2019 16th Conference on Computer and Robot Vision (CRV)

Mapless Online Detection of Dynamic Objects in 3D Lidar

David J. Yoon, Tim Y. Tang, Timothy D. Barfoot
Institute for Aerospace Studies
University of Toronto
Toronto, Canada
Email: {david.yoon, tim.tang}@robotics.utias.utoronto.ca, tim.barfoot@utoronto.ca

***Abstract*—This paper presents a model-free, setting-independent method for online detection of dynamic objects in 3D lidar data. We explicitly compensate for the moving-while-scanning operation (motion distortion) of present-day 3D spinning lidar sensors. Our detection method uses a motion-compensated freespace querying algorithm and classifies between dynamic (currently moving) and static (currently stationary) labels at the point level. For a quantitative analysis, we establish a benchmark with motion-distorted lidar data using CARLA, an open-source simulator for autonomous driving research. We also provide a qualitative analysis with real data using a Velodyne HDL-64E in driving scenarios. Compared to existing 3D lidar methods that are model-free, our method is unique because of its setting independence and compensation for pointcloud motion distortion.**

***Keywords*-Dynamic Object Detection; Lidar;**

I. Introduction

An autonomous system must be aware of dynamic elements in its environment. Our focus is on detection using lidar (light detection and ranging), specifically spinning lidars, which operate by sweeping multiple lasers about an axis for a 360° field of view (FOV). For the remainder of this paper, we refer to detectable elements as objects.

We place detection methods into three categories: methods that use class-specific detectors [1] [2], methods that use maps [3] [4] [5], and methods that only use recently acquired data (live data) [6] [7]. Class-specific, or model-based, detectors use prior information of objects and therefore have restricted use. Methods that use maps are also restricted, but may not require prior information on objects, which we refer to as model-free. Lastly, methods that only use live data may not require prior information on objects or the setting (e.g., maps), at the cost of being unable to detect stationary objects that have the potential to move (e.g., stationary cars in traffic). The categories are complementary, and therefore a combination can be effective [8] [9] [10].

Our work belongs to the category that only uses live data. We detect objects, regardless of the class or setting (e.g., on-road or off-road), as long as they are moving in the current scene. Our method is therefore applicable to a large variety of applications. In the urban driving setting, an application dominated by deep learning methods, our model-free detector could be used as a safety net. In a disaster zone scenario, where a prior map does not exist because of dramatic changes to the scene, our detector could be used to identify survivors. Our method is isotropic to the setting (e.g., we do not exploit knowledge of the gravity vector to help detect a ground plane), which is a contributing factor to setting-independence.

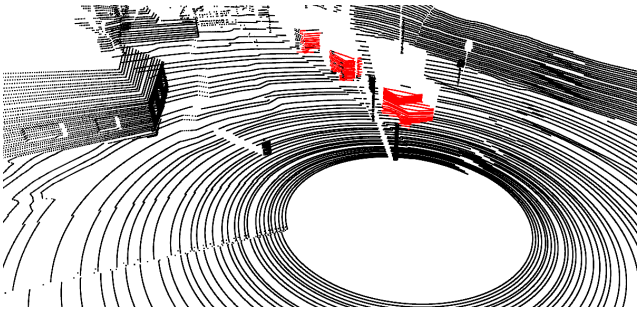


Figure 1: Example pointcloud of simulated lidar data (CARLA [11]). The output of our dynamic object detection method is indicated by the coloured points. We make our motion-distorted datasets with point-level groundtruth on dynamic objects available as a public benchmark.

Our main contribution is setting and object independent detection that outputs point-level labels as dynamic (currently moving) or static (currently stationary). We use a lidar odometry algorithm from previous work that compensates for motion distortion in lidar scans caused by the moving-while-scanning operation of spinning lidars [12] [13]. We use a novel freespace querying algorithm that also compensates for motion distortion using the estimated continuous-time trajectory from the lidar odometry. To the best of our knowledge, our method is the only existing among other model-free ones that combines all of the following traits: motion-compensation, setting independence, only uses live data (i.e., no maps or training data).

At the time of writing, to the best of our knowledge, a benchmark of motion-distorted 3D lidar data with per-point groundtruth labels does not exist. Thus our secondary contribution is a benchmark of simulated data using CARLA [11], an open-source simulator for autonomous driving research (see example in Fig. 1), which we believe to be the first to provide motion-distorted data with per-point groundtruth labels on moving objects. We make this dataset publically available in hopes that other researchers may use to make

978-1-7281-1838-3/19/$31.00 ©2019 IEEE
DOI 10.1109/CRV.2019.00023

comparisons with our work. We also provide a qualitative evaluation of our pipeline with real data collected with a Velodyne HDL-64E sensor on a vehicle.

The remainder of the paper is organized as follows. Section II discusses literature; Section III describes the pipeline methodology; Section IV presents the simulation, experiment, and performance analysis; and Section V provides concluding thoughts and future work recommondations.

II. RELATED WORK

Class-specific, or model-based, detectors take advantage of prior information of the objects to be detected. Petrovskaya et al. [1] model vehicles as 2D bounding boxes, which is applied to 3D data by processing it into a 2D representation. Rather than manually crafting models, recent work focuses on learning methods. Chen et al. [2] (among many others) take lidar and camera data as input to a deep neural network (DNN) and output class-specific detections. While this category is proven to work well and achieves more than dynamic labels (i.e., bounding boxes with orientation), such methods will simply not detect objects for which they have not been trained.

