2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Efficient Dynamic LiDAR Odometry for Mobile Robots with Structured Point Clouds

Jonathan Lichtenfeld, Kevin Daun and Oskar von Stryk[1]

***Abstract*— We propose a real-time dynamic LiDAR odometry pipeline for mobile robots in Urban Search and Rescue (USAR) scenarios. Existing approaches to dynamic object detection often rely on pretrained learned networks or computationally expensive volumetric maps. To enhance efficiency on computationally limited robots, we reuse data between the odometry and detection module. Utilizing a range image segmentation technique and a novel residual-based heuristic, our method distinguishes dynamic from static objects before integrating them into the point cloud map. The approach demonstrates robust object tracking and improved map accuracy in environments with numerous dynamic objects. Even highly non-rigid objects, such as running humans, are accurately detected at point level without prior downsampling of the point cloud and hence, without loss of information. Evaluation on simulated and real-world data validates its computational efficiency. Compared to a state-of-the-art volumetric method, our approach shows comparable detection performance at a fraction of the processing time, adding only 14 ms to the odometry module for dynamic object detection and tracking. The implementation and a new real-world dataset are available as open-source for further research.**

I. INTRODUCTION

The capability to localize itself and create a map of the environment is essential to enable autonomous navigation and other assistance functions with mobile robots in unknown disaster environments without GNSS reception. The problem is referred to as Simultaneous localization and mapping (SLAM). Many established approaches assume a static world, which is usually not the case in challenging Urban Search and Rescue (USAR) environments. Often, there are rescue workers, vehicles, or other robots in proximity. The need for dealing with such objects is motivated by three main aspects. Firstly, for autonomous navigation, obstacle avoidance has to be ensured to improve the semantic understanding of the environment and enable better autonomous functions, such as the consideration of dynamic objects in motion planning and control. Secondly, dynamic objects need explicit handling for mapping as otherwise, they typically create "ghost trace" artifacts in the map and can potentially impede the quality of the localization in highly dynamic environments. Thirdly, as a step towards human-machine-interaction, an understanding of the dynamic semantics in the environments enables novel approaches to human-robot interaction with the rescue teams or possible victims. For instance, a robot could follow a rescue worker walking in front of it or lead a potential victim to a safe area.

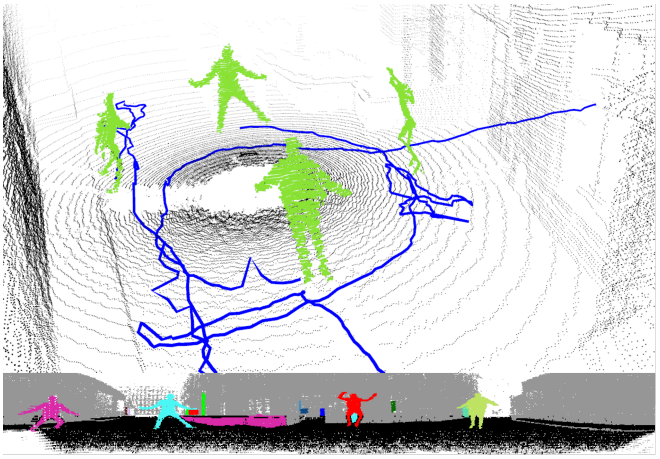

Fig. 1. Application of the proposed approach for dynamic LiDAR odometry, enabling the efficient segmentation and tracking of highly articulated objects. Top: A group of jumping persons is detected and tracked in a highly dynamic sequence. Dynamic points are shown in green, and trajectories are indicated in blue. Bottom: The range image used for object segmentation of the above cloud. Different colors represent different static and dynamic objects.

Existing SLAM approaches considering the environment dynamics can provide very accurate environment segmentations but are typically either limited by computational expense, no online capability, or inaccurate representations of highly articulated objects. For example, the post-processing method ERASOR [1] removes ghost traces from the map after the SLAM process, thus, no online handling of the objects is possible. Learning-based methods that have become widely used in the context of autonomous driving [2]–[4] allow the accurate online detection of (moving) objects but are computationally expensive and fail on instances of classes that are not included in the training data, rendering them unsuitable for USAR applications where potential objects are not necessarily known beforehand.

This work proposes a new approach to dynamic Light Detection and Ranging (LiDAR) odometry, which addresses the aforementioned limitations. In contrast to other recent works, we follow a grid-free approach and operate directly on a structured point cloud, allowing for faster processing of high-resolution scans even in large-scale environments. As a backbone, we use the state-of-the-art LiDAR odometry method proposed by [5]. Working on the range image, the

All authors are with the Simulation, Systems Optimization and Robotics Group, Technical University of Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany. {lichtenfeld, daun, stryk}@sim.tu-darmstadt.de

Research presented in this paper has been supported in parts by the German Federal Ministry of Education and Research (BMBF) within the DRZ project (grant no. 13N16475), and the KIARA project (grant no. 13N16274), and by the LOEWE initiative (Hesse, Germany) within the emergenCITY center.

point cloud is segmented, and objects are tracked individually so that they can be used in other processes, too. Based on the observation that moving objects exhibit higher residuals in the scan matching process than static ones, we propose a novel heuristic to distinguish between the two classes. Dynamic objects can then easily be removed from the scan before integrating it into the map.

