LaserFlow: Efficient and Probabilistic Object Detection and Motion Forecasting

Gregory P. Meyer, Jake Charland, Shreyash Pandey, Ankit Laddha, Carlos Vallespi-Gonzalez, and Carl K. Wellington

Uber Advanced Technologies Group
{gmeyer,jakec,shreyash,aladdha,cvallespi,cwellington}@uber.com

Abstract. In this work, we present LaserFlow, an efficient method for 3D object detection and motion forecasting from LiDAR. Unlike the previous work, our approach utilizes the native range view representation of the LiDAR, which enables our method to operate at the full range of the sensor in real-time without voxelization or compression of the data. We propose a new multi-sweep fusion architecture, which extracts and merges temporal features directly from the range images. Furthermore, we propose a novel technique for learning a probability distribution over future trajectories inspired by curriculum learning. We evaluate LaserFlow on two autonomous driving datasets and demonstrate competitive results when compared to the existing state-of-the-art methods.

Keywords: Range View, Multi-sweep Fusion, Autonomous Driving

1 Introduction

The detection and motion forecasting of objects represent critical capabilities for a self-driving system. Many algorithms proposed for forecasting object motion take a sequence of detections as their input, but more recently there has been a focus on end-to-end approaches that generate predictions of future motion directly from a temporal sequence of the sensor data [5, 6, 15, 29]. LiDAR range sensors are commonly used for this task, and Figure 1 shows an example sequence of the raw input LiDAR sweeps in their native range view (RV) format. Also shown in Figure 1 is the desired output, which includes object detections along with probability distributions over their predicted future motion in the bird’s eye view (BEV). Producing distributions that capture the uncertainty in predicted motion is essential for a self-driving system so that the downstream motion planner can respond appropriately for objects that have high uncertainty. Given that LiDAR sensors inherently produce a set of range measurements, but the desired output motion forecasts are in the BEV, an important design consideration for end-to-end approaches is how to best process RV measurements to produce BEV output.

Prior work in this area starts by transforming the raw range measurements from the sensor into a 3D point cloud in a global coordinate frame, and then this sequence of aligned 3D point clouds is used to predict the motion of actors in
Fig. 1: End-to-end learning of motion predictions from sensor data takes as input the native range view (RV) measurements produced by LiDAR sensors and generates output in the bird’s eye view (BEV) as needed by downstream components. Unlike prior work which starts by transforming the range data into 3D points and then processing the resulting sparse data in the BEV, we operate directly in the compact native RV of the sensor.

The 3D points are generally accumulated in a grid of cells or voxels to allow convolutions to be performed in the BEV which matches the desired output representation. An advantage of the BEV is that the size of rigid objects remains constant independent of range or motion. This consistency adds a strong prior and simplifies the problem of learning to combine multiple views of an object over time.

However, BEV approaches have limitations in runtime efficiency, which is important for a real-time system that needs to react quickly to a changing environment. As shown in Figure 1b, the projected 3D points are sparse, and performing feature extraction in the BEV can require defining a limited region of interest (RoI) for processing [5,6,15,29]. Furthermore, BEV approaches that process the 3D points into cells have a trade-off between performance and runtime that is dependent on the cell size [12].

Alternatively, extracting features in the RV, where the data is naturally dense, supports efficiently operating out to the maximum range of the sensor in all directions, which is important for intersections that require looking to the side of the vehicle. The sensor data in the native RV also captures important context about which parts of the scene are visible by the sensor and which parts are occluded by other objects. This occlusion information is lost when the data is transformed into a 3D point cloud in the BEV. Finally, operating on the raw measurements of the sensor removes the complexity and trade-offs of artificial abstractions like cells and voxels, resulting in an approach that can operate efficiently at all ranges while still performing well on smaller objects [17,18].

This work extends the RV based probabilistic object detection introduced in [18] to the problem of motion forecasting. A key challenge to using the RV for
motion forecasting is how to combine a sequence of multiple LiDAR sweeps that were captured at different sensor locations due to vehicle ego-motion. Projecting this temporal data into a common RV frame to extract multi-sweep features can result in significant distortions due to self-occlusions and perspective changes, and our proposed method is motivated by this challenge (see Section 3.1 and Figure 2 for more details).