Detection without prior object information is possible by comparing current data to a reliable prior map. Given a reliable map of the stationary world, differences from the comparison are indicative of dynamic objects. Sometimes called *change detection* [3] [4], these methods make use of pointcloud comparisons (i.e., end-points of lidar measurements) and freespace comparisons (i.e., paths traced by lidar measurements). Hebel et al. [3] raytrace lidar data into occupancy voxel grids for their freespace representation. Occupancy voxel grids are expensive computationally and in memory, so instead, Pomerleau et al. [5] query lidar freespace by matching measurements with local azimuth-elevation values (i.e., spherical coordinates). The method is efficient, but assumes pointclouds are not motion distorted.

Our interest is in methods that do not require prior information on the objects or the setting (i.e., no maps or training data), only making use of the current lidar scans (live data). Such methods are limited to detecting objects that are moving in the current scene. Objects that are stationary, but may be of interest (e.g., stationary cars in traffic), are not detectable by such methods. We see this as an acceptable trade-off for object and setting independence.

Among live data methods are ones that only use pointcloud information. Dewan et al. [6] compare subsequent pointclouds and sequentially identify motion through a voting scheme. The first detected motion will always be the relative motion of the stationary environment, followed by the largest dynamic objects. They directly compared their work to Moosmann and Stiller [8] and showed superior performance at the object level. In another publication, Dewan et al. [7] produce scene flow (i.e., point-wise velocity estimation), from which dynamic labels are trivial. However, both of their methods are dependent on the setting because they remove ground points as a pre-processing step.

Live data methods are challenging because there are significant differences between subsequent lidar scans not just due to dynamic objects, but because of viewpoint occlusions and data sparsity, causing false detections. For example, Fig. 3a shows the result of a pointcloud comparison between two subsequent scans with differences coloured as red. Notice the abundance of red points on the ground due to the sparsity of points on distant and angled surfaces (i.e., the ground) with respect to the sensor. Removing ground points in this case is helpful, but relies on the setting-dependent assumption of a relatively flat ground (i.e., not applicable to off-road settings) that is in the direction of the gravity vector. It is also not clear how to handle differences from occlusions with only pointclouds.

Live data methods that use freespace handle viewpoint occlusions well. Azim and Aycard [14] raytrace over occupancy voxel grids and compare them over time, but only provide qualitative results. Postica et al. [15] also make comparisons with occupancy voxel grids. They present quantitative results using short sequences from the KITTI dataset [16], for which they have manually annotated for groundtruth, but have not made public. Notable limitations of their work include relying on pre-processing ground points (setting-dependent) and ignoring measurements further than 30 m. The limited range is significant considering lidar sensors are capable of measurements more than twice this distance. Neither works consider motion distortion.

The three categories are complementary to one another, so a combination can be more effective. Moosmann and Stiller [8] segment pointclouds into object proposals, which they track over time. Consistent ones are labelled dynamic by a learned classifer. Ushani et al. [9] use freespace and learning to compute scene flow. Occupancy voxel grids coarsely identify dynamic points, which are then refined by a learned classifier. They make a planar motion assumption and limit their method to a 50 m $\times$ 50 m grid. Dewan et al. combine their prior work on scene flow [7] with a DNN to produce point labels of dynamic, static, and a third label for objects with the potential to move (e.g., stationary cars) [10].

Our method belongs to the live data category and is model-free, labelling each point as dynamic or static. We motion-compensate both pointclouds and freespace querying, which existing methods do not consider. We make full use of the sensor range, which is often limited for methods that use freespace. We do not make assumptions of the setting, such as the existance of the ground for removal of ground points, or use data from previous experiences (e.g., maps or training data). If setting independence is not important, our method can be combined with strategies from the other categories for a more robust detection system.

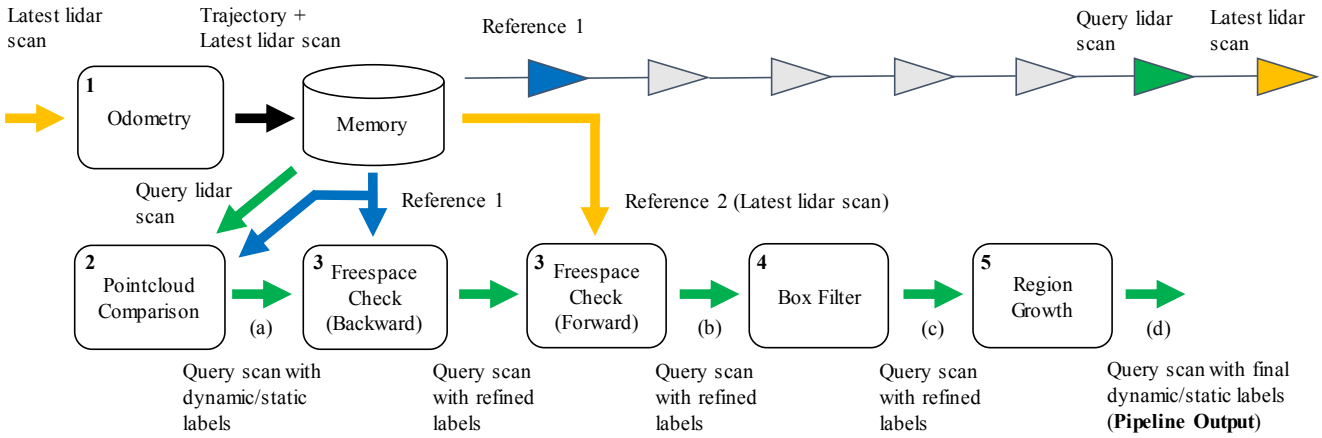
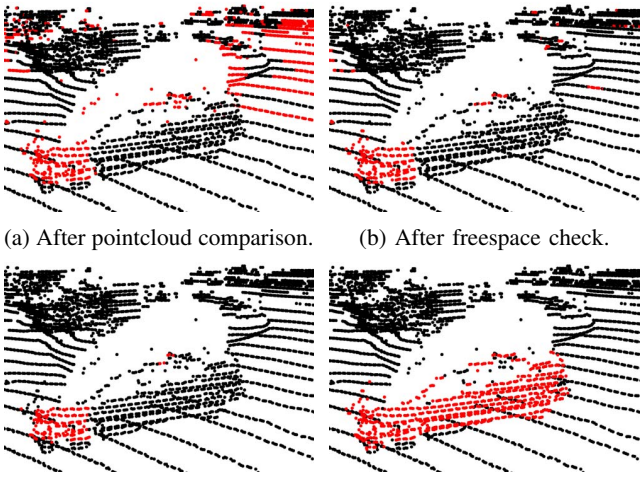