Our approach is designed to consider the specific requirements [6] for USAR applications, which implies that its mechanisms are as general as possible and rely only on few assumptions about the environment. The main focus is on obtaining a light-weight, comprehensible system that runs in real-time on ground-based robots using a single high-resolution LiDAR sensor. In summary, the main contributions of this work are:

- A lightweight end-to-end pipeline for dynamic LiDAR odometry, including the components odometry, object detection, tracking, and mapping
- A novel residual-based heuristic to distinguish dynamic from static objects
- Evaluation on a public benchmark and created real-world data
- The open-source release of the code and dataset[1].

II. RELATED WORK

Most LiDAR Odometry and Mapping methods treat dynamic objects as outliers in the scan matching process [5], [7], [8], which tends to worsen the odometry estimate and thus the map quality. A complementary aspect to this problem is the Detection and Tracking of Moving Objects (DATMO). Although the majority of publications cover only either of them [5], [7]–[9], DATMO can be integrated into the SLAM process, leading to higher fidelity maps and more accurate pose estimates in dynamic environments. At the same time, a precise map helps in providing self-localization and detecting objects. Wang *et al.* [10] were the first to propose a framework that integrates both problems in a complementary manner by detecting moving objects in each scan before obtaining the pose update from a SLAM module. Since tracking of dynamic objects is well studied in the computer vision community and robust off-the-shelf multi-object trackers exist, a key challenge of DATMO lies in the detection of moving objects in the first place.

Following the classification by Yoon *et al.* [11], moving object detection techniques can be categorized into three groups: model-based detectors, methods that utilize a priori known maps, and approaches that rely on the most recent data only. Model-based, or class-specific, methods can provide accurate detections for predefined object classes in both camera images [12] and point clouds [2], [4]. Considering only frame-wise detections, one cannot tell which objects are moving. Semantic labels, however, can be used to distinguish possibly moving and most likely static objects. For example, vehicles, animals, and pedestrians belong to the former category, while buildings, traffic lights, and trees belong to the latter. Nevertheless, purely model-based classifiers fail to differ between temporarily static and actually moving objects, such as parked and driving cars. Especially in USAR environments, objects of many different classes are present, which can hardly all be represented in training data.

In some use cases for mobile robots, a map of the static environment already exists. This applies, for instance, to autonomous forklifts deployed in warehouse logistics or surveillance robots in buildings. If a model of the building is given or the robot has captured a motion-free map before, it is relatively easy to detect discrepancies in the robot's perception and the stored data using change detection techniques [13]. These discrepancies can indicate moving objects.

Map-free methods can only rely on the most recent sensor data. A way to detect changes without a given map is the online construction of occupancy grids. Typically, the space is discretized into voxels and each voxel is assigned a probability of occupancy based on ray-tracing operations [14]. Postica *et al.* [15] then derive moving objects from inconsistencies between free space and occupied space. The work by Schmid *et al.* [9] is based on a similar principle, but builds on the Truncated Signed Distance Fields (TSDF) framework Voxblox [16] for a more efficient map representation. This approach requires highly accurate localization information, which in turn is affected by the presence of moving objects. [17] considers both short-term and long-term dynamics by constructing a novel dense spatio-temporal representation of the robot environment. However, it focuses on indoor RGB-D applications, incorporating also semantic information that is difficult to obtain with LiDAR sensors. Dewan *et al.* [18] use motion models derived from subsequent scans for dynamic detection. Employing Random Sample Consensus (RANSAC), they estimate motion models for the static scene and dynamic objects. Subsequently, each point is assigned to a motion model following a Bayesian approach. Their evaluation on real world scenes shows high detection and tracking accuracy. However, the proposed motion models are incompatible with non-rigid objects, resulting in a failure to detect articulated objects such as humans or animals. Besides, no information on the used hardware or processing times is given, which also applies to a similar work by Moosmann *et al.* [19]. A follow-up work by the same authors computes dense scene flow improved by incorporating a deep neural network [20]. Wang *et al.* [21] propose a vertical voxel occupancy descriptor to discriminate static and dynamic regions. Underwood *et al.* [22] compare two point clouds and include free space reasoning of unmeasured areas by incorporating ray tracing. This procedure is facilitated by using a spherical coordinate space for the point representation [23], [24]. Dynamic detection across a multi-session map has been studied in [25], also using ray casting and a voxelized point cloud represented as a kd-tree.

Lastly, there are methods that remove dynamic points during post-processing [1], [26], using not only past knowledge but also future scans. Typically, the processing time per scan far exceeds the sensor's frame rate. This renders them

[1]https://github.com/tu-darmstadt-ros-pkg/dynamic_direct_lidar_odometry

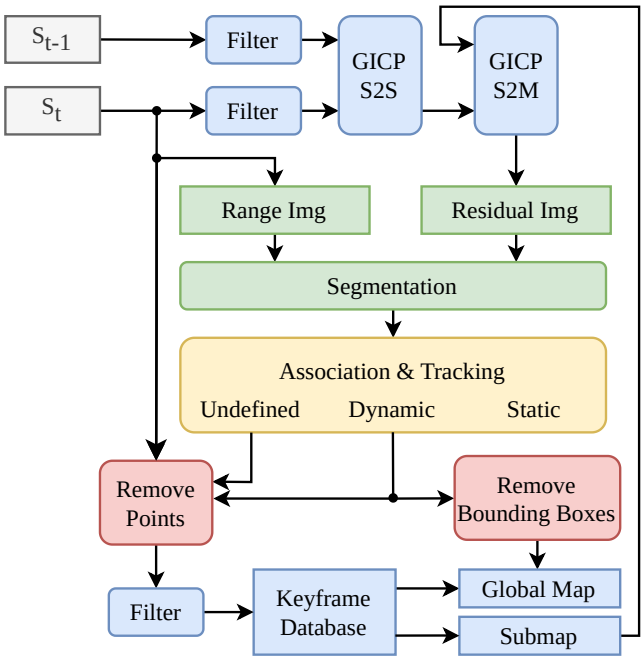

Fig. 2. System overview. Two consecutive scans are registered using the odometry module (blue). The detection module (green) segments the projected range image and residual image into individual objects. The Tracking and Association module (yellow) assigns a dynamic state to each object. Non-static points are removed from the current scan before integrating it into the keyframe database. Ghost traces are removed from the global map (red) based on the objects' bounding boxes.

