Feature Learning for Scene Flow Estimation from LIDAR

Arash K. Ushani
Computer Science and Engineering
University of Michigan Ann Arbor
United States
aushani@umich.edu

Ryan M. Eustice
Naval Architecture and Marine Engineering
University of Michigan Ann Arbor
United States
eustice@umich.edu

Abstract: To perform tasks in dynamic environments, many mobile robots must estimate the motion in the surrounding world. Recently, techniques have been developed to estimate scene flow directly from LIDAR scans, relying on hand-designed features. In this work, we build an encoding network to learn features from an occupancy grid. The network is trained so that these features are discriminative in finding matching or non-matching locations between successive timesteps. This learned feature space is then leveraged to estimate scene flow. We evaluate our method on the KITTI dataset and demonstrate performance that improves upon the accuracy of the current state-of-the-art. We provide an implementation of our method at https://github.com/aushani/flsf.

Keywords: feature learning, scene flow, LIDAR

1 Introduction

Mobile robots often operate in environments that are inherently dynamic. To operate safely, it is necessary for such systems to be aware of the motion in the world around them. For example, self driving cars need to be aware of other cars on the road, and warehouse robots must be able to move in an area with many other agents. Estimating dynamic motion is a core competency for these autonomous systems.

Many approaches to detect motion perform object tracking [1, 2, 3, 4, 5, 6]. Sensor data, such as a camera image or a light detection and ranging (LIDAR) point cloud, is segmented into distinct objects. Data association is performed across timesteps. From the location of an object at each timestep, its motion can be estimated.

Another approach to detect dynamic motion from a stream of sensor data is to solve for optical flow or scene flow [7, 8, 9, 10, 11, 12, 13, 14]. Traditionally computed from camera data, optical flow and scene flow approaches seek to find a motion field between two successive images by leveraging some sort of consistency metric (such as brightness constancy) in an optimization problem.

More recently, similar ideas have been applied to point cloud data [15, 16]. However, these are commonly reliant on a hand-designed feature or metric, such as SHOT features [15] or occupancy constancy [16].

In this work, we propose an encoding network that learns features from an input occupancy grid. The network is trained so that features from corresponding locations across different timesteps will be similar, and features from different locations will be far apart. Thus, we can leverage the learned feature space to estimate scene flow, with the hope that a learning approach can produce a feature space in which distances are more meaningful. By evaluating our proposed method on the KITTI dataset [17], we demonstrate that our approach yields results that beat the accuracy of the current state-of-the-art. The methods presented in this paper build upon our prior work [16].

2 Related Work

The work presented here is at the overlap of research done in the areas of dynamic object tracking, optical flow and scene flow, and feature learning.

2.1 Dynamic Object Tracking

Dynamic object tracking from LIDAR sensors is a well studied problem with similar goals as our work in estimating dynamic motion in the environment. Early approaches would segment the LIDAR scan and extract a geometric observation from the segments, such as a bounding box or the centroid of points [1, 2, 4]. These observations would be accumulated over time in a filtering framework. Later approaches would improve upon the motion estimate between successive timesteps. Feldman et al. [3] aligned segmented snapshots of objects between scans using iterative closest point (ICP). Held et al. [5] proposed annealing histograms to perform a bounded search to find the best relative motion that optimizes a measurement model. Ushani et al. [6] used a continuous stream of the LIDAR observations from an object in a smoothing framework to estimate its trajectory.

Despite the impressive results by these works, they are prone to catastrophic failure if there is an error in object segmentation or data association through time. Some methods were developed that did not realize on explicit segmentation or data association. Tanzmeister et al. [18] and Danescu et al. [19] propose grid based tracking systems, where particles move among cells in a grid and are updated according to the observations. Dequaire et al. [20] propose a recurrent neural network that predicts a future occupancy grid from LIDAR input.

2.2 Optical Flow and Scene Flow

In optical flow, 2D motion in the image plane is solved for using successive sensor observations, such as images from a camera sensor [7, 8]. Commonly, a constancy metric that enforces consistency between images (e.g., brightness constancy) is leveraged. With 3D information, such as from a depth sensor or a stereocamera, 3D motion can be estimated, known as scene flow. Many methods exist to estimate scene flow from camera data. Some methods leverage an initial oversegmentation followed by a CRF [9] or an energy minization framework [10, 11]. Jaimez et al. [12] rely on a primal-dual algorithm. Taniai et al. [13] identify regions that are inconsistent with the ego-motion of the platform and then build an energy minimization problem consisting of appearance, flow, prior, color, and smoothing terms. Behl et al. [14] leverage object recognition in a CRF model.