Figure 2: The pipeline describes the sequence of operations on the lidar scan of interest (*query scan*), outputing the scan with points labelled dynamic or static. A lidar odometry algorithm computes the sensor trajectory, which aligns the *latest scan*. The labels (a) to (d) correspond to the images in Fig. 3. The numbers correspond to the enumerated steps in the text.



(a) After pointcloud comparison. (b) After freespace check.

(c) After box filter. (d) After region growth.

Figure 3: A vehicle throughout the detection pipeline. Refer to the pipeline in Fig. 2 for corresponding letters (a) to (d). There is an abundance of mislabelled ground points in (a) due to pointcloud sparsity and the lack of point neighbours to compute accurate surface normals.

III. METHODOLOGY

We are interested in labelling lidar points as static or dynamic. We define a lidar scan as a full revolution of the spinning lidar which, depending on the lidar, can have more than 100000 points. Each laser of a spinning lidar has a unique position and orientation on the rotating base. This means the laser ray paths (i.e., freespace) are not equivalent to azimuth and elevation computations of the endpoints. Thus we refer to scans as including both pointcloud and freespace information. We refer to measurements as points when appropriate. In a full pass of the pipeline, we label all points in a single scan of interest (*query scan*).

A pipeline diagram is shown in Fig. 2. The sequential steps to the pipeline are explained as follows:

1) *Odometry*: Align *latest scan* using lidar odometry.
2) *Pointcloud Comparison*: Comparison of query scan against another. Discrepencies are set to dynamic.
3) *Freespace Check*: Check dynamic points against freespace of another scan. Points not in freespace are not dynamic and changed to static.
4) *Box Filter*: Apply a sliding box filter on the image representation of the query scan for outlier rejection.
5) *Region Growth*: Cluster the dynamic query scan points. Add nearby points to clusters if they satisfy conditions that indicate they are part of the same object.

We spend the rest of this section to discuss each step in more detail and highlight its contribution to the pipeline.

A. Odometry

Our lidar odometry considers the exact timestamps of measurements to produce a continuous-time trajectory of the moving sensor platform. For more detail, please see our previous work [12], which uses the trajectory representation of Anderson and Barfoot [17].

With a continuous-time trajectory, we can query for any pose and velocity given a timestamp, t. We use the notation $\mathbf{T}_{v,0}(t) \in SE(3)$, where $\mathbf{T}_{v,0}$ is a transformation from the world frame, $\underrightarrow{\mathcal{F}}_0$, to the moving sensor platform, $\underrightarrow{\mathcal{F}}_v$. $\boldsymbol{\varpi}(t)$ is the body-centric velocity of the sensor platform.

B. Pointcloud Comparison

We compute a pointcloud comparison between the query scan and a previous *reference scan*. The scans are of different time intervals, so dynamic objects cause discrepencies.

We identify discrepencies using error metrics commonly used in pointcloud alignment. Depending on the implementation of the previous lidar odometry step, most of, if not all,

the computation may already be completed. We use a point-to-plane metric when points have sufficient neighbours to compute surface normals. Otherwise, we use a point-to-point metric. All points are transformed to $\underrightarrow{\mathcal{F}}_0$ using $\mathbf{T}_{v,0}(t)$, creating motion-compensated pointclouds. Given a *query point*, $\mathbf{q}_0$, its normalized surface normal, $\mathbf{n}^q$, and its nearest reference scan neighbour, $\mathbf{p}_0$, the point-to-plane metric is $|\mathbf{n}^q \cdot (\mathbf{p}_0 - \mathbf{q}_0)|$. The point-to-point metric is $\|\mathbf{p}_0 - \mathbf{q}_0\|_2$.

We compute the error metric for all query points. Static points have low error because there should be a corresponding reference scan point of the same surface observation. We expect high error from dynamic points since they are observations of moving surfaces. However, a moving $\underrightarrow{\mathcal{F}}_v$ causes new surface observations and viewpoint occlusions, which also cause high error. Regardless, we take a constant scalar *error threshold*. Those greater than the error threshold are labelled dynamic, the rest are static. Fig. 3a shows an example pointcloud output, where dynamic points are labelled in red. There is an abundance of mislabelled points on the ground, distant from the sensor. The point-to-plane metric is ineffective in these cases due to a lack of point neighbours to compute dependable surface normals. We refine incorrect dynamic labels later in the pipeline.

