A Learning Approach for Real-Time Temporal Scene Flow Estimation from LIDAR Data

Arash K. Ushani, Ryan W. Wolcott, Jeffrey M. Walls, and Ryan M. Eustice

***Abstract*— Many autonomous systems require the ability to perceive and understand motion in a dynamic environment. We present a novel algorithm that estimates this motion from raw LIDAR data in real-time without the need for segmentation or model-based tracking. The sensor data is first used to construct an occupancy grid. The foreground is then extracted via a learned background filter. Using the filtered occupancy grid, raw scene flow between successive scans is computed. Finally, we incorporate these measurements in a filtering framework to estimate temporal scene flow. We evaluate our method on the KITTI dataset.**

I. Introduction

Autonomous systems, such as self driving vehicles or other mobile robots, operate in dynamic environments where it is critical to be able to accurately perceive and understand the motion of the surrounding environment. Increasingly often, these systems are equipped with one or more light detection and ranging (LIDAR) sensors. These sensors provide a point cloud representation of the world, often collecting millions of points per second.

Many algorithms that consider dynamic scene understanding work with the point cloud directly. For example, obstacle tracking techniques for self-driving vehicles often rely on detection and segmentation of objects directly from the LIDAR generated point cloud, including our previous work [1] and others [2–7].

However, working with point clouds alone disregards a valuable characteristic of the sensor: the notion of free space swept out between the point return and the LIDAR sensor. Occupancy grids are a commonly used representation that can be readily manipulated to capture free and unknown space, in addition to the occupancy of a LIDAR point return. This can be achieved by ray-casting from the sensor emitter to the returned point, populating cells of a grid with observations of "free space" [8].

Applications of occupancy grids have been widely considered in the 2D domain. However, extending the use of occupancy grids to 3D sensors has been limited. The exponential increase in processing required for handling the significant number of voxels in a 3D occupancy grid presents a challenge. In this work, our algorithms are designed so that they can easily be offloaded to a GPU. Thus, we can better formulate a real-time temporal scene flow framework from LIDAR scanners.

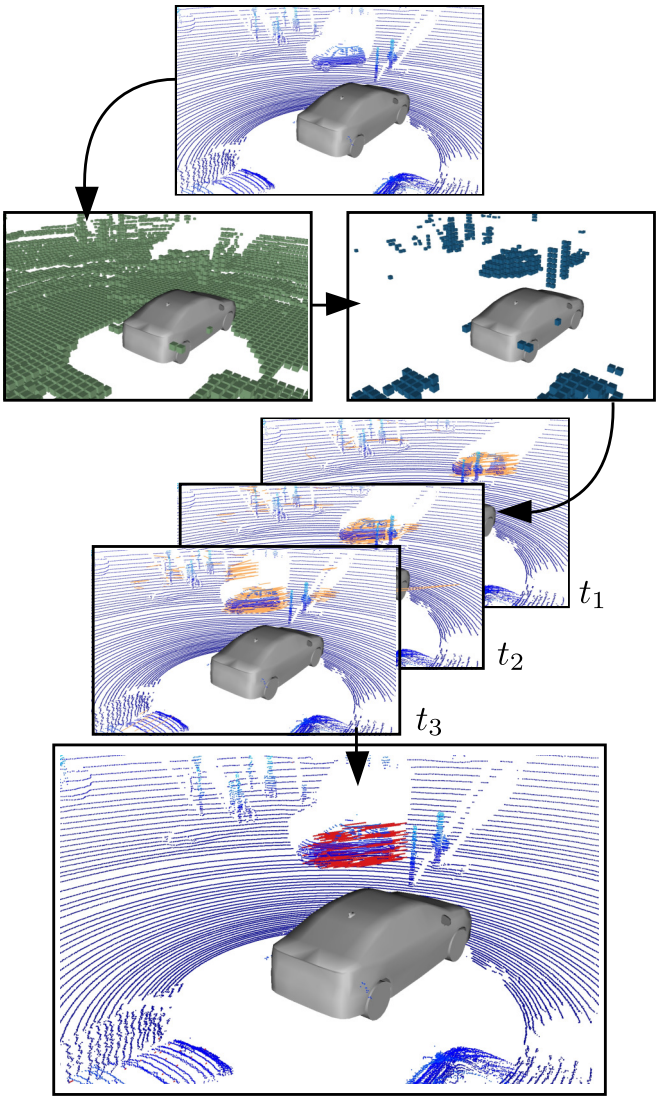


Fig. 1: An overview of our pipeline. As an input to our system, we take raw LIDAR scans. We construct occupancy grids from these scans, which are then filtered to remove the background. We then compute raw scene flow measurements using our occupancy constancy metric. Finally, we incorporate this measurement in a filtering framework to refine the estimate and reject false measurements, producing an estimate of temporal scene flow.

In this paper, we present a pipeline that takes raw LIDAR data as input and produces a scene flow estimate. In our previous work, we considered the similar problem of obstacle

This work was supported by a grant from the Toyota Research Institute under award N021515.

A. Ushani and R. Eustice are with the University of Michigan, Ann Arbor, MI 48109, USA. {aushani, eustice}@umich.edu.

R. Wolcott and J. Walls are with the Toyota Research Institute. {rwolcott, jmwalls}@umich.edu.

tracking [1]. Motivated by some of the shortcomings in this previous work, a key goal of our proposed method here is to avoid relying on temporal data association, segmentation, or use of an object model. Instead, we formulate the problem similarly to that of optical flow over 3D occupancy grids. The novel contributions of this work include:

- A learned framework for tracking LIDAR observations in occupancy grids, leveraging background subtraction and temporal occupancy constancy.
- An expectation-maximization (EM) algorithm for estimating raw, incremental scene flow.
- Real-time implementation on a GPU enabling 10 Hz scene flow estimation.
- Extensive evaluation of our method against known groundtruth from the KITTI dataset [9].