However, estimating scene flow from camera images is typically very slow. The KITTI Scene Flow Challenge provides a public benchmark, where entries estimate the scene flow between images in typical urban and suburban driving scenarios [17]. The current top nine submissions to the challenge take longer than 5 minutes to run, with three reporting runtimes of nearly an hour.

Recently, similar techniques have been applied to LIDAR point clouds. Dewan et al. [15] use an energy minimization problem to estimate rigid scene flow between LIDAR scans that leverages SHOT feature descriptors. Ushani et al. [16] also frame an energy minimization problem, but instead leverage "occupancy constancy", measuring the consistency of the occupancy states between successive occupancy grids built from the LIDAR point clouds.

2.3 Feature Learning

Feature learning approaches have found success in many areas of robotics. This includes face recognition [21], long term image matching [22], and point cloud classification [23, 24]. Generally, these approaches construct a network from the input data to the feature space, and use a loss function to promote the separability of the features for the desired task. For example, Wen et al. [21] proposed center loss, where features from the same class are pulled towards the same center, and centers from different classes are forced to stay apart. Carlevaris-Bianco and Eustice [22] used a loss function that increases as the Euclidean distance between matching feature pairs grows, and decreases as the Euclidean distance between non-matching feature pairs grows.

More similar to our method, several hand designed approaches have been proposed, such as spin images [25] or Point Feature Histograms [26]. More recently, feature learning has been applied in this area. Notably, Zeng et al. [27] present 3DMatch. Patches are extracted from red, green, blue, and depth (RGB-D) reconstructions. Correspondences are collected from different views. A convolutional neural network (CNN) is then trained to learn a geometric descriptor that outperforms existing methods in determining correspondences. However, unlike our work, the dynamics of the scene are not considered.

3 Method

In this section, we describe our proposed method. We will make similar assumptions as other works in this area. Chiefly, as we target autonomous vehicle applications with the KITTI Dataset in this work, we will assume that all dynamic motion is in the horizontal plane and that the motion field is consistent for everything at the same (x, y) location. We seek to estimate the motion (u, v) for every 2D location $\mathbf{x} = (x, y)$ from one timestep to the next. That is to say, for a location $\mathbf{x}_t$ at time t, we seek to find the corresponding location at the next timestep, $\mathbf{x}_{t+1} = \mathbf{x}_t + (u, v)$ such that whatever was at $\mathbf{x}_t$ at time t is now at $\mathbf{x}_{t+1}$ at time $t+1$. We term these $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ to be matching locations (and non-matching otherwise).

Furthermore, we assume that the motion corresponds to locally rigid, non-deforming flow. Finally, since much of the environment is background structure that is static, we focus on our attention on the dynamic objects in the scene and assume that the flow of static objects can well estimated from odometry.

3.1 Input

Our system takes a point cloud $\mathrm{P}_t = \{\mathbf{z}_{t,1:n}\}$. We seek to construct an occupancy grid G_t that maps each (x, y, z) location to a probability that the given location is occupied [28, 29]. By doing so, we can properly model the true nature of the sensor by capturing free space, occupied space, and unknown space.

For each voxel $\mathbf{v}$ at location (x, y, z), the occupancy probability given the set of observations P_t is given by:

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

We assume an uninformative prior $p(\mathbf{v}) = 0.5$. Using log-odds notation, denoted by $LO(\,\cdot\,)$, we can rewrite (1) as

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

where

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

To compute G_t, this ray tracing is achieved using Bresenham's ray tracing algorithm [30]. Each observation $\mathbf{z}_{t,i}$ results in a ray tracing operation to find its log-odds updates for the corresponding voxels. After processing all observations, these updates are summed for all voxels to produce G_t. This algorithm is implemented on the GPU for efficient computation.