Notice there is a *scan gap* of 4 between the query scan and reference scan (Reference 1) in the top-right illustration of Fig. 2. We require a scan gap to ensure dynamic objects sufficiently displace between the two scans. Fig. 3a shows a partially labelled object since the ideal scan gap requires unknown quantities (i.e., object speed and size). Instead, we aim to partially label objects and recover the rest later in the pipeline. The error threshold, scan gap, and lidar scan rate together define the minimum speed of detectable objects.

C. Freespace Check

Recall that many query points are mislabelled dynamic from pointcloud comparisons because of point sparsity, viewpoint occlusions, or they are new surface observations. We check all dynamic query points, which can be as many as half the query scan points, against the freespace of another scan to correct mislabels from the pointcloud comparison.

Given a dynamic query point and reference scan that defines the freespace of interest, we wish to determine if the query point is inside, on the border of, or outside freespace. Freespace discrepencies are only caused by dynamic objects. Dynamic points inside the freespace of other scans are consistent with their current label, while ones on the border or outside freespace may not truly be dynamic. We use this argument to refine incorrect dynamic labels (see Fig. 3a and Fig. 3b) of the previous pointcloud comparison step. We do not check freespace for static points since a pointcloud comparison is equivalent to a freespace border check.

Recall that the spinning lidar continuously sweeps lasers about an axis for a 360° FOV. The laser ray paths, from the sensor to their endpoints, define freespace. We designed our freespace method specifically for a single lidar configuration that has its lasers approximately radiate outward vertically (i.e., along the sweeping axis). This is important because we exploit the elevation order of the lasers to speed up our freespace query.

Representing the freespace of the reference scan in its entirety is possible, but expensive (e.g., occupancy voxel grid). Instead, we only determine the reference scan measurement ray that has a direction that passes nearest to the query point (i.e., smallest point-to-line distance). Consider the surface plane that the query point lies on and the identified reference scan ray – there are three cases:

- Case 1: Ray intersects the surface plane.
- Case 2: Ray lies on the surface plane.
- Case 3: Ray does not reach the surface plane.

Case 1 means the surface plane is absent during the acquisition time period of the reference scan, which is possible if the query point is of a dynamic object. Case 2 means the measurement is an observation of the same surface plane (freespace border). Finally, Case 3 means another surface obstructed the measurement (outside freespace).

We assume a total of L lasers rotate together at constant speed, ω. Each laser ℓ, indexed by increasing elevation, has a unique pose with respect to the sensor hub (i.e., the rotating base), $\underrightarrow{\mathcal{F}}_h$, defined by the transformation $\mathbf{T}_{\ell,h} \in SE(3)$.

Given a query point, $\mathbf{q}_0$, we formulate for each laser, ℓ, the point-to-line distance as a continuous function of time:

$$\|\mathbf{e}^{\ell}(t)\|_2 = \left\| \mathbf{D}\mathbf{T}_{\ell,h}\mathbf{T}_{h,v}(t)\,\mathbf{T}_{v,0}(t) \begin{bmatrix}\mathbf{q}_0\\ 1\end{bmatrix} \right\|_2 . \quad (1)$$

$$\mathbf{T}_{h,v}(t) = \begin{bmatrix}\mathbf{R}^z(\omega t) & \mathbf{0}\\ \mathbf{0}^T & 1\end{bmatrix} \in SE(3), \quad \mathbf{D} = \begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix},$$

where $\mathbf{R}^z(\omega t) \in SO(3)$ is the constant rotation of $\underrightarrow{\mathcal{F}}_h$ with respect to $\underrightarrow{\mathcal{F}}_v$ at rotation speed ω. Note we define $\underrightarrow{\mathcal{F}}_v$ such that there is no translation between $\underrightarrow{\mathcal{F}}_h$ and $\underrightarrow{\mathcal{F}}_v$, and $\underrightarrow{\mathcal{F}}_h$ rotates about the z-axis of $\underrightarrow{\mathcal{F}}_v$. We define $\underrightarrow{\mathcal{F}}_\ell$ such that the laser points along the x-axis. See Fig. 4 for a visualization.

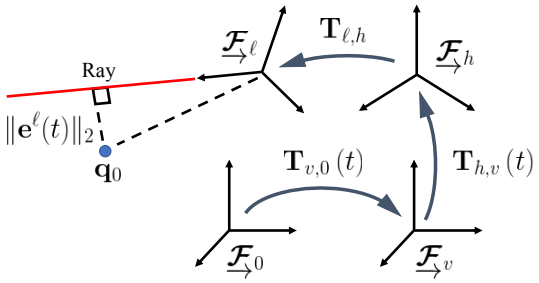


Figure 4: A visualization of the point-to-line distance (see Eq. 1). $\mathbf{T}_{v,0}$ is the sensor trajectory, $\mathbf{T}_{h,v}$ is the spinning lidar rotation, and $\mathbf{T}_{\ell,h}$ is the unique pose for laser ℓ. We require the time, t, and laser, ℓ, that minimizes $\|\mathbf{e}^{\ell}(t)\|_2$.