II. Related Work

Estimating dynamic motion from sensor data has been studied extensively in various communities, including computer vision, self driving vehicles, and mobile robotics.

In the field of computer vision, optical flow or scene flow has been a popular research area in which motion in an image due to a moving platform or dynamic objects in the environment is estimated. While motion estimation in camera data and LIDAR have many similarities, it is important to note the unique advantages and challenges that each sensor modality provides. For example, unlike LIDAR sensors which provide a relatively sparse set of observations, cameras provide a rich view of the scene. In addition to denser measurements, cameras also provide a much more accurate estimate of the pixel appearance (such as color or pixel intensity). While some LIDAR sensors do report the intensity of the laser return, these intensity values are usually not very discriminative in general and unreliable, as they could vary significantly due to incidence angle or between laser sensors.

Optical flow is typically approached by solving for a two dimensional motion field in the image plane that preserves some constancy metric (such as brightness constancy) and a regularization term to promote spatially smooth flow [10, 11]. In scene flow, the 3D motion is estimated with the use of a stereo camera or some depth sensor [12–15]. However, scene flow is generally not capable of real-time performance: 12 of the 15 submissions to the KITTI scene flow evaluation benchmark, including the top four, currently take 50 s or longer to process a single scene [12]. Recently, Jaimez et al. [15] presented a method to estimate scene flow in real-time for displacements up to 15 cm. However, due to the difference in sensing modalities described above, these methods do not directly translate from computer vision to LIDAR sensing.

While there are some key differences, obstacle tracking considers a similar problem. Whereas we are interested in sensing and detecting motion in general in a dynamic environment, in obstacle tracking, discrete objects are extracted from sensor data and detected over time. These observations of objects are associated temporally and used to compute trajectories or build appearance models. Many obstacle tracking methods rely on simple geometric models of obstacles, such as boxes or ellipses, which are fit to measurements through time [3–7]. More recently, some methods do not make assumptions about the obstacle's appearance or structure. In our previous work, we framed obstacle tracking as a problem similar to that of SLAM, in which an obstacle's trajectory and "map" (i.e., point cloud model) are computed [1]. Held et al. [16] provide a framework by which to efficiently register successive scans of an obstacle, providing an estimate of relative motion between these two scans and incrementally building up an obstacle model. Unlike our problem area, however, many approaches in obstacle tracking assume that the sensor data has been segmented into discrete objects and associated over time.

More recently, Dewan et al. [17] consider a similar problem to ours, estimating rigid scene flow between LIDAR scans. Similar to our work, they formulate an energy minimization problem based on matching SHOT feature descriptors for a subset of keypoints. However, unlike our work, they rely on point correspondences, whereas we do not rely on any data association. Additionally, they do not temporally filter this result over successive scans.

Our filtering framework has some similarities to that of Tanzmeister et al. [18] and Danescu et al. [19], where particles with position and speed are spread throughout a two dimensional grid and can move from cell to cell. They rely on stereovision to produce positional observations that are used in the resampling step for the particle filter. Aside from the difference in sensing modality, we attempt to directly exploit perceived motion in the sensor data.

III. Problem Statement

Our goal is to compute scene flow from LIDAR scans in real-time. Our work has direct application for autonomous vehicles, so we make the assumption that the world is locally planar—as is common in many autonomous vehicle applications. Namely, we assume that object motion is restricted to this horizontal plane and that all objects in a vertical column tangent to this plane move together. It is important to note that we make this assumption for our application domain to help run in real-time, though our proposed method could be modified to compute non-horizontal motion if need be. Thus, our goal is to compute the temporal scene flow $s_{i,j}$ for every location (i, j) in the plane.

Many sensor observations in a LIDAR scan correspond to static background structure in which scene flow estimation is trivial (e.g., the ground plane). As autonomous vehicles are commonly instrumented to estimate odometry, scene flow of static structures can instead be estimated via the relative motion of the car. Thus, our work is primarily interested in estimating the scene flow for dynamic objects or potentially dynamic objects in the environment.

In the following sections, we describe the stages of our framework. Beginning with a point cloud, $\mathbf{z}_{t,1:n} = \left\{[x_i, y_i, z_i]^\top\right\}_{i=1}^{n}$ that we receive from the LIDAR sensor

at time t, we first construct a 3D occupancy grid G_t. We will consider both $\mathbf{z}_{t,1:n}$ and G_t in the reference frame of the vehicle platform at time t. We then use a learned background filter to extract the foreground from G_t. Next, we estimate the scene flow between two successive occupancy grids G_{t-1} and G_t using a learned approach. We compute this flow in the reference frame of the vehicle, although this could easily be converted to the world frame using the odometry estimate of the ego-motion of the vehicle platform. Finally, we accumulate these raw scene flow measurements into a filtering framework to provide an estimate of the temporal scene flow.

IV. LIDAR Preprocessing

In this section, we describe the preprocessing performed on the LIDAR sensor data, $\mathbf{z}_{t,1:n}$, to create an occupancy grid G_t.

