous mapping and positioning for exploration of perceptually-degraded subterranean environments," in *IEEE International Conference on Robotics and Automation*, 2020.
[4] T. Shan, B. Englot *et al.*, "Lvi-sam: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping," in *IEEE International Conference on Robotics and Automation*, 2021.
[5] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *International Conference on Robotics and Automation*, 2019.
[6] T. Shan, B. Englot *et al.*, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.
[7] N. Bhatia, "Survey of nearest neighbor techniques," *International Journal of Computer Science and Information Security*, 2010.
[8] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, 1992.
[9] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: Science and Systems (RSS)*, 2009.
[10] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, 2014.
[11] Z. J. Yew and G. H. Lee, "3dfeat-net: Weakly supervised local 3d features for point cloud registration," in *Proceedings of the European Conference on Computer Vision*, 2018.
[12] E. Nelson, "B(erkeley) l(ocalization) a(nd) m(apping)." [Online]. Available: https://github.com/erik-nelson/blam
[13] M. Palieri, B. Morrell *et al.*, "Locus: A multi-sensor lidar-centric solution for high-precision odometry and 3d mapping in real-time," *IEEE Robotics and Automation Letters*, 2020.
[14] W. Xu and F. Zhang, "Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter," *IEEE Robotics and Automation Letters*, 2021.
[15] W. Xu, Y. Cai *et al.*, "Fast-lio2: Fast direct lidar-inertial odometry," *arXiv preprint arXiv:2107.06829*, 2021.
[16] A. Agha, K. Otsu *et al.*, "Nebula: Quest for robotic autonomy in challenging environments; team costar at the darpa subterranean challenge," *Journal of Field Robotics*, 2021.
[17] K. Koide, M. Yokozuka *et al.*, "Voxelized gicp for fast and accurate 3d point cloud registration," in *IEEE International Conference on Robotics and Automation*, 2021.
[18] 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.
[19] W. Hess, D. Kohler *et al.*, "Real-time loop closure in 2d lidar slam," in *IEEE International Conference on Robotics and Automation*, 2016.
[20] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation*, 2011.