In this paper, we propose a highly efficient method for probabilistic object detection and motion forecasting using LiDAR. We achieve comparable performance to state-of-the-art methods on the task of motion forecasting with an efficient runtime that supports operating over the full range of the sensor. This is achieved by extracting features on the raw range view measurements produced natively by the LiDAR sensor and then using a novel method of fusing multiple sweeps. Finally, using an approach inspired by curriculum learning, we present a novel approach for learning a distribution over future trajectories that initially emphasizes earlier predictions before learning longer horizons.

2 Related Work

2.1 3D Object Detection from LiDAR

Over the past few years, a wealth of approaches \[8,12,17,18,23,26,27,28,31,32\] have been proposed to solve 3D object detection using LiDAR. The methods typically vary in terms of the representation of the LiDAR data. VoxelNet \[32\] and SECOND \[26\] represent the world as a 3D grid and employ 3D convolutions. PIXOR \[27\] and PointPillars \[12\] demonstrated that a 2D grid and 2D convolutions could be used as an alternative to high-cost 3D convolutions. However, representing the world as a 2D or 3D grid is still inefficient due to the sparsity of the LiDAR, especially at long range, resulting in a significant amount of empty cells. Furthermore, due to voxelization, these representations lose fine-grain geometrical details, which make the detection of small objects difficult. As a result, PointRCNN \[23\] and STD \[28\] proposed to use a point cloud representation. However, by treating the LiDAR data as an unordered set of points, these methods lose information about how the scene was captured, which makes it challenging to reason about occlusions. Additionally, these methods are often slower due to the unstructured nature of the representation. LaserNet \[18\] proposed to use a spherical representation, referred to as the range view, where each pixel represents a LiDAR point. The range view is the native representation of the LiDAR; therefore, it is compact and retains the fine-grain details. In this work, we extend LaserNet \[18\] to predict the motion of objects.

2.2 Motion Forecasting from Detections

Motion forecasting requires a history of observations. Several previous methods \[1,2,7,9,10,11,13,21,22,25,30\] have used a sequence of past detections to predict future motion. These methods can be broadly divided into two groups:
approaches that use a feed-forward convolutional neural network (CNN) and methods that encode historical information using a recurrent neural network (RNN). Methods that leverage RNNs often model the motion of each object independently and employ separate models for different interactions, such as actor to actor, actor to scene, or both. CNN based methods capture both object motion and interactions in a sequence of images and use a feed-forward network to predict the future. For each actor, they render a set of images containing the position of the actor and other nearby objects across time. Afterwards, these images are passed to a CNN which forecasts the motion for the object of interest. Recently, have been proposed which predict trajectories for all the actors together. These approaches typically require some amount of computation to be run for every actor in the scene, making it difficult to scale to dense urban environments. Furthermore, all of these methods disregard the rich information provided by the sensor.

2.3 Motion Forecasting from LiDAR

To our knowledge, the first method to attempt joint 3D object detection and motion forecasting from LiDAR was FaF. Their approach used a 3D grid to represent a sequence of LiDAR point clouds and predicted a trajectory one second in length for all vehicles within a region of interest. IntentNet extended FaF to predict the intent of each actor, e.g. keep straight, turn right, merge left, etc., and increased the length of the trajectory to 3 seconds. Their method was further improved by NMP which added the motion planner of the self-driving vehicle as an additional source of supervision to the model. Most recently, SpAGNN introduced a graph-based interaction model and uncertainty estimates to IntentNet. All of the previous work that performs joint object detection and motion forecasting uses a 3D or bird’s eye view representation of the LiDAR. In contrast, we propose to use the native range view representation of the LiDAR.

3 Proposed Method

In the following sections, we will describe our approach to multi-sweep fusion, our proposed method to predicting a probability distribution of trajectories, and our end-to-end training procedure.

3.1 Multi-Sweep Fusion

As the LiDAR spins, it produces an image using a set of range sensing lasers. For each measurement, the sensor returns the range \( r \) and reflectance \( e \) of the observed surface. In addition, it provides the azimuth angle of the sensor \( \theta \) and elevation angle of the corresponding laser \( \varphi \) from which the 3D position \( x \) of the surface can be calculated. A complete revolution of the LiDAR is colloquially...
Fig. 2: An illustration of the bird’s eye view and range view representations for two sweeps captured from different viewpoints. Note that the measurements generated by the LiDAR depend on its position relative to the observed surface. Fusing multiple sweeps in the bird’s eye view amounts to translating and rotating the LiDAR points from the previous sweeps into the current sweep’s coordinate frame. On the other hand, the fusion of multiple sweeps in a range view representation requires the LiDAR points to be projected onto a shared spherical image. Changing the center of projection, as shown above, can introduce a significant amount of distortion into the range image due to partially observed surfaces and self-occlusions. Notice the blue object has sparser measurements when compared to the red object in the shared view. However, these distortions do not exist in the original range image, which motivates our proposed method.

referred to as a sweep. Multiple sweeps of the LiDAR can be utilized to capture the motion of objects across time.