A. Occupancy Grid

To process the LIDAR data, we first build an occupancy grid [20, 21], composed of 3D voxels representing how likely it is that an object occupies the given space. For each voxel, the probability it is occupied is computed by

$$p(v|\mathbf{z}_{t,1:n}) = \left[1 + \frac{1-p(v|\mathbf{z}_{t,n})}{p(v|\mathbf{z}_{t,n})}\frac{1-p(v|\mathbf{z}_{t,1:n-1})}{p(v|\mathbf{z}_{t,1:n-1})}\beta\right]^{-1}, \tag{1}$$

where $\mathbf{z}_{t,1:n} = \{\mathbf{z}_{t,1}, \ldots, \mathbf{z}_{t,n}\}$ is the collection of laser returns from the LIDAR sensor, $\beta = \frac{p(v)}{1-p(v)}$, and $p(v)$ is a prior on the state of v. If we assume that our prior is $p(v) = 0.5$ and use log-odds (denoted L) [21], we can represent the recursive formulation of (1) as

$$L(v|\mathbf{z}_{t,1:n}) = \sum_{i=1}^{n} L(v|\mathbf{z}_{t,i}) \tag{2}$$

$$L(v|\mathbf{z}_{t,i}) = \begin{cases} l_{\text{free}} & \text{the ray to } \mathbf{z}_{t,i} \text{ passes through } v \\ l_{\text{occupied}} & \text{the ray to } \mathbf{z}_{t,i} \text{ ends in } v \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $L(v|\mathbf{z}_{t,i})$ is the log-odds update given by the observation $\mathbf{z}_{t,i}$ and l_{free} and l_{occupied} are the log-odds updates given by an observation of free or occupied space, respectively. To compute all the of log-odds updates for all of our LIDAR observations, we use Bresenham's ray tracing algorithm [22], which can be efficiently implemented on the GPU.

B. Background Filter

In order to reduce the computational burden of computing flow for the entire scene, we first apply background subtraction to identify static structure. This preprocessing step identifies possible columns in the occupancy grid that could be dynamic and have some scene flow associated with them. While this filter need not be perfect, we find that it helps with runtime performance and errors due to perceptual aliasing.

Many techniques that use LIDAR sensors rely on various methods to eliminate static background (e.g., ground plane or buildings). One set of approaches rely on prior maps of the environment that are leveraged by localizing within these maps during runtime. These maps contain information about the ground plane or other static structure in the scene. For example, some obstacle trackers rely on these maps to identify only the sensor measurements from objects that need to be tracked [1]. However, these approaches fail if the map is not accurate (e.g., due to construction) or the localization system experiences any errors.

Other approaches leverage the appearance and structure of the data to discriminate between foreground and background, such as [23, 24]. Features such as normal vectors and shape distributions are extracted from patches of points to create a handcrafted feature vector which is fed into a classifier. Wang et al. [23] report a runtime of 5 s on 3D data, while Wang et al. [24] report a runtime of 336 ms on 2D data. As our use requires a filter that is part of a full system that can run in under 100 ms, we propose a method that is simpler in nature but can run much faster than these previous techniques while still operating on our 3D occupancy grid.

We propose to solve this task via classification using a logistic classifier. Given a location (i, j) in our occupancy grid, we seek to find a label $y \in \{\text{Foreground}, \text{Background}\}$. This method requires no prior maps and allows us to learn from training data the structure of columns that appear to be dynamic.

For the occupancy column (i, j), we build a binary feature vector as follows. We extract 5×5 neighborhood patch of columns, $N_{i,j}$, from the occupancy grid around the location (i, j). We build two binary feature vectors $\mathbf{s}_{\text{free}}$ and $\mathbf{s}_{\text{occu}}$, each encoding the state of the full neighborhood patch. The binary elements of $\mathbf{s}_{\text{free}}$ and $\mathbf{s}_{\text{occu}}$ are given by

$$\mathbf{s}^n_{\text{free}} = p(v_n) < (0.5 - \epsilon) \tag{4}$$

$$\mathbf{s}^n_{\text{occu}} = p(v_n) > (0.5 + \epsilon), \tag{5}$$

for every voxel $v_n \in N_{i,j}$, where ϵ is a parameter controlling the certainty of free or occupied space. These vectors are concatenated to make one binary feature vector $\mathbf{s} = \left[\mathbf{s}^n_{\text{free}}{}^\top, \mathbf{s}^n_{\text{occu}}{}^\top\right]^\top$, which is then used in a logistic classifier. Note that this feature vector implicitly captures the notion of unknown cell state where these two decision variables both evaluate to false.

To train this classifier, we extract training data from the KITTI dataset. For ten KITTI log sequences of data, we build feature vectors by sampling columns in the occupancy grid and extract labels by determining whether or not these columns are contained within the labeled KITTI tracklet data. We build a training set of 243,828 samples. We use TensorFlow [25] to train our logistic classifier. Finally, we choose our decision threshold for this classifier to give us 95% accuracy on extracting the foreground; this allows us to reject much of the static background without catastrophically removing dynamic columns.

Our background filter can then be used online by running the classifier for every column of the occupancy grid. This step is embarrassingly parallel, and thus can be implemented

very efficiently on the GPU. For any columns that are identified to be part of the background, we assign a raw scene flow measurement derived from the odometry estimate of the ego-motion of the vehicle platform.

V. Temporal Scene Flow Computation

In this section, we will describe the process by which we compute temporal scene flow between two occupancy grids G_{t-1} and G_t. We use the background filter described above to filter G_{t-1}, but importantly we do not filter G_t at this time. This helps mitigate errors in the background filter, as voxels that were not properly filtered in G_{t-1} can still be matched to the corresponding location in G_t.

A. Occupancy Constancy

To compute the scene flow between two occupancy grids, we need a method by which we can measure the consistency of the occupancy state of columns in successive occupancy grids. To do so, we will rely on *occupation constancy*: the occupation state of matching columns should not change between two successive scans. Here, we assume rigid, non-deforming motion, which is generally valid in our application. As we will demonstrate in the results, our method can still achieve good performance even for objects that can deform, such as cyclists and pedestrians.