unsuitable for real-time applications, despite their capability to build highly accurate static maps.

The work by Mao *et al.* [27] is the most similar to ours, since it also comprises a combined odometry and detection approach. The main difference is that they apply a spatial region growing algorithm for clustering points. This increases the computation time for high-resolution point clouds if no down-sampling is applied. Dynamic object detection is then based on the comparison of the bounding box overlap between the current and the previous frame. We assume that this approach struggles with slow-moving objects, as the bounding box overlap is too high to detect them, or with ambiguous associations between scans with multiple objects close to each other. We address this issue by tracking objects over longer periods.

This work uses new scans and, to a small extent, the online constructed map to detect moving objects in previously unknown surroundings. As one of few works only, we exploit the data structure of modern LiDAR sensors for speeding up the object detection. Some weak assumptions about the model's shapes are made, hence it is not completely model-free. Instead of a dense map representation [9], [17], we choose raw point clouds to maintain efficiency even for large-scale scenes.

III. METHOD

We propose a novel approach to dynamic LiDAR odometry with a focus on identifying and tracking distinct dynamic objects. The concept can be divided into an odometry, detection, tracking and point removal modules that influence one another. A complete overview is given in Fig. 2.

A. Overview

Input to the pipeline is a series of laser range scans. We build upon and extend the LiDAR odometry method DLO proposed by Chen *et al.* [5] which employs a two-stage Generalized Iterative Closest Points (GICP) approach for fast registration. The first stage computes an initial guess of the robot's motion by aligning two consecutive scans $\mathcal{S}_{t-1}$ and $\mathcal{S}_t$. The obtained transformation is then used as a prior for the second stage, which aligns $\mathcal{S}_t$ to a submap $\mathcal{M}_t$ for fine registration. Both source and target cloud are stored as kd-trees for efficient nearest neighbor searches. For more details, we refer the reader to the original work. The remaining residuals after convergence are projected onto an image, which in turn is combined with the range image to segment the input into individual geometric objects. The Tracking and Association module then updates a Kalman filter for each object, associates new detections to existing objects, and assigns a dynamic state to each object. Finally, we remove those points in the current scan referring to segments labeled as non-static before integrating the scan into the keyframe database. The latter is used to generate submaps and the global map.

B. Detection

To detect dynamic elements in the scene, we first detect relevant segments as objects by performing clustering with a range image representations of the scan cloud. Then, we utilize the scan matching residuals to classify the segments into dynamic and static.

Range Image-based Segmantation: Geometric segmentation of point clouds is expensive because spatial clustering requires nearest neighbor searches, which is costly for large point clouds. Instead, we employ the established range image segmentation by Bogoslavskyi *et al.* [28]. First, the point cloud P is projected onto a cylindrical image I by a mapping $\Pi\colon \mathbb{R}^3 \to \mathbb{R}^2$. Each point $\mathbf{p}_i = (x, y, z)^{\mathrm{T}} \in P$ is assigned to an image coordinate (u, v) via

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \lfloor \left[f_{up} - \arctan\left(\frac{z}{\sqrt{x^2+y^2}} \right) \right] \alpha_v^{-1} \rfloor \\ \lfloor \operatorname{atan2}(x, y)\, \alpha_h^{-1} \rfloor \end{pmatrix} , \tag{1}$$

where f_{up} denotes the upper aperture angle, α_h the horizontal and α_v the vertical sensor resolution, respectively.

This approach can be applied to any point cloud, regardless of their sparsity and point order. The image pixels store the range, i.e. the distance of the regarding point to the sensor

$$I(u(x, y, z), v(x, y, z)) = \sqrt{x^2 + y^2 + z^2} \ . \tag{2}$$

Pixels without a belonging point are marked as invalid. If the point cloud is structured with height h and width w, each point $\mathbf{p}_i$ directly represents a pixel. Assuming a row-major

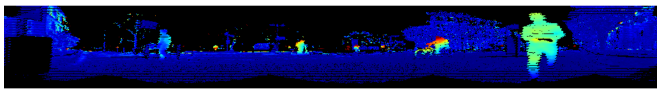

Fig. 3. Residual image as obtained from the GICP algorithm. For better visibility, the registration was performed on the original point cloud, whereas the method uses the downsampled and filtered cloud. A rainbow color map highlights the points with higher residuals. Black pixels represent invalid points.