3.2 Network

The probabilities of the constructed occupancy grid G_t are first rescaled between -0.5 and 0.5 to produce $\tilde{\mathrm{G}}_t$. Note that any unknown voxels (i.e., $p(\mathbf{v}) = 0.5$) are scaled to 0 in $\tilde{\mathrm{G}}_t$. $\tilde{\mathrm{G}}_t$ is passed through a series of two dimensional convolution layers with leaky RELU activations, with kernel sizes and output layers as depicted in Fig. 1. This network has two outputs. First, we have an encoding output, F_t, that maps each (x, y) location to a N_f dimensional feature vector $\mathbf{f}_{(x,y)}$. Second, we have a classification output, C_t, that yields softmax scores for the foreground/background classifier for each location (x, y). We intentionally use only a single layer between the encoding output and

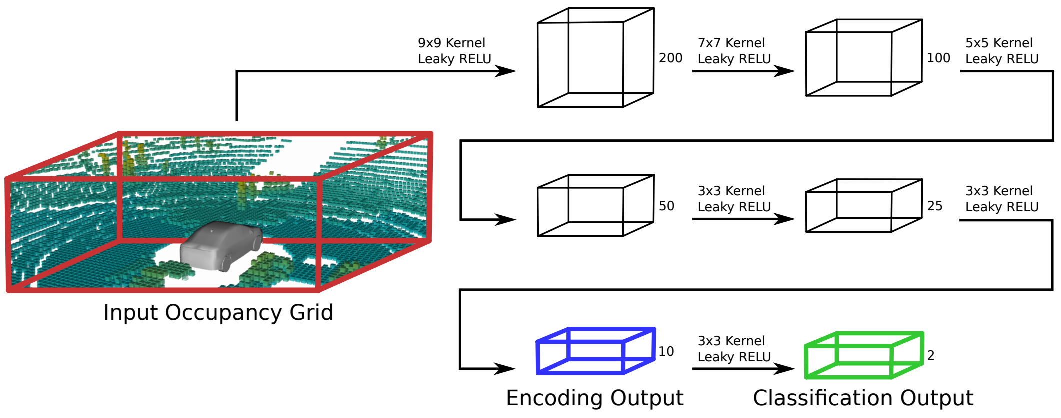

Figure 1: The network architecture. The input occupancy grid is passed through a series of convolution layers to produce a grid of feature vectors. This encoding is further passed through a single convolutional layer to produce softmax scores for the foreground/background classifier.

the filter output to help promote learning a discriminative feature space. Notably, our network does not reduce the resolution of the input occupancy grid (i.e., we produce a feature vector for every location in the grid).

The loss function for the encoding output is as follows. For locations $\mathbf{x}_1$ and $\mathbf{x}_2$ with feature vectors $\mathbf{f}_1$ and $\mathbf{f}_2$, respectively, the loss is given by

$$L_{\mathbf{f}_1,\mathbf{f}_2} = \begin{cases} \|\mathbf{f}_1 - \mathbf{f}_2\| & \mathbf{x}_1 \text{ and } \mathbf{x}_2 \text{ are matching locations} \\ d_{max} - \|\mathbf{f}_1 - \mathbf{f}_2\| & \mathbf{x}_1 \text{ and } \mathbf{x}_2 \text{ are non-matching locations, } \|\mathbf{f}_1 - \mathbf{f}_2\| < d_{max} \\ 0 & \text{otherwise} \end{cases}, \quad (4)$$

where $\|\cdot\|$ denotes the L2 norm. We choose $d_{max} = 10$.

The loss function for the classification output, L_{C}, is simply the mean weighted softmax cross entropy loss. As the training data contains far more instances of background than foreground, we weight the loss of each class by the inverse of their respective frequency. Thus, we have:

$$L_{\text{C}} = \frac{1}{n_{\text{samples}}} \left(\frac{1}{f_{\text{fg}}} \sum_{\mathbf{x} \in \text{foreground}} L_{\text{S.C.E.}}(\text{C}_{t,\mathbf{x}}) + \frac{1}{f_{\text{bg}}} \sum_{\mathbf{x} \in \text{background}} L_{\text{S.C.E.}}(\text{C}_{t,\mathbf{x}}) \right) \quad (5)$$

where n_{samples} is the number of labeled foreground or background locations in the training sample, f_{fg} and f_{bg} are the frequencies of foreground and background locations respectively in the training set, and $L_{\text{S.C.E.}}$ is the softmax cross entropy loss.