We formulate occupation constancy as a learning problem, as we find that a learning approach works significantly better than any hand designed metric. We take two candidate columns $c_{t-1} \in \mathrm{G}_{t-1}$ at (i_{t-1}, j_{t-1}) and $c_t \in \mathrm{G}_t$ at (i_t, j_t), each consisting of a vertical array of voxels $v_{i,j,k}$ in the occupancy grid at their respective location. We wish to determine whether or not they are consistent. We construct three binary feature vectors, $\mathbf{f}_{\text{free}}$, $\mathbf{f}_{\text{occu}}$, and $\mathbf{f}_{\text{diff}}$, encoding whether or not the two columns have similar occupation. Each element k of these feature vectors are given by

$$\mathbf{f}_{\text{free}}^{(k)} = \big(p(v_{c_{t-1},k}) < (0.5 - \epsilon)\big) \wedge \big(p(v_{c_t,k}) < (0.5 - \epsilon)\big) \tag{6}$$

$$\mathbf{f}_{\text{occu}}^{(k)} = \big(p(v_{c_{t-1},k}) > (0.5 + \epsilon)\big) \wedge \big(p(v_{c_t,k}) > (0.5 + \epsilon)\big) \tag{7}$$

$$\begin{aligned}\mathbf{f}_{\text{diff}}^{(k)} = &\Big(\big(p(v_{c_{t-1},k}) > 0.5 + \epsilon\big) \wedge \big(p(v_{c_t,k}) < 0.5 - \epsilon\big)\Big) \\ &\vee \Big(\big(p(v_{c_{t-1},k}) < 0.5 - \epsilon\big) \wedge \big(p(v_{c_t,k}) > 0.5 + \epsilon\big)\Big).\end{aligned} \tag{8}$$

We concatenate these three binary feature vectors into one binary feature vector $\mathbf{f}_{c_{t-1},c_t}$. This is then used in a logistic classifier, $P_{\text{match}}(\mathbf{f}_{c_{t-1},c_t})$, which yields the probability that the two given columns are consistent with each other given their occupation state.

To train this classifier, we again rely on the KITTI dataset to extract training data. We create a training dataset in the following manner. We sample a column $c_{t-1} \in \mathrm{G}_{t-1}$ at location (i, j). Using scan matching [26] for a ground truth relative pose estimate and the labeled KITTI tracklet data, we compute the true scene flow and find the true corresponding column $c_t \in \mathrm{G}_t$. Thus, c_{t-1} and c_t are used to construct a positive training sample. We then additionally sample from a $n_s \times n_s$ neighborhood about location (i, j) in G_t (denoted by $N_{c_{t-1}}$), excluding the true corresponding column, to construct negative training samples.

We use the same ten KITTI log sequences of data as we did before to build our training dataset, comprising of 1,028,381 training samples. We train our logistic classifier for occupancy constancy using TensorFlow.

For efficient computation, we preprocess P_{match} for all pairs of columns we will consider. We take the log-probability and apply a window to promote spatial smoothness (e.g., $c_{i,j}$ and $c_{i+1,j+1}$ will likely experience similar scene flow). We store this result in a lookup table T_{match},

$$T_{\text{match}}(c_{t-1}, c_t) = \sum_{c_w \in W_{c_{t-1}}} \log\big(P_{\text{match}}(\mathbf{f}_{c_w,c_t})\big), \tag{9}$$

where $W_{c_{t-1}}$ is $n_w \times n_w$ window of columns $c_w \in \mathrm{G}_{t-1}$ centered on c_{t-1}.

B. Raw Scene Flow Computation

To solve for the scene flow, we formulate an energy minimization problem that leverages our learned occupancy constancy metric. We use an iterative EM algorithm to estimate a locally rigid, non-deforming flow between successive occupancy grids, which we run for n_{em} iterations.

At each step of the EM algorithm, for every $c_{t-1} \in \mathrm{G}_{t-1}$ we maintain the current estimate of scene flow $s(c_{t-1})$ and matched column $m(c_{t-1}) \in \mathrm{G}_t$, both initially all flagged as invalid. We will compute an energy, $E(c_{t-1}, c_t)$, associated with a potential scene flow estimate that leads from $c_{t-1} \in \mathrm{G}_{t-1}$ to $c_t \in \mathrm{G}_t$. Additionally, for every $c_t \in \mathrm{G}_t$, we store the energy associated with the currently computed scene flow estimate that leads to $c_t \in \mathrm{G}_t$, $\hat{E}(c_t)$, which is initialized to ∞.

This algorithm must be run for every column in the occupancy grid. However, we can process columns in parallel, and thus this can be efficiently implemented on the GPU.

1) Expectation: During the expectation step, we estimate the most likely scene flow for every column. For a given column $c_{t-1} \in \mathrm{G}_{t-1}$ at location (i, j), we search through a neighborhood $N_{c_{t-1}}$, which is a $n_s \times n_s$ neighborhood of columns $c_t \in \mathrm{G}_t$ centered around location (i, j). We compute an energy for each $c_t \in N_{c_{t-1}}$,

$$\begin{aligned}E(c_{t-1}, c_t) = &-T_{\text{match}}(c_{t-1}, c_t) \\ &+ w_p \sum_{c_p \in P_{c_{t-1}}} \left\| s_{c_{t-1},c_t} - s(c_p) \right\|^2,\end{aligned} \tag{10}$$

where w_p is an L2 penalty weight to enforce spatial smoothness, $P_{c_{t-1}}$ contains all $c_p \in \mathrm{G}_{t-1}$ in a 5×5 neighborhood around c_{t-1} for which we have a valid scene flow estimate $s(c_p)$, and s_{c_{t-1},c_t} is the candidate scene flow vector which associates c_{t-1} and c_t.

For every $c_{t-1} \in \mathrm{G}_{t-1}$, we find the corresponding $c_t^* \in \mathrm{G}_t$ that minimizes $E(c_{t-1}, c_t)$ such that either $E(c_{t-1}, c_t^*) < \hat{E}(c_t^*)$ or $m(c_{t-1}) = c_t^*$ (i.e., the same scene flow was computed previously during an earlier iteration and we

simply need to update the energy that may have changed due to the L2 penalty). We store the estimated scene flow at this iteration in $s(c_{t-1}) = s_{c_{t-1},c_t^*}$ and $m(c_{t-1}) = c_t^*$.