Unlike the previous work \cite{5,6,15,29}, our proposed method uses the native range view (RV) representation of the LiDAR data instead of the artificial bird’s eye view (BEV). Using the RV has advantages in terms of efficiency and small object detection \cite{17,18}; however, the fusion of multiple LiDAR sweeps becomes non-trivial. In order to fuse measurements from multiple sweeps, the 3D points are transformed into a global coordinate frame that accounts for the ego-motion of the self-driving vehicle. Afterwards, the points can be projected into a shared image frame where multi-sweep features are extracted using a convolutional neural network (CNN). With the BEV, an orthogonal projection is used; therefore, the difference between the original and shared image frames amounts to a 2D rotation and translation. For the RV, a spherical projection is used; therefore, transforming from the original to a shared image frame may cause the center of projection to be moved. As a result, a significant amount of distortion can be introduced into the image, including shadows or holes from unobserved or partially observed surfaces and data loss due to self-occlusions. Figure 2 illustrates the challenge of utilizing the RV representation.

To overcome these issues, we propose a novel multi-sweep fusion architecture. With the proposed architecture, features are extracted independently from
Fig. 3: An overview of our multi-sweep fusion architecture. For each sweep, input images are represented using the native range view representation of the LiDAR. The images are then passed to a convolution network to extract features. Afterwards, the features extracted from the past sweeps are concatenated with the ego-motion features, which capture the motion of the self-driving vehicle from the past to the current sweep. The resulting features are passed to another network that transforms the features into the coordinate frame of the current sweep. Finally, the past features are warped into the current image frame and concatenated with the current sweep’s features.

**Input Features** The input to our multi-sweep architecture is a set of images, \( \{I_S, I_{S+1}, \ldots, I_0\} \), corresponding to a sequence of sweeps. The images are constructed by mapping LiDAR measurements, also referred to as points, to a column based on the azimuth angle of the sensor at the time the measurement was captured and to a row based on the elevation angle of the laser that produced the measurement. For each pixel in the image, we generate a set of channels corresponding to the measurement’s range, reflectance, and a flag indicating whether the measurement is valid. Furthermore, if a high-definition map is available, we provide the LiDAR point’s height above the ground and a flag indicating whether the point is on or above a road surface. Figure 1 depicts the range channel for a set of input sweeps.
**Ego-Motion Features** Since the self-driving vehicle moves over time, we provide the ego-motion as a feature. Let $P_s \in \mathbb{SE}(3)$ represent the pose of the LiDAR at the start of the $s$-th sweep. The motion of the ego-vehicle from sweep $s$ to the current sweep can be computed as

$$\Delta_s = P_0 P_s^{-1} [0, 0, 0, 1]^T.$$  

(1)

Our proposed method utilizes a fully convolutional neural network; therefore, features should not require specific knowledge about its position within the image. For each pixel, we rotate the ego-motion features based on the measurement’s azimuth angle $\theta$,

$$\delta = R_\theta^T [\Delta_s^x, \Delta_s^y]^T$$  

(2)

where $\Delta_s^x$ and $\Delta_s^y$ are the $x$ and $y$ component of $\Delta_s$, respectively. The importance of rotating the ego-motion feature is shown in our ablation study.

**Feature Warping** To transform the features between sweeps, we need to define a mapping between all pixels in one image to another. We refer to this process as feature warping. Each valid pixel corresponds to a LiDAR point $x$, and each image has a corresponding pose $P_s$. To map from a previous sweep to the current sweep, we first transform the point into the current sweep’s coordinate frame,

$$x' = P_0 P_s^{-1} x.$$  

(3)

Afterwards, we determine the position of the transformed point in the current sweep’s image frame,

$$\theta' = \text{atan2} (y', x') \quad \text{and} \quad \varphi' = \arcsin \frac{z'}{r'}$$  

(4)

where $x' = [x', y', z']^T$ and $r' = \|x'\|$. By repeating this process for each LiDAR point in the previous sweeps, we obtain a mapping between all the previous images, $\{I_s, I_{s+1}, \ldots, I_{-1}\}$, and the current image, $I_0$. If multiple points from a single sweep map to the same position, we keep the point with the smallest $r'$. This mapping is used to warp the learned features into the shared image frame defined by the current sweep. Note that the warping still introduces distortion, but the features themselves are extracted from the undistorted images.