We require t^* and ℓ^* that minimize $\|\mathbf{e}^{\ell}(t)\|_2$. However, ℓ is discrete. We minimize $\|\mathbf{e}^{\ell}(t)\|_2$ iteratively by selecting ℓ and solving for t using least-squares optimization:

$$t = \arg\min_{t} \frac{1}{2} \mathbf{e}^{\ell}(t)^T \mathbf{e}^{\ell}(t). \tag{2}$$

We apply Gauss-Newton optimization by linearizing $\mathbf{e}^{\ell}(t) \approx \bar{\mathbf{e}}^{\ell}(t) + \mathbf{E}\delta t$, where $\bar{\mathbf{e}}^{\ell}(t)$ is the nominal term and

$$\mathbf{E} = \frac{d\mathbf{e}^{\ell}(t)}{dt} = \mathbf{D}\mathbf{T}_{\ell,h}\left(\dot{\mathbf{T}}_{h,v}\mathbf{T}_{v,0} + \mathbf{T}_{h,v}\dot{\mathbf{T}}_{v,0}\right)\begin{bmatrix}\mathbf{q}_0\\1\end{bmatrix}. \tag{3}$$

The time derivative, $\mathbf{E}$, has an exact expression since $\dot{\mathbf{T}} = \boldsymbol{\varpi}^{\wedge}\mathbf{T}$ [18] [1]. Thus the ith iterative update equation for t is

$$t^{(i)} \leftarrow t^{(i-1)} - \left[\left(\mathbf{E}^T\mathbf{E}\right)^{-1}\mathbf{E}^T\bar{\mathbf{e}}^{\ell}(t)\right]\Big|_{t=t^{(i-1)}}, \tag{4}$$

which is an inexpensive scalar update.

We additionally iterate by exploiting laser elevation order. We first solve Eq. 2 using initial guesses for t and ℓ, the laser neighbour above it, $\ell + 1$, and below it, $\ell - 1$. If a neighbour optimizes to a smaller $\|\mathbf{e}^{\ell}(t)\|_2$, we iteratively search along that direction by single laser increments, re-solving Eq. 2 and comparing optimized $\|\mathbf{e}^{\ell}(t)\|_2$ values. The iteration stops once $\|\mathbf{e}^{\ell}(t)\|_2$ no longer decreases or we run out of neighbours.

Our method relies on a good initial condition for t and ℓ, which is easily computed by assuming no motion distortion [5]. We experimentally verified this initialization choice always converges.

Given t^* and ℓ^*, the reference scan ray is the one with the closest timestamp. Adhering to the three possible cases, we check for an intersection between the ray and surface plane of the query point. This is easily done by computing the point-to-plane error metric of Section III-B with the ray endpoint for a freespace border test. Otherwise, the query point is inside or outside freespace depending on which side of its surface the endpoint resides in.

Fig. 2 shows two freespace blocks, *backward* (comparison with a previous scan), followed by *forward* (comparison with a later scan). Ideally, only backward is required. Unfortunately, objects moving away from the sensor will never be within freespace of a previous scan. Backward shares the same reference scan (or scan gap) as in the pointcloud comparison. A scan gap is not needed for forward because objects moving away have surface geometry perpendicular to the movement direction. We only need to compute forward for points identified as outside freespace in backward.

D. Box Filter

Our freespace check is susceptible to error because of finite lidar resolution (e.g., consider freespace at far ranges), leaving sparse traces of mislabelled dynamic points (see Fig. 3b). The query scan measurements are arranged into an image representation. Each laser forms a row and the consecutive measurements the columns. Dynamic labels have an image value of 1 and static labels have a value of 0. We filter outliers (dynamic mislabels) by sliding a box filter thoughout the image (see Fig. 5 for an example).

[1]The operator $\wedge$ converts $\boldsymbol{\xi} \in \mathbb{R}^6$ to a member of $\mathfrak{se}(3)$ [18].

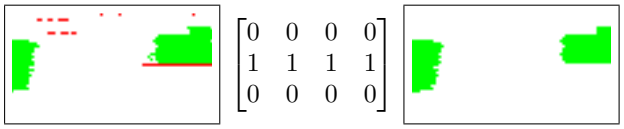


Figure 5: We apply a filter (middle) to a scan's image representation to remove mislabelled dynamic points (red) while maintaining correct ones (green). We show a before (left) and after (right) example (limited horizontal FOV).

We apply our filter (Fig. 5 middle) with a pixelwise XNOR (exclusive logical NOR) operation. The sum of all XNOR operations is a numerical score. Scores greater than a constant *score threshold* are considered outliers. The score threshold depends on the lidar resolution.

E. Region Growth

Often, the scan gap is not large enough for an object to completely displace from its previous position (see Fig. 3c). Thus we require a method for region growing (see Fig. 3d).

We first cluster the dynamic query points into object clusters. We use the 3D pointcloud clustering method presented by Klasing et al. [19]. A radially bounded nearest neighbour strategy incrementally groups dynamic points into clusters.

We region grow clusters using the iterative pointcloud segmentation method of Moosmann et al. [20]. Clusters are grown by testing neighbouring points for parallelism or convexity, until none are found. Given two points $\mathbf{p}^1$ and $\mathbf{p}^2$, with unit surface normals $\mathbf{n}^1$ and $\mathbf{n}^2$, the two points are convex if the following two conditions are both true:

$$\mathbf{n}^1 \cdot \left(\mathbf{p}^2 - \mathbf{p}^1\right) \leq 0, \quad \mathbf{n}^2 \cdot \left(\mathbf{p}^1 - \mathbf{p}^2\right) \leq 0.$$

IV. RESULTS

A. Simulated Lidar Benchmark (CARLA)

We chose to create a simulated dataset because of the difficulty in obtaining accurate groundtruth labels at the point level. The KITTI Vision Benchmark Suite [16], a popular dataset, lacks point-level labels and does not provide raw lidar data (range and bearing with laser positions and orientations). The Paris-Lille-3D lidar dataset [21] has point-level labels, but similar to KITTI, only provides motion-compensated pointclouds. They also removed points with range further than 20 m. Publications with work comparable to ours either omit quantitative results or use short sequences of manually labelled data, which they do not make public.