2) Maximization: During the maximization we step, we enforce our assumption of locally rigid, non-deforming flow. This means that only one column $c_{t-1} \in \mathrm{G}_{t-1}$ can lead to any column $c_t \in \mathrm{G}_t$.

For each $c_t \in \mathrm{G}_t$, we consider all $c_{t-1} \in \mathrm{G}_{t-1}$ such that $m(c_{t-1}) = c_t$. If there are any such c_{t-1}, we take the one with the lowest energy $E(c_{t-1}, c_t)$, denoted c_{t-1}^*, essentially picking the most likely column in the previous scan that leads to c_t. We set $\hat{E}_c(c_t) = E(c_{t-1}^*, c_t)$, and we flag all other $m(c_{t-1}) = c_t$ and $s(c_{t-1})$ where $c_{t-1} \neq c_{t-1}^*$ as being invalidated.

C. Temporal Scene Flow

We further refine this raw scene flow measurement by incorporating it in a filtering framework. We maintain a two dimensional array of *flow tracklets*, each maintaining a temporally filtered estimate of the scene flow at the given position (i, j). This step in our procedure is somewhat similar to the work presented in [18].

Each flow tracklet is essentially an EKF filter using a constant velocity model. We represent the state as $\mathbf{s} = [x, y, \theta, v, \dot{\theta}]^\top$. We incorporate the raw flow measurements by treating them as x, y observations, with a covariance determined by the resolution of the occupancy grid. We use Mahalanobis gating to reject any flow measurements that are outliers.

After we compute the raw scene flow between G_{t-1} and G_t, we process this result with our two dimensional array of flow tracklets. First, we use a constant velocity process model to update all of our flow tracklets to the current time. Then, each raw scene flow measurement for each location (i, j) is assigned to a flow tracklet at that location and used for the EKF filter's observation update. If no such flow tracklet exists at the location (i, j), a new one is created. Flow tracklets without a valid raw scene flow measurement (for example, due to Mahalanobis gating or background filtering) are discarded. Finally, all flow tracklets are moved to their new location (i, j) in the two dimensional array as given by the scene flow.

Not only does maintaining this filter better estimate the scene flow, but it helps eliminate outliers, both due to the background filter missing static background and any erroneous raw scene flow values. Additionally, for each flow tracklet, we maintain a count of many how observations it has received, or its "age". As flow tracklets increase in age and incorporate more raw observations, their estimate of the scene flow becomes more reliable.

VI. Results

We evaluated our proposed method using the KITTI dataset [9], using sequences of raw data from city driving. For computation, we ran all of our experiments on a machine with an Intel i7-4790K CPU with 16 GB of memory and a NVIDIA GeForce GTX 1080 GPU.

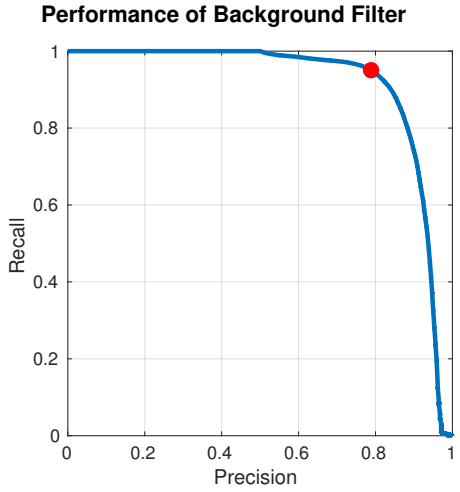


Fig. 2: Precision-recall curve of the background filter. The red dot on the curve indicates the decision threshold chosen throughout our evaluation, where we detect 95% of the foreground.

A. Parameter Selection

For the occupancy grid, we construct a $50\ \mathrm{m} \times 50\ \mathrm{m}$ grid centered on the vehicle at a resolution of $30\ \mathrm{cm}$.

To construct our binary feature vectors, we use $\epsilon = 0$. Note that this does not make our feature vectors complimentary. Any voxel v that represents unknown space will have $p(v) = 0.5$, and thus will not factor into any of the binary features due to the strict inequality in the decision variable. This is in fact what allows us to implicitly account for unknown space.

We use a search space size of $n_s = 31$, allowing for a relative motion estimation of up to $45\ \mathrm{m/s}$. We use a window size of $n_w = 3$. We run our EM algorithm for $n_{em} = 20$ iterations. For the L2 penalty weight, we use a value of $w_p = 1$.

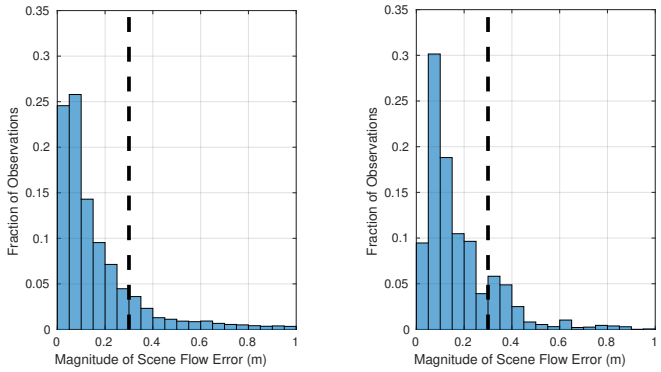
B. Background Filter

We first evaluate the performance of our background filter. We show the precision-recall curve by sweeping out the decision threshold for our classifier, as shown in Fig. 2; here the foreground is the positive class. As discussed in §IV-B, we choose a decision threshold for our classifier to achieve a 95% accuracy rate on foreground (as indicated by the *red* dot). This results in a 74.5% accuracy on the background.

While our background filter may not achieve state-of-the-art performance, it still performs quite well with respect to other methods [23, 24]. However, a core strength of our method is in efficiency, where we are able to execute extremely fast, taking only $1.2\ \mathrm{ms}$ to run.