Algorithm 1 Average Residuals Computation

Input: Segment S, residual image R_{ICP}
Output: Average of residuals r_{avg}

```
n ← 0, r ← 0
for (u, v) in S do
    if R_ICP(u, v) ≠ 0 then
        r ← r + R_ICP(u, v)
        n ← n + 1
return r/n if n > 0 else 0
```

layout, the mapping simplifies to

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \lfloor \frac{i}{w} \rfloor \\ i \mod w \end{pmatrix} . \tag{3}$$

Prior to the object segmentation, we apply a coarse ground removal to the range image as proposed in [8]. The remaining points are clustered into segments as described in [28]. Starting from the upper left point in the range image with label 1, all direct neighbors of that point are considered candidates for this segment. Since the range image is a cylindrical projection, the left and right image edges are wrapped. If a candidate fulfills a heuristic condition and is not yet labeled as part of a segment, ground, or invalid, it is added to a queue and receives the label of the current segment. All points in that queue are treated this way, iteratively, until the queue is empty. In this case, the label is increased by 1 and the process is repeated, starting with the next unlabeled point. More information about the chosen heuristic can be found in [28].

Scan-Matching Residual-based Classification: The above segmentation technique usually yields many segments, depending on the complexity of the environment and chosen parameters. To classify between static and dynamic objects at the tracking stage, we propose the use of so-called residual images. They are built from the residuals of the GICP algorithm implemented in [5]. It iteratively searches for nearest neighbor correspondences between two point clouds and finds a transformation $\mathbf{T}$ that minimizes a cost function such that

$$\mathbf{T} = \arg\min_{\mathbf{T}} \sum_i {d_i^{(\mathbf{T})}}^T \left(C_i^{\mathrm{t}} + \mathbf{T} C_i^{\mathrm{s}} \mathbf{T}^T \right)^{-1} d_i^{(\mathbf{T})} \tag{4}$$

with the covariance matrices of the target and source point C_i^{t} and C_k^{s}, respectively, and the distance $d_i^{(\mathbf{T})} = p_i^t - \mathbf{T} p_i^s$ between the corresponding points. After convergence, a residuum remains for each point in the source cloud, which expresses the Euclidean distance to its nearest neighbor in the target cloud. Considering the scan-to-map module, the source is the current scan, and the target is the current submap. The idea in the following is that moving objects are not represented in the submap, which serves as the target cloud. Therefore, points belonging to these objects tend to have higher residuals compared to the rest of the points, as their distance to the nearest neighbor is greater. We project the point-wise residuals onto an image R_{ICP} (see Fig. 3) using Equation (1). Due to the sparsity of the preprocessed point cloud used for registration, most pixels remain empty. The average residuals for each segment can be computed by averaging the pixel values within the segment's mask, as shown in Algorithm 1. These average residuals are stored for each segment and are used in the dynamic state update step of the tracking pipeline, which will be explained in the next section. One key advantage of these residual images is that they are a byproduct of the odometry module and require minimal additional computation, making them easy to integrate into the pipeline.

C. Tracking and State Update

1) Tracking and Association: It is crucial that objects are tracked in the global frame S_W, because movements parallel to the robot appear static in the robot's frame S_R. At each time step, the detection module provides a set of detections. Since each image pixel refers to a unique point in the cloud, we can directly recover their 3D shapes from the point cloud. For an effective and compact state representation, we compute the object's bounding box through Principal Component Analysis (PCA) and store the point indices as well as the average residuum of the segment. A Kalman filter is then initialized for each object, which internally uses a 10-dimensional state representation consisting of the object's position, rotation around the z-axis, bounding box dimensions and velocity. At each time step, the filters are updated with the associated detections. Upon receiving new detections, we associate them with existing objects, using the approach proposed by Weng *et al.* [29] which consists of Kalman filter updates, data association and a Birth-Death memory. Data association is based on the Hungarian algorithm [30] which minimizes the sum of the costs of the associations. We assign the cost $c_{i,j}$ between the ith tracked object and jth detection

$$c_{i,j} = \alpha \cdot (1 - d_1(BB_i, BB_j)) + \beta \cdot d_2(|p_i|, |p_j|) \tag{5}$$

where α and β are weights, d_1 is the Intersection over Union (IoU) between the bounding boxes and $d_2 = 1 - \frac{\min(|p_i|, |p_j|)}{\max(|p_i|, |p_j|)}$ relates the number of points in both objects. Then, we update each object's Kalman filter with the associated detection. Objects without matches for n consecutive steps are removed.

2) Dynamic State Update: After the data association step, we update the state of each object (see Fig. 4). An object can be in one of three states: *undefined*, *static*, or *dynamic*. The default state for new objects is *undefined*. If it does not satisfy the conditions for a state transition to *dynamic* within n_{max} detections, we mark it as *static*. To transition from *static* or *undefined* to *dynamic*, we check the following conditions:

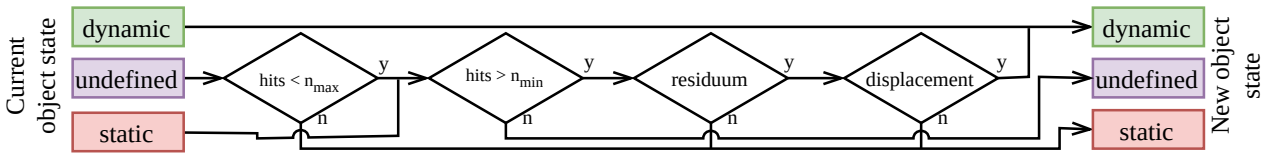

Fig. 4. Dynamic state update. At each time step, all tracked objects are updated and can change their dynamic state based on the number of detections (hits), their average residuum and the displacement from their origin.