CARLA is an open-source simulator for autonomous driving research [11]. Two urban scenarios are provided with 2.9 km (Town 1) and 1.9 km (Town 2) of drivable roads. We currently made 5 min sequences of each,

which we plan on expanding in the near future (visit http://asrl.utias.utoronto.ca/datasets/mdlidar/index.html).

We encourage use of our dataset for comparison. We made modifications to the CARLA source code to produce datasets matching a real Velodyne HDL-64E (e.g., laser positions and orientations). We capture motion distortion by making each laser activate once every simulation step, resulting in 128000 measurements (120 m maximum range) at 10 Hz frequency. See an example pointcloud in Fig. 1.

For groundtruth, points moving faster than 0.2 m/s are dynamic. True positives (TP) are points correctly labelled dynamic. False positives (FP) and False negatives (FN) are points incorrectly labelled dynamic and static, respectively.

We compute precision, P, and recall, R, in two ways. Given the scan index, n, and the total number of scans, N, the *total* computation is:

$$\mathrm{P}_t = \frac{\sum_n^N \mathrm{TP}_n}{\sum_n^N (\mathrm{TP}_n + \mathrm{FP}_n)}, \; \mathrm{R}_t = \frac{\sum_n^N \mathrm{TP}_n}{\sum_n^N (\mathrm{TP}_n + \mathrm{FN}_n)}. \quad (5)$$

Given the total number of valid scans for P, N_p, and the total number of valid scans for R, N_r, the *average* computation is:

$$\mathrm{P}_a = \frac{\sum_n^{N_p} \mathrm{TP}_n / (\mathrm{TP}_n + \mathrm{FP}_n)}{N_p}, \; \mathrm{R}_a = \frac{\sum_n^{N_r} \mathrm{TP}_n / (\mathrm{TP}_n + \mathrm{FN}_n)}{N_r}. \quad (6)$$

Scans where the denominator is 0 are ignored (e.g., $\mathrm{TP}_n + \mathrm{FP}_n = 0$), which is why we distinguish N_p and N_r.

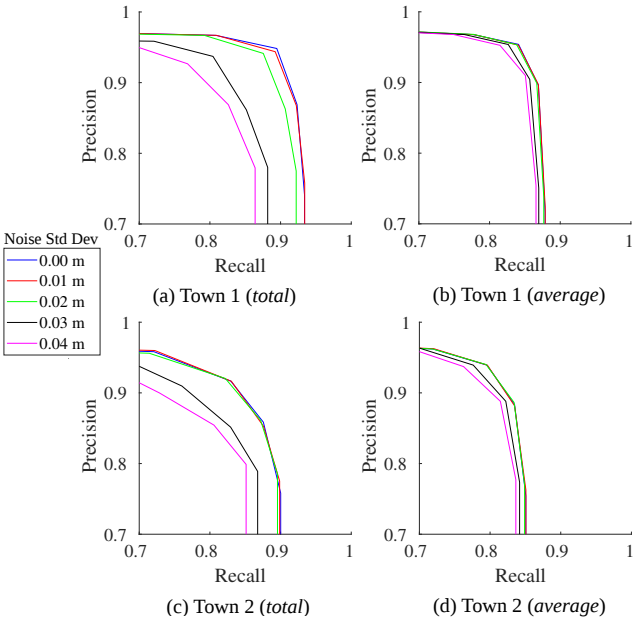


Figure 6: Precision-recall plots (*total* and *average* - see Eq. 5 and 6) on two simulated sequences. The standard deviation of range measurement noise was varied. Note that a Velodyne HDL-64E has a standard deviation rated less than 0.02 m.

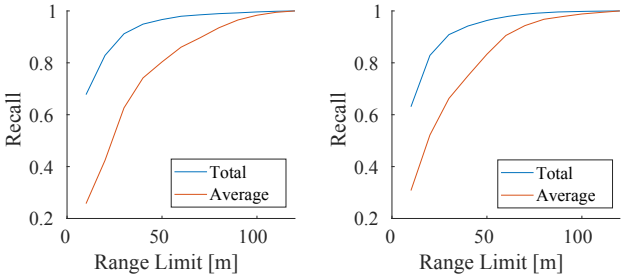


Figure 7: Recall (*total* and *average* - see Eq. 5 and 6) using groundtruth labels on two simulated sequences with varying limited range. Total recall is high at low range limits because nearby objects have more points, downplaying far-away ones.

B. Benchmark Results

We computed PR curves by varying the error threshold (see Fig. 6). Noise was added to range measurements with varying standard deviation. A Velodyne HDL-64E has a range standard deviation rated less than 0.02 m. The scan gap was set to 4, allowing sufficent object displacement for a 10 Hz lidar. The score threshold was set to 10, which was determined experimentally on data from Town 2. We emphasize that the score threshold depends on the lidar resolution and not the application setting. For these simulated results, we used the groundtruth trajectory instead of lidar odometry to focus on the detection aspect.

Fig. 7 shows R evaluated with groundtruth at varying range limits. R_t is higher for low range limits because there are more points on closer objects, downplaying the neglection of far-away objects. This also explains why range noise affects the total curves more in Fig. 6. Greater noise reduces performance, but taking the average downplays scans with more mistakes (i.e., scans with very close objects).

We perform worse in Town 2 because of more partial occlusion instances (e.g., by fences), which we struggle with. There is latency for detecting objects accelerating from being stationary because of the scan gap, reducing R.