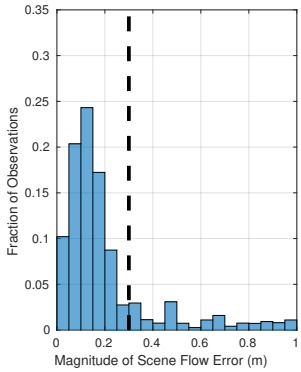
C. Raw Scene Flow

Next, we evaluate our raw scene flow measurements. To generate the ground truth scene flow values, we combine the KITTI labeled tracklet data with the relative pose between the two successive LIDAR scans, as computed by scan matching [26]. We then compute the norm of the error between the ground truth flow vector and our computed raw

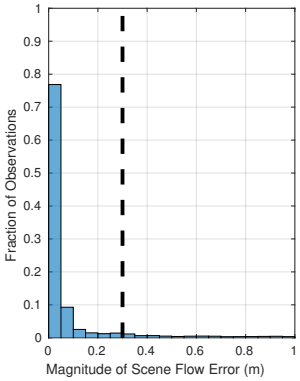


(a) Car

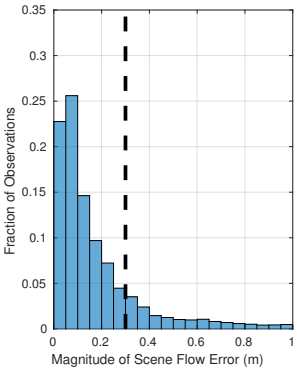
(b) Cyclist



(c) Pedestrian



(d) Background



(e) All labeled KITTI tracklets

Fig. 3: Error histograms for raw scene flow measurements, by class, and for all labeled KITTI tracklets. The dashed vertical line in all plots indicates the resolution of our occupancy grid. Note the different vertical scale for Fig. 3d.

scene flow. Results are presented as error histograms by class in Fig. 3 and computed error statistics as tabulated in Table I.

We find that our raw scene flow measurements are generally quite accurate. While there are a number of outliers that arise from alaising of occupancy constancy, the error is frequently less than the resolution of the occupancy grid. Additionally, we find that we are still able to compute scene flow for objects that violate our assumption of rigid, non-deforming motion such as cyclists or pedestrians.

Class	Count	Median Error	Mean Error	Percent Within 30 cm
Car	873,947	10.2 cm	19.3 cm	83.8%
Cyclist	9,890	13.2 cm	24.5 cm	78.8%
Pedestrian	7,527	15.5 cm	38.9 cm	74.3%
Background	19,899,708	2.6 cm	14.9 cm	88.9%
All Labeled Tracklets	1,126,264	11.0 cm	22.1 cm	81.4%

TABLE I: Error statistics for the raw scene flow measurements. We perform this analysis by class and overall labeled tracklets in the KITTI datasets. Note that the background can have non-zero error due to inaccuracies in the background filter, noisy odometry, or discretization effects.

Class	Count	Median Error	Mean Error
Car	216,026	0.49 m/s	0.65 m/s
Cyclist	1,507	1.01 m/s	1.09 m/s
Pedestrian	537	0.86 m/s	0.93 m/s
All Labeled Tracklets	266,237	0.50 m/s	0.66 m/s

TABLE II: Error statistics for the temporal scene flow measurements for all flow tracklets with an age of at least 10. We perform this analysis by class and over all labeled tracklets in the KITTI datasets.

D. Temporal Scene Flow

Finally, we evaluate the temporal scene flow as computed by our flow tracklets against the same benchmark as discussed in §VI-C. We evaluate this error as the tracklets age, which allows for the filtered scene flow estimate to become more and more accurate. The age of each tracklet as measured in these results is simply the number of scans that the tracklet has been active for (i.e., each age step corresponds to 0.1 s of sensor data). We compute the CDF of the norm of the velocity error vector for each flow tracklet for various ages and also for all tracklets that have been seen for at least a full second. We show these results in Fig. 4 and present error statistics in Table II.

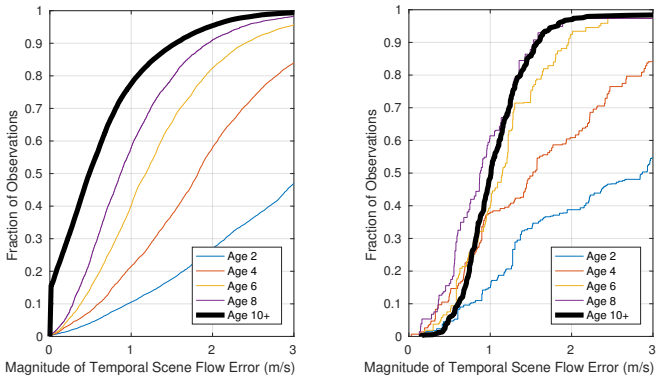
Unsurprisingly, the accuracy of the temporal scene flow steadily improves as tracklets get older and receive more observations. By the time a flow tracklet has been observed for at least a second, the median error is roughly 0.5 m/s, which corresponds to one sixth of the resolution of our occupancy grid.

A few sample temporal scene flow results are shown in Fig. 5.

E. Runtime Performance

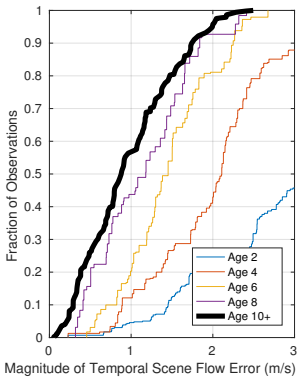
A key goal is to enable real-time use of our proposed framework. We provide a thorough runtime analysis of our algorithm and all the core steps. As the LIDAR data is received at 10 Hz, our algorithm must run in under 100 ms to achieve real-time performance. The mean runtimes and standard deviations over the full set of KITTI city log sequences are provided in Table III.