First, a minimum number of n_{min} detections is required to avoid false positives. Then, the average residual of the object's segment in the residual image R_{ICP} is considered. We observe that the average residual of an object tends to increase with its height, as the closest corresponding points are often ground points. Therefore, using a static threshold would either exclude all small objects if set too low or include almost all objects if set too high. Instead, we check for each object if it fulfills the condition

$$r_{avg} \geq \theta_{res} h_S \tag{6}$$

where h_S is the range of z-coordinates belonging to the segment and θ_{res} is a heuristic threshold. Lastly, we check if the object has moved by at least θ_{disp} meters from the start position where it was detected for the first time. If all these conditions are met, the object is marked as *dynamic* and keeps this attribute for its lifetime. This is to reliably filter objects out of the map that are not permanently moving, e.g. a car that stops at a traffic light for several time steps. Another state, e.g. *temporarily static*, could be introduced to handle such cases but is not considered in this work.

D. Dynamic Points Removal

The tracking module forwards the indices of all points that belong to dynamic objects to the odometry module. These points can directly be removed from the transformed input scan. Afterward, the filtered and downsampled cloud is added to the keyframe database from which the submaps are generated. To prevent ghost traces in the global map caused by initially static objects, we maintain a rolling window history of the bounding boxes of all objects. If a static object turns dynamic, we remove all points from the global map that are within these bounding boxes. As the global map is used for visualization only, this step is not time-critical for the odometry computation and is therefore performed asynchronously. Note that dynamic points are removed *after* the scan-to-scan registration step, which means that they are present in the GICP module's source cloud but not in the target cloud. This is due to the fact that the segmentation step is performed on the input cloud after transforming it into the global frame. The required transform is obtained as the result of the registration, which leads to a "chicken-egg-problem". One could use the registration result as a first guess, then perform the detection and point removal on the transformed cloud, and finally repeat the registration with the cleaned input cloud. However, this would require a second costly kd-tree generation for the new cloud. The presented procedure shows promising results despite this inaccuracy.

IV. EVALUATION

The evaluation is performed on real-world data and a simulated public benchmark and results are compared to Dynablox [9]. We focus on the accuracy of both detection and tracking of moving objects. Since close- and mid-range objects are most relevant for rescue robots, we highlight them in the results. Furthermore, the influence of moving objects on the constructed map and the processing time is investigated.

A. Experimental Setup

We use a four-wheeled differential drive robot for the data recording, which is equipped with an Ouster OS-0 128 LiDAR sensor mounted on top for an occlusion-free 360° view. It provides structured point clouds of size 128×1024 at a rate of 10 Hz. The experiments are conducted on a consumer-grade laptop with an Intel Core i7-10875H CPU and 32 GB of RAM and are therefore representative for applicability on mobile robots.

B. Datasets

To the author's best knowledge, only few public benchmark datasets with structured undistorted point clouds are available, of which very few were recorded on mobile ground robots in typical robotic environments. Therefore, we recorded our own dataset *kantplatz*, which is publicly available[2]. The entire trajectory is about $500\,\mathrm{m}$ long and has a duration of $480\,\mathrm{s}$. It is recorded in an urban environment with a variety of moving objects, such as pedestrians, cyclists, cars, and buses. Two persons are walking close to the robot during the entire recording, which serves for the evaluation of the detection and tracking performance. To compare on public benchmark data, we also evaluate our method on the *small town simulation* sequence of the DOALS dataset [31]. It exhibits simple geometric shapes and more complex rigid objects such as animals that move along predefined looped trajectories. The point cloud structure is 64×2048. We changed the original layout from column-major to row-major format to meet the requirements of our method.

[2]https://tudatalib.ulb.tu-darmstadt.de/handle/tudatalib/4303

C. Point Classification Accuracy

For better comparability, we limit the detection range of both approaches to 25 m, i.e. farther objects are not considered and only points within that range are included in the evaluation. However, a direct comparison is made difficult by the large number of parameters in both approaches. We have adjusted them manually with reasonable effort to achieve optimal results.

On the real-world data, our method is able to detect and track most of the moving objects, as shown qualitatively in Fig. 6. Even fast, non-rigid motions are detected with high accuracy (Fig. 8) whereas Dynablox often detected only segments of the objects. Newly detected dynamic objects transition from *undefined* to *dynamic* after five time steps on average, depending on their velocity and the chosen thresholds. This is a good compromise between the detection of slow moving objects and the filtering of false positives such as static objects or noisy sensor readings. Storing the history of static objects is particularly useful for the detection of slow moving objects, as they are detected as *dynamic* only after they have moved a certain distance. If they have been labeled as *undefined* in the previous scans, the detection module removes them from the scan before integrating it into the map. If they have been marked as *static*, the history is used to reliably remove them from the map after they have been detected as dynamic. This is particularly important for the beginnings of the recordings, where some objects are not yet moving, but it can also lead to the removal of few false positive points inside the respective bounding boxes.

We quantitatively compare the detected dynamic points from the *small town simulation* sequence to the ground truth annotations. The average IoU, precision, and recall (0.48/0.78/0.49) stay behind the results of Dynablox (0.69/0.99/0.69). We explain this mainly by the fact the dynamic objects in the sequence move in repetitive patterns, i.e. they cross the same area multiple times. If a distant object is not detected due to its sparse point cloud representation, it can lead to traces in the submap. These traces considerably reduce the residuals of objects at that location at a later time, which in turn prevents them from being detected as dynamic. In contrast, if we omit the residuum check and base the dynamic detection solely on the distance that the object has moved, we risk FP dynamic detections. This is because depending on the angle from which the object is observed, its bounding box can vary significantly between two scans, thus leading to an apparent movement. Fig. 5 shows the trade-off between FP and FN point detections for different residuum thresholds. Unsurprisingly, a volumetric mapping approach like Dynablox is more robust to these issues, as it can rely on the entire submap to classify each point.