These loss functions are combined to form the total loss function,

$$L_{total} = L_{\mathbf{f}_1,\mathbf{f}_2} + L_{\text{C}}, \quad (6)$$

which is used to train the network.

While different approaches can be used to learn a feature space, such as an autoencoder or a Generative Adversarial Network (GAN), our approach allows for learning a feature space that is specifically tailored to be discriminative for the desired task.

3.3 Training

At each iteration, we train the network using one sample of locations $\mathbf{x}_1$ and $\mathbf{x}_2$ (which could be matching or non-matching) and one sample of a classification map C_t, simultaneously.

3.3.1 Training Data

We construct a training data set using the first ten KITTI city sequences. We step through each log and construct an occupancy grid from each velodyne point cloud, as described in §3.1.

Using the labeled KITTI tracklet data, we construct a 2D classification map for ground truth foreground/background classification. For any location $\mathbf{x}$ that is within the bounding box of a object labeled in the KITTI tracklets, we mark the location as foreground. Otherwise, we mark the location as background.

For the encoding output, we take successive occupancy grids G_1 and G_2. We first iterate through all foreground locations $\mathbf{x}_1$ from G_1 that are part of a labeled KITTI tracklet (i.e., the foreground). We then sample from a location $\mathbf{x}_2$ from G_2, chosen from an $n_s \times n_s$ neighborhood of locations centered on $\mathbf{x}_1$. Using the labeled KITTI tracklet data, we record whether $\mathbf{x}_1$ and $\mathbf{x}_2$ are matching or non-matching locations in the successive occupancy grids. $\mathbf{x}_1$ and $\mathbf{x}_2$ are sampled such that we have approximately an equal number of matches and non-matches. Additionally, we repeat this procedure for 1% of all background locations.

Note that the labeled KITTI tracklets are only valid in the field of view of the camera. Accordingly, we take care as to only such locations in our training data set. To help mitigate this, we augment our data with random rotations and reflections of the occupancy grids.

3.3.2 Training Procedure

We use TensorFlow's AdamOptimizer with a learning rate of 10^{-6} [31] to train the network using the loss function described in (6). During training, we use dropout at each layer in the encoding network with a dropout probability of 20%. Note that at the input, due to the rescaling of the occupancy grid, dropout essentially amounts to setting voxels from free or occupied (i.e., $p(\mathbf{v}) < 0$ or $p(\mathbf{v}) > 0$, respectively) to unknown (i.e., $p(\mathbf{v}) = 0$). The training parameters were empirically tuned.

We trained the network for approximately two days using a NVIDIA GeForce GTX TITAN X GPU.

3.4 Flow Computation

We take two successive point clouds, P_1 and P_2. From these, we construct occupancy grids, G_1 and G_2. Each is rescaled and passed through the network described in described in §3.2 using NVIDIA's cudNN to produce encoding ouptuts F_1 and F_2 and classification outputs C_1 and C_2. Note that as we deal with a stream of data, we can cache P_2, G_2, F_2, and C_2 for use at the next timestep for faster runtime performance.

For each location $\mathbf{x}_1$ from G_1, we consider the feature vector $\mathbf{f}_1$. We then consider a $n_s \times n_s$ window of locations $\mathbf{x}_2$ from G_2 around $\mathbf{x}_1$. For each $\mathbf{x}_2$ and $\mathbf{f}_2$, we compute the L2 norm between the two feature vectors and store this result in a lookup table,

$$T_{\text{distance}}(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{f}_1 - \mathbf{f}_2\| . \tag{7}$$

Note that T_{distance} is similar to T_{match} from the work of Ushani et al. [16]. The key difference is that T_{match} is rooted in a hand designed notion of occupancy constancy, whereas T_{distance} takes a fully learned approach to measure distances between feature vectors in our learned feature space.

From this point, we compute scene flow by using an iterative expectation-maximization (EM) algorithm similar to the one proposed by Ushani et al. [16], which we briefly summarize here.

We construct an energy minimization problem to compute the flow (u, v) for every location $\mathbf{x}_1$ from G_1. If, according to the classification output C_1, the given location is more likely to be static background structure, we flag $\mathbf{x}$ as such and assume that the flow can be estimated from odometry sensors that are measuring the relative motion of the platform. Thus, we focus our attention on foreground locations $\mathbf{x}$ that are dynamic (or could be dynamic).