We find that most of the steps of the pipeline are fairly consistent in runtime with the exception of the iterative EM procedure. This can be attributed to scene variation, as we only perform the EM procedure for columns that have not been pruned by the background filter. If we were to

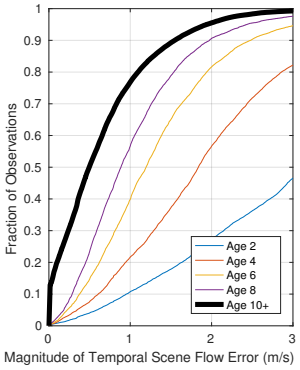


(a) Car

(b) Cyclist



(c) Pedestrian



(d) All labeled KITTI tracklets

Fig. 4: The CDF of the error of our temporal scene flow estimates, by class, and for all labeled KITTI tracklets. Note that these results are measured in terms of velocity ($\mathrm{m/s}$) as opposed to the results in Fig. 3 which are measured in terms of position.

Step	Runtime (ms)	Standard Deviation (ms)
Occupancy Grid Generation	12.5	0.87
Background Filter	1.2	0.10
Occupancy Constancy	41.4	0.83
Iterative Expectation-Maximization	28.5	5.53
Filtered Temporal Flow	0.4	0.43
Total	85.7	6.38

TABLE III: Runtime performance of our proposed method. We present both the total runtime and the runtime for each key step.

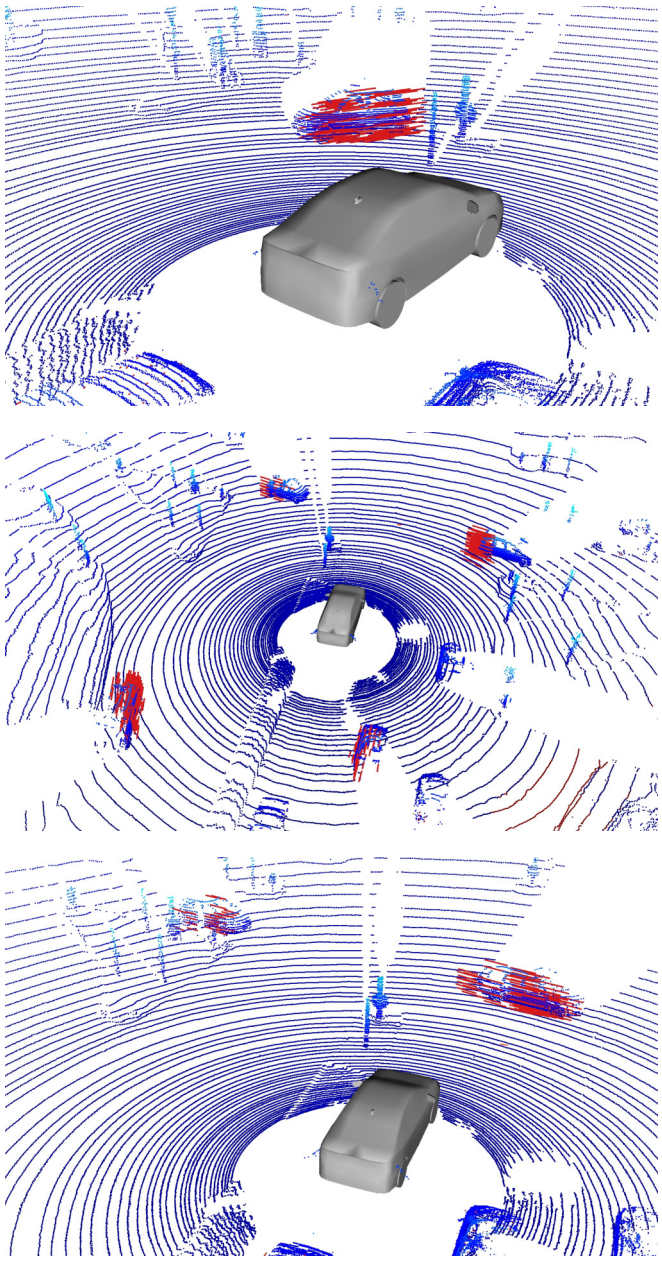


Fig. 5: Sample temporal scene flow results. The temporal scene flow is depicted in red.

perform the EM procedure for all columns in the scene, the runtime would increase by approximately $35\ \mathrm{ms}$, putting us just beyond our allotted $100\ \mathrm{ms}$ for real-time performance without decreasing the number of iterations.

Nevertheless, we find that for over 99% of the input data, the total runtime is under $100\ \mathrm{ms}$, achieving real-time performance.

VII. Discussion

We find that our estimate of scene flow performs quite well. Between 74.3% and 83.8% of the KITTI tracklets, depending on the class, produce raw measurements of scene flow that are within $30\ \mathrm{cm}$ of the ground truth flow, which is the resolution of our occupancy grid. For static background structure, even though our background filter is conservative, we are able to extract scene flow accurately.

It is interesting to compare our results for raw scene flow with Dewan et al. [17]. Like our work, they evaluate their method on the KITTI dataset, although they only provide results for translation error for sequences with no moving objects. Although our method has some differences in formulation, most notably the use of an occupancy grid as opposed to directly dealing with the points, the results are comparable. However, we do not rely on point correspondences or any data association between scans.

Unsurprisingly, our results are even more impressive when filtered over time. Our temporal model of scene flow allows us to quickly improve the accuracy of the estimate over time by both rejecting outliers and filtering measurements.

This quickly achieves accurate sub-voxel level estimates of temporal scene flow.