D. Tracking and Map Quality

Evaluation of the *kantplatz* sequence shows that the tracking performance is very robust. The two persons following the robot are tracked throughout the entire sequence of about 480 s duration. For one of them, the trajectory is interrupted three times, i.e. the ID changes. This occurred twice when the

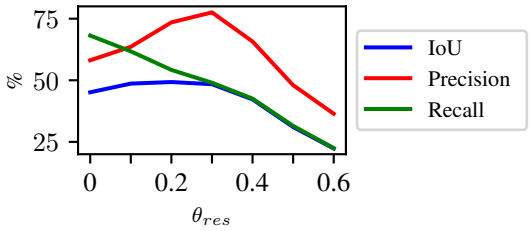

Fig. 5. IoU, precision, and recall on the DOALS *small town simulation* sequence for different values of the residuum threshold (see Eq. 6).

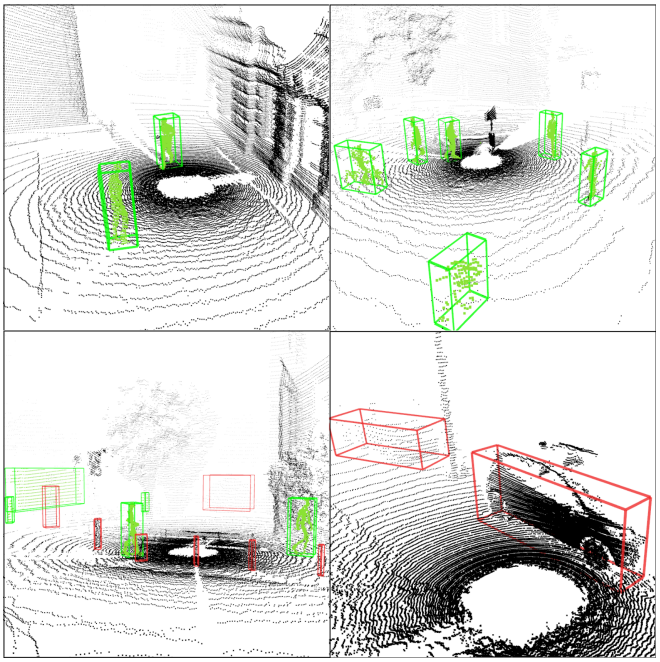

Fig. 6. Segmentation examples of the *kantplatz* dataset. Dynamic objects indicated by green bounding boxes and points, static objects by red bounding boxes. Top: Several pedestrians and cyclists are detected. Bottom left: Both dynamic and static objects, such as street posts. A passing van on the left is detected, another car in the background is wrongly classified as static. Bottom right: Parked cars detected as static.

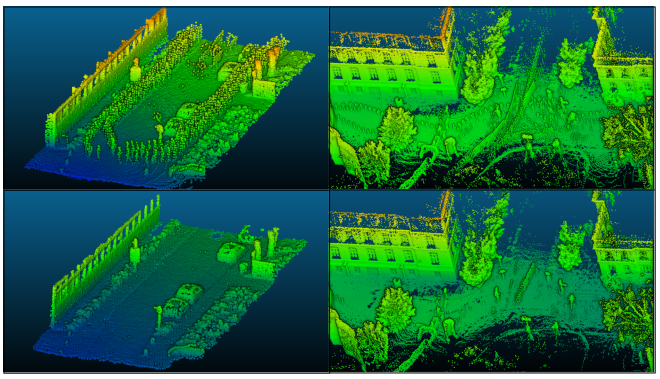

Fig. 7. Comparison of the mapping results on the *kantplatz* dataset. Top: Pure DLO [5] without dynamic object handling, bottom: our approach.

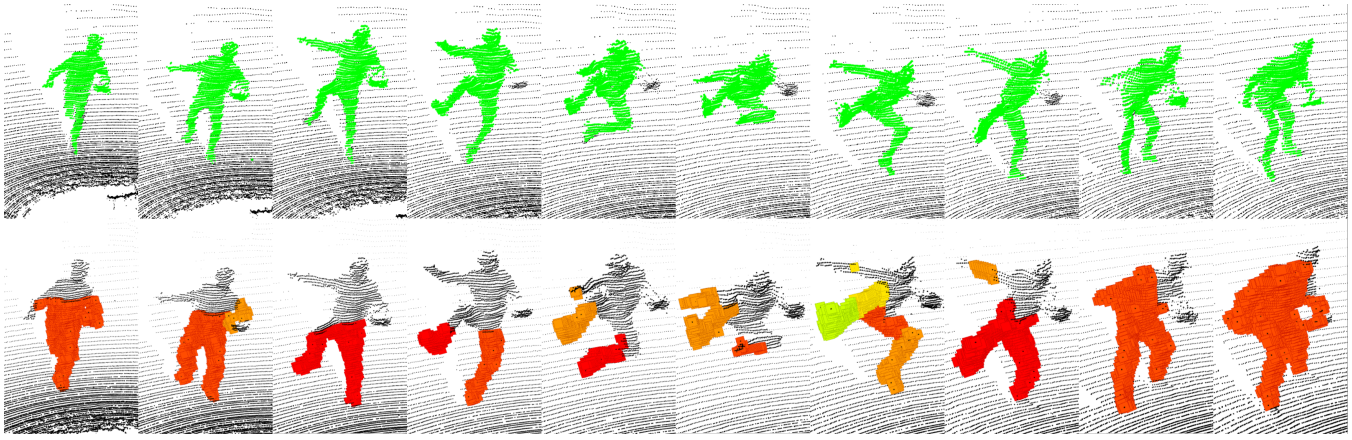

Fig. 8. Dynamic scene from the *kantplatz* dataset. Our approach (top) accurately detects the jumping person, only one arm is partially cut off. Dynablox (bottom) detects only parts of the person and tends to over-segment it.