We hope for future comparisons to other works as our dataset is public. For now we make an indirect comparison to Dewan et al. [10] as the state of the art. They used two manually labelled sequences of Velodyne HDL-64E data at the point level. Their pipeline without deep learning, which is setting-dependent because of ground point removal, reports the following maximum F1-score PR values: 72.8 P and 92.3 R (38 s length), and 59.5 P and 69.6 R (50 s length). Adding learning increases P at the cost of R. They do not distinguish PR in two ways like we did and is unclear about the exact computation. We stress that a fair comparison is not possible since they used real data, but we at least see our total PR values using longer, simulated sequences are comparable.

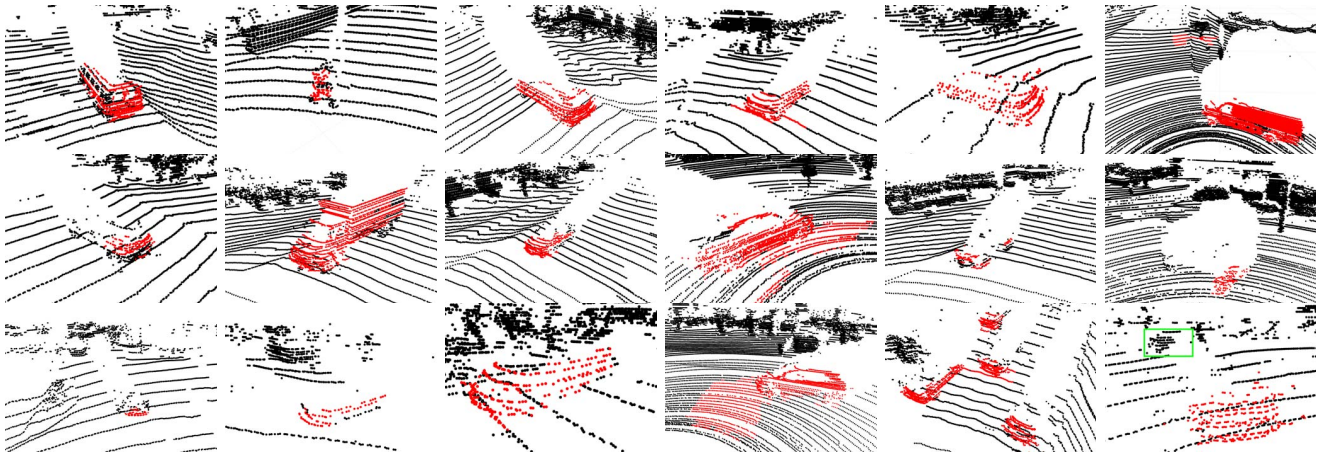


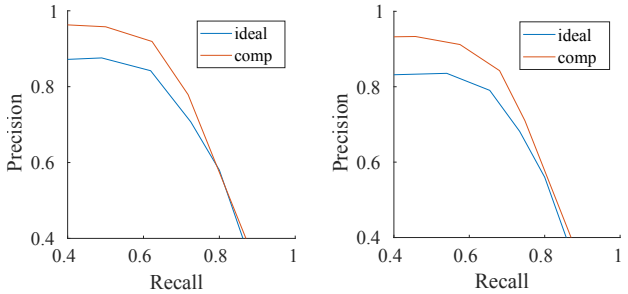
Figure 8: Real data examples (22 dynamic objects) of our detection method. False detections can occur (row 1, col. 6), but rarely persist in the next scan. Inaccurate surface normals cause incomplete region growth (col. 1 and 2) or growth to static points (col. 4). The green box in the last image indicates an unlabelled dynamic object due to occlusion by the other object in the freespace scan(s). The example in row 2, col. 6 is a barely visible vehicle due to many faulty returns.

C. Motion Compensation

We compare our motion-compensated freespace querying algorithm with the azimuth-elevation method of Pomerleau et al. [5]. Their method approximates the laser ray paths of reference points by their azimuths and elevations in their local frame, requiring the ideal pointcloud assumption that each scan is an instantaneous snapshot (i.e., no motion distortion) with all measurements originating from a single origin point. For this comparison experiment, we do not apply the box filter and region growth steps in the pipeline to isolate the performance of the freespace queries.

Recall that a freespace query for a query point involves finding the reference scan ray that passes closest to it. We make the comparison by evaluating two pipelines. The motion-compensated pipeline is a pipeline as presented in this work without the box filter and region growth. The ideal assumption pipeline is a similar pipeline that queries freespace using the azimuth-elevation method. We construct a 2D kd-tree of azimuth-elevation pairs computed from all points of the reference scan in the local frame of the reference scan. The nearest reference ray for each query point is determined by searching the kd-tree for its nearest azimuth-elevation neighbour.

We evaluated the motion-compensated (comp) and ideal assumption (ideal) pipelines on Town 1 data. Fig. 9 shows PR curves by varying the error threshold with no measurement noise. R is low for both pipelines because region growth is not applied for partial object detections (e.g., see Fig. 3b). P is generally higher for the motion compensated pipeline, converging to identical performance at the highest R values. Therefore our motion-compensated algorithm has better performance compared to the azimuth-elevation method, demonstrating higher P without sacrificing R.



(a) Total (b) Average

Figure 9: Precision-recall plots (*total* and *average* - see Eq. 5 and 6) comparing our freespace query algorithm (comp) to the azimuth-elevation method (ideal) [5]. See text for experiment setup. Our algorithm has better precision without lowering recall. Recall is generally low for both as we did not apply region growth.