While our algorithm is not an obstacle tracker, it is interesting to compare the accuracy of our temporal scene flow estimate to the performance of full fledged obstacle tracking algorithms. State of the art obstacle trackers such as [1] and [27] report average velocity errors of between $0.314\ \mathrm{m/s}$ and $0.56\ \mathrm{m/s}$, respectively. Despite the fact that we do not make similar assumptions as these trackers, such as accurate segmentation, our error metrics are close to these marks. Additionally, we are not prone to errors due to not correctly modeling free space as shown in [1]. However, these obstacle trackers are able to provide refined trajectory estimates and crisp point cloud models of tracked obstacles, something our method does not aim to achieve.

It is important to note that the results we have presented for temporal scene flow are for individual flow tracklets. A dynamic object in the scene, such as a car, nominally induces several flow tracklets moving together. If we were to assume accurate segmentation, we could take cluster groups of flow tracklets together and combine their temporal flow estimates to produce an even more accurate result.

One key feature of our proposed method is the extendibility to other sensor modalities or environments. Occupancy grids can easily handle measurements from sensors other than LIDAR, such as radar or camera systems. Accounting for different environments or sensor configurations can be handled by simply retraining the background filter and occupancy constancy metric with new data. This is unlike many other approaches for autonomous vehicle applications that have a heavy reliance on prior maps, such as [1].

Unlike many other methods which estimate scene flow, our timing results demonstrate the real-time capability of our algorithm. Our method is capable of consuming LIDAR data at $10\ \mathrm{Hz}$, which is the nominal rate for such sensors, and thus is appropriate for online use with autonomous systems.

VIII. Conclusion

We have presented an end-to-end pipeline for consuming LIDAR data and producing estimates of temporal scene flow. We have demonstrated the performance of this algorithm on the KITTI dataset, presenting results that are competitive with or better than the current state of the art. We have shown that this algorithm can be run at $10\ \mathrm{Hz}$, enabling real-time use for mobile robotic applications.

References

[1] A. K. Ushani, N. Carlevaris-Bianco, A. G. Cunningham, E. Galceran, and R. M. Eustice, "Continuous-time estimation for dynamic obstacle tracking," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, Hamburg, Germany, Sep. 2015, pp. 1137–1143.

[2] D. Held, J. Levinson, S. Thrun, and S. Savarese, "Robust real-time tracking combining 3d shape, color, and motion," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 30–49, 2016.

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

[4] R. Kaestner, J. Maye, Y. Pilat, and R. Siegwart, "Generative object detection and tracking in 3d range data," in *Proc. IEEE Int. Conf. Robot. and Automation*, St. Paul, MN, USA, 2012, pp. 3075–3081.

[5] M. Darms, P. Rybski, and C. Urmson, "Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments," in *Proc. Intell. Veh. Symp.*, Eindhoven, Netherlands, 2008, pp. 1197–1202.

[6] T.-D. Vu and O. Aycard, "Laser-based detection and tracking moving objects using data-driven markov chain monte carlo," in *Proc. IEEE Int. Conf. Robot. and Automation*, Kobe, Japan, 2009, pp. 3800–3806.

[7] M. Baum and U. D. Hanebeck, "Extended Object Tracking with Random Hypersurface Models," *IEEE Trans. on Aero. and Elec. Sys.*, vol. 50, pp. 149–159, 2014.

[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[10] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981.

[11] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Int. Joint Conf. Artif. Intell.*, Vancouver, BC, Aug. 1981, pp. 674–679.

[12] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, Boston, MA, USA, 2015, pp. 3061–3070.

[13] C. Vogel, K. Schindler, and S. Roth, "3d scene flow estimation with a piecewise rigid scene model," *Int. J. Comput. Vis.*, vol. 115, no. 1, pp. 1–28, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0806-0

[14] ——, "Piecewise rigid scene flow," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 1377–1384.

[15] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers, "A primal-dual framework for real-time dense rgb-d scene flow," in *Proc. IEEE Int. Conf. Robot. and Automation*, Chicago, IL, USA, 2015, pp. 98–104.

[16] D. Held, J. Levinson, and S. Thrun, "Precision tracking with sparse 3d and dense color 2d data," in *Proc. IEEE Int. Conf. Robot. and Automation*, Karlsruhe, Germany, 2013, pp. 1138–1145.

[17] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard, "Rigid scene flow for 3d lidar scans," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, Daejeon, Korea, 2016.

[18] G. Tanzmeister, J. Thomas, D. Wollherr, and M. Buss, "Grid-based mapping and tracking in dynamic environments using a uniform evidential environment representation," in *Proc. IEEE Int. Conf. Robot. and Automation*, Hong Kong, China, May 2014, pp. 6090–6095.

[19] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Trans. Intell. Transp. Sys.*, vol. 12, no. 4, pp. 1331–1342, 2011.

[20] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proc. IEEE Int. Conf. Robot. and Automation*, vol. 2, St. Louis, MO, USA, 1985, pp. 116–121.

[21] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Auton. Robot.*, vol. 34, no. 3, pp. 189–206, 2013.

[22] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems journal*, vol. 4, no. 1, pp. 25–30, 1965.

[23] D. Z. Wang, I. Posner, and P. Newman, "What could move? finding cars, pedestrians and bicyclists in 3d laser data," in *Proc. IEEE Int. Conf. Robot. and Automation*, St. Paul, MN, USA, 2012, pp. 4038–4044.

[24] ——, "Model-free detection and tracking of dynamic objects with 2D lidar," *Int. J. Robot. Res.*, vol. 34, no. 7, pp. 1039–1063, 2015.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[26] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Proc. Robot.: Sci. & Syst. Conf.*, Jun. 2009.

[27] D. Held, J. Levinson, S. Thrun, and S. Savarese, "Combining 3d shape, color, and motion for robust anytime tracking," in *Proc. Robot.: Sci. & Syst. Conf.*, Berkeley, CA, USA, 2014, pp. 1–8.