person was occluded for several scans, and once when both persons were standing close to each other and appeared as one segment in the range image. Fast and abrupt movements, such as jumping or running, are tracked well. As a stress test, a group of four persons running in circles around the robot is tracked correctly (Fig. 1). Consequently, the map quality is improved by the accurate tracking (Fig. 7). Some ghost traces are still visible in the map that mostly arise from remote objects. This could be tackled by tuning the parameters of the image segmentation. In terms of localization accuracy, we did not find any significant differences compared to pure DLO. Despite the large number of moving objects, the ratio of dynamic to static points is too low to have a strong influence on localization. This aspect could be specifically investigated in a follow-up study.

E. Processing Time Analysis

As one of our main motivations is to provide a lightweight solution, we analyze the processing time of our method by its algorithmic components (Table I). On average, the pipeline takes $46.0\,\mathrm{ms}$ to process one scan from the *kantplatz* sequence, which equals a frame rate of 21.7 Hz. We emphasize that this includes the computation of the odometry, which is the most time-consuming part and depends strongly on the down-sampling of the point cloud. The initial filtering reduces the number of points to 5% of the original size, which is sufficient for the scan matching process. The overhead of the dynamic object detection is only $14.3\,\mathrm{ms}$ on average. As Fig. 9 shows, the dynamic detection time is almost constant, while the odometry time increases slightly with the number of scans. Looking at the individual components, we notice that the tracking time depends on the number of tracked objects, but is negligible. Due to the structure of the point clouds, the projection time is also small. The dense volumetric mapping approach used by Dynablox requires significantly more processing time and did not run in real time in our experiments. It should be noted, though, that the dense voxel map created by Dynablox, which contributes to the high computation cost, provides more valuable information regarding scene understanding than our raw point cloud map.

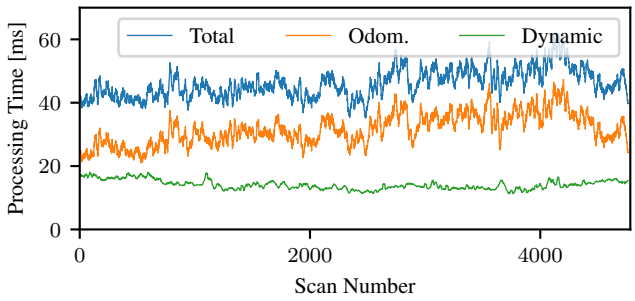

Fig. 9. Processing time analysis of the *kantplatz* sequence. Smoothed with a filter kernel of size 10 for better readability.

TABLE I
PROCESSING TIME [MS] FOR A SINGLE SCAN FROM AN OUSTER LIDAR SENSOR

	kantplatz (OS-0 128)	small town simulation (OS-1 64)
odometry	31.7 ± 12.7	31.7 ± 11.4
projection	4.2 ± 0.8	3.5 ± 0.7
segm.	9.0 ± 1.3	4.4 ± 0.8
tracking	1.1 ± 0.6	0.3 ± 0.3
total	46.0 ± 12.9	40.0 ± 11.5
Dynablox	138.2 ± 27.0	111.0 ± 12.2

F. Limitations

The most frequent failure case is the detection of false positives. As explained before, they can arise from the changing bounding boxes of static objects viewed from different angles. A more robust dynamic threshold could improve this, but carries the risk of delayed detection of actually moving objects. Another source of false positives are apparent motions caused by the robot's own motion: If the robot passes by an opening, the captured segment of a wall behind it seems to move opposite to the robot's motion. This special case is handled better by occupancy grid-based methods. In areas that are crossed by objects multiple times,

the detection of dynamic objects is less reliable. Once an undetected object leaves a trace in the submap, it deteriorates the residuals of other objects at that location, which in turn prevents them from being detected as dynamic.

The method works best for structured point clouds that provide dense range images. In theory, unstructured point clouds could be used as input as well, but tests show that the range image projection and segmentation suffer in this case because of inaccurate point-to-pixel assignments. Furthermore, the processing time increases due to the additional computation of the projection.

V. CONCLUSION

In this work, we presented a novel approach to combine LiDAR odometry with light-weight dynamic object detection and tracking. In contrast to other methods, our method does not rely on any pre-trained data and avoids the computational burden of volumetric mapping approaches by detecting dynamic objects directly in the structured point cloud. Efficiency is gained by reusing residuum information from the scan matching process in combination with heuristics to distinguish between dynamic and static objects. We qualitatively evaluated our method on a public dataset and a new custom dataset, demonstrating significant improvements in the computational efficiency in comparison to a state-of-the-art volumetric mapping approach. We demonstrated that our approach is able to detect articulated and fast-moving objects in mid-range distances and track them over long sequences. Both the data and the code are available as open-source.