**Multi-Sweep Architecture** As shown in Figure 3, the images corresponding to the individual sweeps are passed into a CNN where the parameters are shared across all sweeps. By sharing the weights, the network is forced to extract the same set of features from each sweep. An additional benefit is that the features for a particular sweep do not change, so they can be cached and reused in subsequent runs of the network. Afterwards, the extracted features from the previous sweeps are concatenated with the corresponding ego-motion features and passed into another CNN, which we refer to as the transformer network. Again the weights are shared across the sweeps, and the purpose of this network...
is to undo the effect of the ego-motion on the extracted features. Ideally, after this network, the only remaining difference between the sweep features will be due to the motion of the objects and not the motion of the self-driving vehicle. The resulting feature maps are warped as described above and concatenated with the current sweep’s feature map and passed to a U-Net style [20] backbone network. For a detailed description of the entire network, refer to Appendix A.

3.2 Model Predictions

Our goal is to predict the probability of all object trajectories within the scene, \( p(\mathcal{T}) \). We assume all the objects and time-steps are independent; as a result, the probability factorizes into the following:

\[
p(\mathcal{T}) = \prod_{t=0}^{T} \prod_{i=1}^{N} p(\mathbf{\tau}_t^i)
\]

where \( N \) is the number of detected objects, \( T \) is the number of time-steps into the future for which we predict the motion of the objects, and \( \mathbf{\tau}_t^i \) is the position of the \( i \)-th object at the \( t \)-th time-step represented by the corners of its bounding box. Additionally, we assume the spatial dimensions of the bounding box are independent and drawn from Laplace distributions; therefore, the probability of the trajectories becomes:

\[
p(\mathcal{T}) = \prod_{t=0}^{T} \prod_{i=1}^{N} \prod_{j=1}^{D} p(\tau_{ij}^t; \nu_{ij}^t, b_{ij}^t) = \prod_{t=0}^{T} \prod_{i=1}^{N} \prod_{j=1}^{D} \frac{1}{2b_{ij}^t} \exp \left( -\frac{|\tau_{ij}^t - \nu_{ij}^t|}{b_{ij}^t} \right)
\]

where \( D \) is the dimensionality of the bounding box, and \( \nu_{ij}^t \in \mathbb{R} \) and \( b_{ij}^t \in \mathbb{R}_+ \) are the mean and scale of the Laplace distribution corresponding to the \( j \)-th dimension of the \( i \)-th object’s bounding box at time \( t \). We assume a Laplace distribution because the previous work [18,19] has shown that it closely matches the empirical distribution of the bounding box corners for the detection task. To estimate the probability \( p(\mathcal{T}) \) given a sequence of LiDAR sweeps, our network predicts a set of means \( \mathbf{V} = \{ \nu_{00}^0, \nu_{01}^0, ..., \nu_N^0 \} \) and scales \( \mathbf{B} = \{ b_{00}^0, b_{01}^0, ..., b_{N}^0 \} \), where \( \nu_i^t = [\nu_{i0}^t, \nu_{i1}^t, ..., \nu_{iD}^t] \in \mathbb{R}^D \) and \( b_i^t = [b_{i0}^t, b_{i1}^t, ..., b_{iD}^t] \in \mathbb{R}_+^D \).

**Predicting Trajectories** In order to predict the trajectories of all the objects, we first need to identify the LiDAR points that lie on objects. Therefore, we begin by predicting a set of class probabilities for each point in the range image at \( t = 0 \). Assuming a LiDAR point is on an object, the network predicts a probability distribution over bounding box trajectories. Following [18], a bounding box is represented by its four corners in the BEV, i.e. \( D = 8 \), and all predictions are made relative to LiDAR points. For each point, the network outputs the dimensions of the object’s bounding box \((l, w)\), which we assume is constant across time. Also, the network outputs a set of displacement vectors \( \{(d_{xt}^0, d_{yt}^0)\}_{t=0}^{T} \), a set
of rotation angles \( \{(\omega_x^t, \omega_y^t) := (\cos 2\omega t, \sin 2\omega t)\}_{t=0}^T \), and a set of uncertainties \( \{(s_x^t, s_y^t) := (\log b_x^t, \log b_y^t)\}_{t=0}^T \) for all of the time-steps. At \( t = 0 \), the center and orientation of the bounding box is computed as follows:

\[
c_0 = [x, y]^T + R_\theta [d_x^0, d_y^0]^T \quad \text{and} \quad \phi_0 = \theta + \frac{1}{2} \tan 2(\omega_y^0, \omega_x^0)
\]

where \((x, y)\) is the position of the LiDAR point in the BEV, \(\theta\) is the azimuth angle of the LiDAR point, and \(R_\theta\) is the rotation matrix parameterized by \(\theta\). Notice that the network predicts an orientation between \(-90^\circ\) and \(90^\circ\), since the bounding box is symmetrical. Furthermore, when \(t \geq 1\),

\[
c_t = c_{t-1} + R_\theta [d_x^t, d_y^t]^T \quad \text{and} \quad \phi_t = \phi_{t-1} + \frac{1}{2} \tan 2(\omega_y^t, \omega_x^t). \tag{8}
\]

The corners of the bounding box at each time-step are calculated by

\[
v_1^t = c_t + \frac{1}{2} R_{\phi_t} [l, w]^T, \quad v_2^t = c_t + \frac{1}{2} R_{\phi_t} [-l, -w]^T, \quad v_3^t = c_t + \frac{1}{2} R_{\phi_t} [l, -w]^T, \quad v_4^t = c_t + \frac{1}{2} R_{\phi_t} [-l, w]^T \tag{9},
\]

and \(v_t = [v_1^t, v_2^t, \ldots, v_3^t, v_4^t] \in \mathbb{R}^8\). The along track uncertainty, \(b_x^t = \exp s_x^t\), and the cross track uncertainty, \(b_y^t = \exp s_y^t\), represent the uncertainties at time \(t\) in the corners along the direction of motion and perpendicular to the motion, respectively. At any particular time-step \(t\), the scale of the distribution is assumed to be constant across all four corners, \(b_t = [b_x^t, b_y^t, \ldots, b_x^t, b_y^t] \in \mathbb{R}_+^8\).

In [18], the model is trained to predict a multimodal probability distribution over bounding boxes, which improves the recall of the detector. However, to reduce the complexity and to simplify the comparisons with the previous work, we predict a unimodal distribution over bounding box trajectories.

**Post-Processing Predictions** To cluster predictions from individual LiDAR points, we use the approximate mean shift algorithm proposed in [18]. Unlike the previous work, we do not perform the weighted average over the corners to combine the predictions within a cluster because it is not guaranteed to produce a rectangular bounding box. Instead, we use a simple average over the individual bounding box parameters at each time-step, i.e. the mean of each time-step’s center, dimensions, and orientation. For an in-depth analysis of this issue, refer to Appendix B. Following mean shift clustering, the output is a set of means \(\mathbf{V}\) and scales \(\mathcal{B}\) for all of the detected objects across all time-steps.

### 3.3 End-to-end Training

In order to train the network to predict the parameters of the probability distribution of trajectories, \(\mathbf{V}\) and \(\mathcal{B}\), we employ the Kullback-Leibler (KL) divergence to penalize the difference between the predicted probability distribution and the
ground-truth distribution. Given our independence assumptions, minimizing the KL divergence between the ground-truth distribution of the trajectories and the prediction distribution becomes

$$\arg \min_{\mathbf{V}, \mathbf{B}} D_{KL}(p(T; \mathbf{V}, \mathbf{B}) \| q(T; \mathbf{V}, \mathbf{B}))$$

$$= \arg \min \sum_{t=0}^{T} \sum_{i=1}^{N} \sum_{j=1}^{D} D_{KL}(p(\tau_{ij}^t; \tilde{\nu}_{ij}^t, \tilde{b}_{ij}^t) \| q(\tau_{ij}^t; \nu_{ij}^t, b_{ij}^t)).$$

(10)

A proof of this equality is provided in Appendix C. In the following sections, we will explain our loss functions in detail, and describe how we obtain the parameters of the ground-truth distribution, $\mathbf{V}$ and $\mathbf{B}$.