We construct an iterative EM algorithm to estimate the flow. We use $\mathbf{x}_1$ to denote a location in G_1 and $\mathbf{x}_2$ for locations in G_2. $s(\mathbf{x}_1) = (u, v)$ denotes the current estimate of the scene flow at location $\mathbf{x}_1$. $m(\mathbf{x}_1) = \mathbf{x}_1 + s(\mathbf{x}_1)$ denotes the estimated matching location $\mathbf{x}_2$ in G_2 for $\mathbf{x}_1$. At the start of the algorithm, all $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ are marked as being invalid.

$E(\mathbf{x}_1, \mathbf{x}_2)$ denotes the energy of the flow from $\mathbf{x}_1$ to $\mathbf{x}_2$. $\hat{E}(\mathbf{x}_2)$ denotes the energy of the current flow estimate that leads to $\mathbf{x}_2$, initialized with ∞.

3.4.1 Expectation

During the expectation step, we seek the most likely flow (u, v) for every $\mathbf{x}_1$. We search through a $n_s \times n_s$ window of locations centered on $\mathbf{x}_1$, $N_{\mathbf{x}_1}$, in G_2. For each $\mathbf{x}_1$, $\mathbf{x}_2$, we compute an energy,

$$E(\mathbf{x}_1, \mathbf{x}_2) = T_{\text{distance}} + w_p \sum_{\mathbf{x} \in P_{\mathbf{x}_1}} \|(\mathbf{x}_2 - \mathbf{x}_1) - s(\mathbf{x})\|^2, \tag{8}$$

where w_p is a smoothness penalty weight and $P_{\mathbf{x}_1}$ is a 5×5 window around $\mathbf{x}_1$ of locations for which we currently have a valid estimate for scene flow.

We seek

$$\mathbf{x}_2^\star = \underset{\mathbf{x}_2}{\operatorname{argmin}}\, E(\mathbf{x}_1, \mathbf{x}_2), \tag{9}$$

subject to the conditions

$$E(\mathbf{x}_1, \mathbf{x}_2) < \hat{E}(\mathbf{x}_2) \tag{10}$$

or

$$m(\mathbf{x}_1) = \mathbf{x}_2. \tag{11}$$

We then update $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ with $\mathbf{x}_2^\star - \mathbf{x}_1$ and $\mathbf{x}_2^\star$, respectively. If no such $\mathbf{x}_2^\star$ is found, the estimated flow is marked as being invalid.

3.4.2 Maximization

For each $\mathbf{x}_2$, we consider all $\mathbf{x}_1$ with $m(\mathbf{x}_1) = \mathbf{x}_2$. If there is at least one, we find

$$\mathbf{x}_1^\star = \underset{\mathbf{x}_1}{\operatorname{argmin}}\, E(\mathbf{x}_1, \mathbf{x}_2), \tag{12}$$

in essence selecting the $\mathbf{x}_1$ that best matches $\mathbf{x}_2$. We update $E(\mathbf{x}_1) = E(\mathbf{x}_1^\star, \mathbf{x}_2)$. We also invalidate all $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ for $\mathbf{x}_1 \neq \mathbf{x}_1^\star$.

4 Results

We evaluate our proposed method in a number of experiments using the KITTI dataset [17]. For all results presented here, we build a training set from the first ten KITTI city sequences and a test set from the remaining sequences. We used $n_s = 31$, 20 EM iterations, and an occupancy grid with a resolution of $30\ \text{cm}$ that extends over a $50\ \text{m}$ by $50\ \text{m}$ area. We set our smoothing parameter $w_p = 0.07$ by inspecting a validation set constructed from the first ten KITTI city log sequences.

4.1 Visualizing the Learned Feature Space

We first present a qualitative evaluation of our learned feature space. We present two example scenes in Fig. 3 and Fig. 4. In each scene, we examine dynamic objects, including a car and two pedestrians. We consider a number of locations on these objects, as marked in Fig. 3a and Fig. 4a. We compute feature vectors for these locations and then measure the distance in feature space to locations in the scan at the following timestep.