D. Experiments on Real Data

A Velodyne HDL-64E, mounted on a vehicle, was driven through Richmond Hill, Ontario. Lidar odometry was used to estimate motion. The error threshold was set to 0.5 m. Other parameters were not changed from the simulated benchmark.

A limitation was the inability to distinguish between maximum range measurements and faulty returns. Faulty returns often occur on vehicles, particularly darker ones [1]. Thus we cannot use maximum range measurements for freespace computation without incorrectly using faulty returns. This is not detrimental for ground-based applications, since Velodyne lasers are angled downward to the ground. Applications where objects have no geometry behind them for the lidar to perceive (e.g., flying objects) are an issue.

Our pipeline works well in scenarios with consistent motion (e.g., no traffic slow-downs). Fig. 8 is a collage of

real data examples, showing 22 different dynamic objects. Our pipeline struggles with occluded objects (row 3, col. 6), which is also reflected in our simulated benchmark. The pipeline also struggles with inaccurate surface normal computations, causing incomplete region growth (col. 1 and 2), or excessive growth to static points (col. 4). Row 2, col. 6 shows a barely visible vehicle due to many faulty returns.

On a laptop with an Intel Core i7-6820HQ CPU, we currently process lidar scans of a Velodyne HDL-64E at 3 Hz on average on a single thread, slower than the scan rate (10 Hz). However, much of the required computation is per query point (e.g., freespace query), which is easily parallelizable. An implementation running at 10 Hz is not a concern and will be completed in the near future.

V. CONCLUSIONS AND FUTURE WORK

This paper presents an online detection method for labeling 3D lidar points as dynamic (moving) or static (stationary). Motion distortion is explicitly compensated, which existing methods do not consider. We only rely on the latest scans of lidar data (i.e., no maps or training data). Another trait that makes our method unique is environment isotropy. Thus our detection method is model-free and setting-independent, applicable to a wide variety of applications. We also establish and make public a benchmark with simulated motion-distorted lidar data with point-level groundtruth on dynamic objects (http://asrl.utias.utoronto.ca/datasets/mdlidar/index.html).

Future work involves resolving the issue of detecting objects without geometry behind them (e.g., flying objects), which is currently an issue because of the inability to distinguish between maximum range and faulty returns.

ACKNOWLEDGMENT

This work was supported financially by Applanix Corporation, the Natural Sciences and Engineering Research Council (NSERC), and the Ontario Graduate Scholarship (OGS). We thank General Motors (GM) for the vehicle donation and Defence Research and Development Canada (DRDC) for the Velodyne HDL-64E loan.

REFERENCES

[1] A. Petrovskaya and S. Thrun, "Model based vehicle detection and tracking for autonomous urban driving," *Autonomous Robots*, vol. 26, no. 2-3, pp. 123–139, 2009.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Computer Vision and Pattern Recognition*, 2017.

[3] M. Hebel, M. Arens, and U. Stilla, "Change detection in urban areas by direct comparison of multi-view and multi-temporal ALS data," in *Photogrammetric Image Analysis*. Springer, 2011, pp. 185–196.

[4] J. P. Underwood, D. Gillsjö, T. Bailey, and V. Vlaskine, "Explicit 3D change detection using ray-tracing in spherical coordinates," in *International Conference on Robotics and Automation (ICRA)*, 2013, pp. 4735–4741.

[5] F. Pomerleau, P. Krüsi, F. Colas, P. Furgale, and R. Siegwart, "Long-term 3D map maintenance in dynamic environments," in *International Conference on Robotics and Automation (ICRA)*, 2014, pp. 3712–3719.

[6] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard, "Motion-based detection and tracking in 3D lidar scans," in *International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4508–4513.

[7] ——, "Rigid scene flow for 3D lidar scans," in *Intelligent Robots and Systems (IROS)*, 2016, pp. 1765–1770.

[8] F. Moosmann and C. Stiller, "Joint self-localization and tracking of generic objects in 3D range data," in *International Conference on Robotics and Automation (ICRA)*, 2013, pp. 1146–1152.

[9] A. K. Ushani, R. W. Wolcott, J. M. Walls, and R. M. Eustice, "A learning approach for real-time temporal scene flow estimation from lidar data," in *International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5666–5673.

[10] A. Dewan, G. L. Oliveira, and W. Burgard, "Deep semantic classification for 3D lidar data," in *Intelligent Robots and Systems (IROS)*, 2017, pp. 3544–3549.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[12] T. Y. Tang, D. J. Yoon, F. Pomerleau, and T. D. Barfoot, "Learning a bias correction for lidar-only motion estimation," in *Computer Robot and Vision (CRV)*, 2018.

[13] P. McGarey, D. Yoon, T. Tang, F. Pomerleau, and T. Barfoot, "Field deployment of the tethered robotic eXplorer to map extremely steep terrain," in *Field and Service Robotics (FSR)*, 2018, pp. 303–317.

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

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

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 11, pp. 1231–1237, 2013.

[17] S. Anderson and T. D. Barfoot, "Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE(3)," in *Intelligent Robots and Systems (IROS)*, 2015, pp. 157–164.

[18] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[19] K. Klassing, D. Wollherr, and M. Buss, "A clustering method for efficient segmentation of 3D laser data," in *International Conference on Robotics and Automation (ICRA)*, 2008, pp. 4043–4048.

[20] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion," in *Intelligent Vehicle Symposium (IV)*, 2009, pp. 215–220.

[21] X. Roynard, J. Deschaud, and F. Goulette, "Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification," *International Journal of Robotics Research (IJRR)*, vol. 37, no. 6, pp. 545–557, 2018.