REFERENCES

[1] H. Lim, S. Hwang, and H. Myung, "ERASOR: Egocentric Ratio of Pseudo Occupancy-based Dynamic Object Removal for Static 3D Point Cloud Map Building," *IEEE Robotics and Automation Letters*, pp. 2272–2279, 2021.

[2] Z. Yang, Y. Sun, S. Liu, *et al.*, "3dssd: Point-based 3d single stage object detector," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 11 040–11 048.

[3] A. H. Lang, S. Vora, H. Caesar, *et al.*, "Pointpillars: Fast encoders for object detection from point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 12 697–12 705.

[4] A. Milioto, I. Vizzo, J. Behley, *et al.*, "Rangenet++: Fast and accurate lidar semantic segmentation," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 4213–4220.

[5] K. Chen, B. T. Lopez, A.-a. Agha-mohammadi, *et al.*, "Direct lidar odometry: Fast localization with dense point clouds," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2000–2007, 2022.

[6] K. Daun and O. von Stryk, "Requirements and challenges for autonomy and assistance functions for ground rescue robots in reconnaissance missions," in *IEEE Intl. Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2023 in press.

[7] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time.," in *Robotics: Science and systems*, Berkeley, CA, vol. 2, 2014, pp. 1–9.

[8] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 4758–4765.

[9] L. Schmid, O. Andersson, A. Sulser, *et al.*, "Dynablox: Real-time detection of diverse dynamic objects in complex environments," *IEEE Robotics and Automation Letters*, 2023.

[10] C.-C. Wang, C. Thorpe, and S. Thrun, "Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas," in *IEEE Intl. Conf. Robot. Automat.*, vol. 1, 2003, pp. 842–849.

[11] D. Yoon, T. Tang, and T. Barfoot, "Mapless online detection of dynamic objects in 3d lidar," in *IEEE Conf. Comp. Robot Vision (CRV)*, 2019, pp. 113–120.

[12] D. Bolya, C. Zhou, F. Xiao, *et al.*, "Yolact: Real-time instance segmentation," in *Proc. IEEE/CVF Intl. Conf. Comput. Vis. (ICCV)*, 2019, pp. 9157–9166.

[13] W. Xiao, B. Vallet, M. Brédif, *et al.*, "Street environment change detection from mobile laser scanning point clouds," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 107, pp. 38–49, 2015.

[14] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3d environment," in *IEEE Intelligent Vehicles Symposium*, 2012, pp. 802–807.

[15] G. Postica, A. Romanoni, and M. Matteucci, "Robust moving objects detection in lidar data exploiting visual cues," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 1093–1098.

[16] H. Oleynikova, Z. Taylor, M. Fehr, *et al.*, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 1366–1373.

[17] L. Schmid, M. Abate, Y. Chang, *et al.*, "Khronos: A unified approach for spatio-temporal metric-semantic slam in dynamic environments," *arXiv preprint arXiv:2402.13817*, 2024.

[18] A. Dewan, T. Caselitz, G. D. Tipaldi, *et al.*, "Motion-based detection and tracking in 3d lidar scans," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 4508–4513.

[19] F. Moosmann and C. Stiller, "Joint self-localization and tracking of generic objects in 3d range data," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 1146–1152.

[20] A. Dewan, T. Caselitz, G. D. Tipaldi, *et al.*, "Rigid scene flow for 3d lidar scans," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 1765–1770.

[21] Y. Wang, W. Yao, B. Zhang, *et al.*, "DRR-LIO: A Dynamic-Region-Removal-Based LiDAR Inertial Odometry in Dynamic Environments," *IEEE Sensors Journal*, vol. 23, pp. 13 175–13 185, 2023.

[22] J. P. Underwood, D. Gillsjö, T. Bailey, *et al.*, "Explicit 3D change detection using ray-tracing in spherical coordinates," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013, pp. 4735–4741.

[23] F. Ferri, M. Gianni, M. Menna, *et al.*, "Dynamic obstacles detection and 3d map updating," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 5694–5699.

[24] F. Pomerleau, P. Krusi, F. Colas, *et al.*, "Long-term 3D map maintenance in dynamic environments," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 3712–3719.

[25] X. Ding, Y. Wang, H. Yin, *et al.*, "Multi-session map construction in outdoor dynamic environment," in *IEEE Intl. Conf. on Real-time Computing and Robotics (RCAR)*, 2018, pp. 384–389.

[26] J. Schauer and A. Nüchter, "The peopleremover—removing dynamic objects from 3-d point cloud data by traversing a voxel occupancy grid," *IEEE Robotics and Automation letters*, vol. 3, pp. 1679–1686, 2018.

[27] L. Mao, W. Gao, H. Chen, *et al.*, "Lio-dor: A robust lidar inertial odometry with real-time dynamic object removal," in *IEEE Intl. Conf. on Robotics and Biomimetics (ROBIO)*, 2023, pp. 1–7.

[28] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3D laser scans for online operation," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 163–169.

[29] X. Weng, J. Wang, D. Held, *et al.*, "Ab3dmot: A baseline for 3d multi-object tracking and new evaluation metrics," *arXiv preprint arXiv:2008.08063*, 2020.

[30] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[31] P. Pfreundschuh, H. F. C. Hendrikx, V. Reijgwart, *et al.*, "Dynamic Object Aware LiDAR SLAM based on Automatic Generation of Training Data," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 11 641–11 647.