We can see that our learned features are discriminative. Each chosen location is closest in feature space to its matching location in the following timestep. Unsurprisingly, we see that similar locations sometimes yield similar feature vectors, such as different corners of the car, between the two pedestrians, or sometimes background that has similar structure. Nevertheless, we see that the most similar feature vectors are clustered around the respective matching location.

4.2 Learned Feature Space vs. Occupancy Constancy

In additional to the qualitative results in §4.1, we also perform a quantitative analysis to demonstrate the performance of our learned feature space. We compare the our learned feature space with the occupancy constancy metric proposed by Ushani et al. [16] in terms of how discriminative they are in determining matching or non-matching locations.

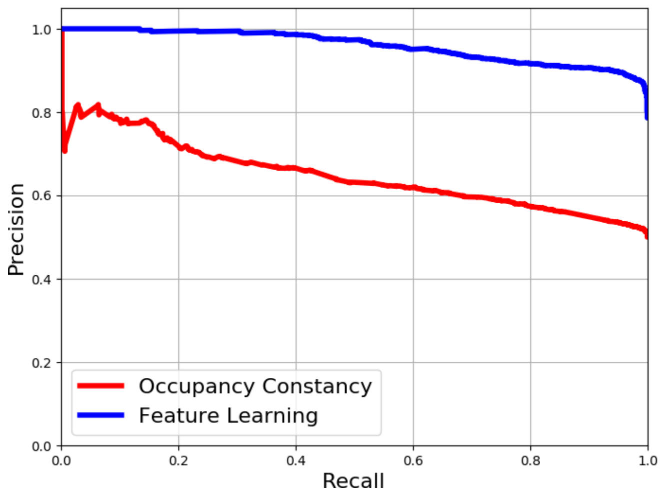

Figure 2: Precision-recall curves for classifiers based on occupancy constancy and feature learning. We find that feature learning results in a more discriminative classifier.

We build a simple binary classifier for each method that determines if two locations, $\mathbf{x}_1$ from G_t and $\mathbf{x}_2$ from G_{t+1}, are matching or non-matching. For our learned feature space, we take the distance in feature space, T_{distance}, and compare this against a threshold τ. If the distance is small, our classifier predicts that $\mathbf{x}_1$ and $\mathbf{x}_2$ are matching and otherwise predicts that they are non-matching. For occupancy constancy, we perform a similar procedure with the occupancy constancy metric T_{match} from [16].

We sample 5000 locations $\mathbf{x}_1$ from the foreground of some occupancy grid G_t from the test set. For each $\mathbf{x}_1$, we sample another location $\mathbf{x}_2$ from the following occupancy grid G_{t+1}. We evenly sample $\mathbf{x}_2$ such that is equally likely to match or not match $\mathbf{x}_1$.

We present precision recall curves for each described classifier in Fig. 2. As we can see, our learned feature space is significantly better at distinguishing between matching and non-matching locations than occupancy constancy, demonstrating that it is more discriminative for the task at hand.

4.3 Scene Flow Results

Finally, we evaluate the performance of the scene flow estimate. We perform this evaluation for cars, cyclist, and pedestrians, and also over all of the foreground classes. Results can be found in Table 1. We find a significant improvement in the error statistics of the scene flow estimate across all classes for our proposed feature learning approach over the occupancy constancy approach proposed by Ushani et al. [16].

One downside of our method however is increased runtime. While Ushani et al. [16] reports real-time performance (i.e., runtime of under 100 ms for 10 Hz data), our method takes about 188 ms on average using a machine with an Intel i7 CPU and an NVIDIA GeForce GTX 1080 GPU. We attribute this added runtime mainly to the time it takes to pass the input data through the network and evaluate pairwise distances in feature space. Nevertheless, with current trends in specialized hardware, we anticipate that this gap in runtime performance will be bridged in the near future.

	Car		**Cyclist**		**Pedestrian**		**All Foreground**	
	Mean Error	Within 30 cm	Mean Error	Within 30 cm	Mean Error	Within 30 cm	Mean Error	Within 30 cm
[16]	19.3 cm	83.8%	24.5 cm	78.8%	38.9 cm	74.3%	22.1 cm	81.4%
Proposed	15.6 cm	89.2%	15.9 cm	93.1%	22.9 cm	89.0%	16.4 cm	88.2%

Table 1: Error statistics for the scene flow estimated for various classes and over all foreground.

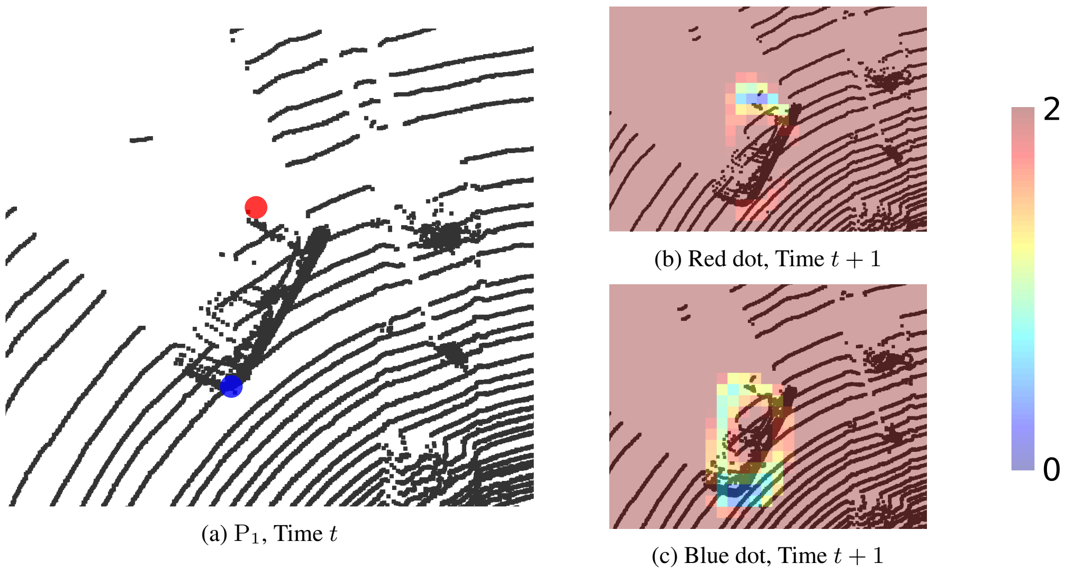

Figure 3: A visual look at the distances in the feature space for a scene with a turning car. For the two locations from P_1 (bird's eye view shown in Fig. 3a), we evaluate the distance in feature space, T_{distance}, to locations in the successive scan P_2, shown in Fig. 3b and Fig. 3c. The shown colormap ranges from a distance of 0 to 2 in the feature space.

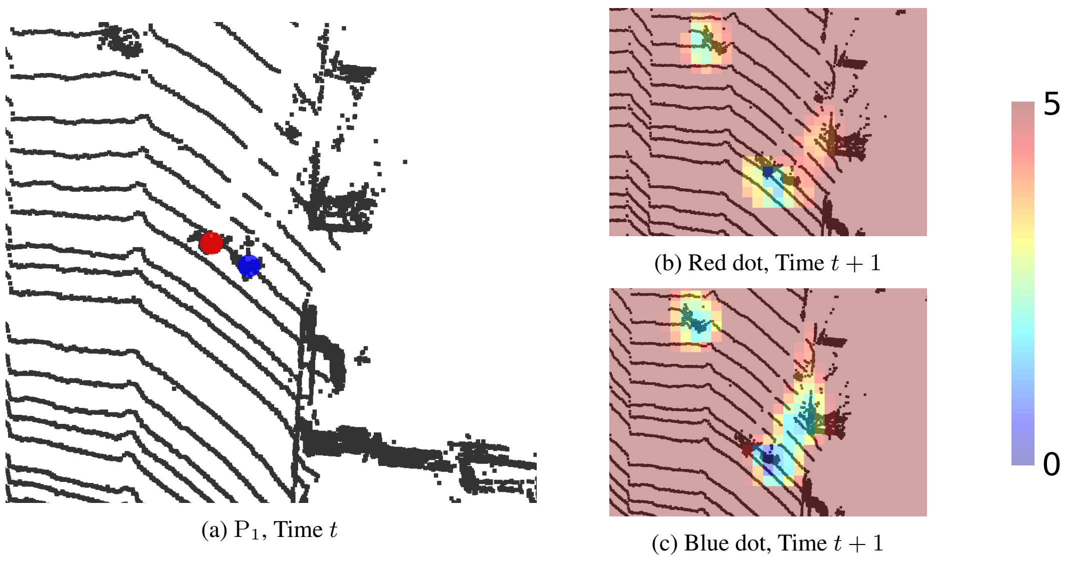

Figure 4: A visual look at the distances in the feature space for a scene with pedestrians walking along a sidewalk. For the two locations from P_1 (bird's eye view in Fig. 4a), we evaluate the distance in feature space, T_{distance}, to locations in the successive scan P_2, shown in Fig. 4b and Fig. 4c. The shown colormap ranges from a distance of 0 to 5 in the feature space.

5 Conclusion

In this work, we have presented a feature learning method for scene flow estimation from LIDAR data. We train an encoding network to extract features from an occupancy grid. This learned feature space is then leveraged in an energy minimization problem to solve for scene flow between successive scans. This approach yields improved results, both directly in the feature space itself and in the improved scene flow estimate, that beats the accuracy of the current state-of-the-art. We provide an implementation of our method at `https://github.com/aushani/flsf`.

Acknowledgments

The authors would like to thank Maani Ghaffari Jadidi for his helpful comments. Also, we would like to thank the reviewers for their helpful feedback. This work was supported by a grant from the Toyota Research Institute under award N021515.

References

[1] A. Petrovskaya and S. Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Auton. Robot.*, 26(2-3):123–139, 2009.

[2] 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*, pages 3800–3806, Kobe, Japan, 2009.

[3] A. Feldman, M. Hybinette, and T. Balch. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *J. Field Robot.*, 29(2):258–276, 2012.

[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*, pages 3075–3081, St. Paul, MN, USA, 2012.

[5] 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.*, pages 1–8, Berkeley, CA, USA, 2014.

[6] 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.*, pages 1137–1143, Hamburg, Germany, Sept. 2015.

[7] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.

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

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

[10] C. Vogel, K. Schindler, and S. Roth. Piecewise rigid scene flow. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 1377–1384, 2013.

[11] C. Vogel, K. Schindler, and S. Roth. 3d scene flow estimation with a piecewise rigid scene model. *Int. J. Comput. Vis.*, 115(1):1–28, 2015.

[12] 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*, pages 98–104, Chicago, IL, USA, 2015.

[13] T. Taniai, S. N. Sinha, and Y. Sato. Fast multi-frame stereo scene flow with motion segmentation. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, Honolulu, HI, USA, 2017.

[14] A. Behl, O. H. Jafari, S. K. Mustikovela, H. A. Alhaija, C. Rother, and A. Geiger. Bounding boxes, segmentations and object coordinates: How important is recognition for 3d scene flow estimation in autonomous driving scenarios? In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2574–2583, Honolulu, HI, USA, 2017.

[15] 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.

[16] 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 *Proc. IEEE Int. Conf. Robot. and Automation*, Singapore, May 2017.

[17] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res.*, 2013.

[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*, pages 6090–6095, Hong Kong, China, 2014.

[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.*, 12(4):1331–1342, 2011.

[20] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *Int. J. Robot. Res.*, 2017.

[21] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *Proc. European Conf. Comput. Vis.*, pages 499–515, Amsterdam, Netherlands, 2016.

[22] N. Carlevaris-Bianco and R. M. Eustice. Learning visual feature descriptors for dynamic lighting conditions. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pages 2769–2776, Chicago, IL, USA, September 2014.

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 1(2):4, 2017.

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. Advances Neural Inform. Process. Syst. Conf.*, pages 5105–5114, 2017.

[25] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Trans. on Pattern Anal. and Mach. Intell.*, 21(5):433–449, 1999.

[26] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3d registration. In *Proc. IEEE Int. Conf. Robot. and Automation*, pages 3212–3217, Kobe, Japan, 2009.

[27] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1802–1811, Honolulu, HI, USA, July 2017.

[28] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. Robot. and Automation*, volume 2, pages 116–121, St. Louis, MO, USA, 1985.

[29] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Auton. Robot.*, 34(3):189–206, 2013.

[30] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4 (1):25–30, 1965.

[31] 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. URL `http://tensorflow.org/`. Software available from tensorflow.org.