**Loss Functions** Our loss functions consist of the focal loss [14] used to learn the class probabilities for each LiDAR point and the KL divergence [19] used to learn the distribution of trajectories. The classification loss is defined as

$$\mathcal{L}_{cls} = \frac{1}{HW} \sum_{i=1}^{C} \sum_{j=1}^{C} -[\tilde{c}_i = j](1 - p_{ij})^\gamma \log p_{ij}$$

(11)

where $p_{ij}$ is the probability of the $j$-th class at the $i$-th pixel, $\tilde{c}_i$ is the ground-truth class of the $i$-th pixel, $[\cdot]$ is an indicator function, $\gamma$ is the focusing parameter [14], $C$ is the number of object classes plus a background class, and $W$ and $H$ is the width and height of the range image, respectively. The regression loss is defined as

$$\mathcal{L}_{reg} = \frac{1}{TND} \sum_{t=0}^{T} \sum_{i=1}^{N} \sum_{j=1}^{D} D_{KL}(p(\tau_{ij}^t; \tilde{\nu}_{ij}^t, \tilde{b}_{ij}^t) \| q(\tau_{ij}^t; \nu_{ij}^t, b_{ij}^t))$$

(12)

where $\nu_{ij}^t$ is the $j$-th dimension of the $i$-th predicted bounding box at the $t$-th time-step, $b_{ij}^t$ is the corresponding predicted uncertainty, and $\tilde{\nu}_{ij}^t$ and $\tilde{b}_{ij}^t$ define the ground-truth distribution. The probability distributions $p(\tau_{ij}^t; \tilde{\nu}_{ij}^t, \tilde{b}_{ij}^t)$ and $q(\tau_{ij}^t; \nu_{ij}^t, b_{ij}^t)$ are assumed to be Laplace distributions; therefore, the KL divergence becomes:

$$D_{KL}(p(x; \tilde{\mu}, \tilde{b}) \| q(x; \mu, b)) = \log \frac{b}{\tilde{b}} + \frac{\tilde{b}}{b} \exp \left( -\frac{|\mu - \tilde{\mu}|}{b} \right) + \frac{|\mu - \tilde{\mu}|}{b} - 1.$$  

(13)

Refer to [16] for a derivation and [19] for an analysis of the loss function. In order to properly learn the along and cross track uncertainties, the predicted and ground-truth bounding boxes need to be transformed into the appropriate coordinate frame. Before applying the regression loss, the corners of each bounding box are rotated such that the $x$-axis is aligned to the object’s direction of motion and the $y$-axis is perpendicular to the object’s motion. This is accomplished by rotating the predicted and ground-truth corners corresponding to the
i-th object at time $t$ by $R_{\phi^i_t}$, where $\phi^i_t$ is the predicted future orientation of the object. The total loss utilized by our proposed method is $L_{\text{total}} = L_{\text{cls}} + \lambda L_{\text{reg}}$ where $\lambda$ is used to weight the relative importance of the individual losses. For all experiments, $\gamma = 2$ and $\lambda = 4$, which we empirically found to perform well.

**Uncertainty Curriculum** To utilize the KL divergence, the parameters of the ground-truth distribution need to be specified for all objects and time-steps. The means of the distribution $\bar{\mathbf{V}}$ are simply set to the ground-truth bounding box provided by an annotator, but how to set the scales of the distribution $\bar{\mathbf{B}}$ remains an open question. In [19], heuristics for approximating the uncertainty of the ground-truth were explored for object detection. In this work, we propose a different approach inspired by curriculum learning [3].

Our network is trained such that the early predictions affect the future predictions as expressed in Eq. (8). As a result, early predictions need to be reliable to accurately predict future time-steps. Based on this observation, at the start of training, we set the uncertainty of the ground-truth for later time-steps to be high while the uncertainty of the early time-steps are set to be low,

$$\tilde{b}_t = \alpha b^t_{\text{max}} + (1 - \alpha) b^t_{\text{min}}$$

where $\alpha \in [0, 1]$, $b^{t}_{\text{max}} = \frac{t}{T} \eta + \epsilon$, $b^{t}_{\text{min}} = \epsilon$, and $\tilde{b}_t$ is shared across all $N$ objects and all $D$ dimensions at time $t$. As training progresses, the uncertainty of all time-steps is exponentially reduced, $\alpha = \exp(-\beta k)$, where $\beta$ controls the rate of decay and $k$ is the training iteration. As shown in [19], increasing the ground-truth uncertainty of a particular example reduces the loss for that example as long as the predicted uncertainty does not under-estimate the ground-truth uncertainty. Therefore, by setting the uncertainty of the earlier time-steps to be lower than the later time-steps, we are placing a larger emphasis on learning the earlier time-steps. For all of our experiments, $\eta = 100 \text{ cm}$, $\epsilon = 5 \text{ cm}$, and $\beta$ is set such that $\alpha \approx 0$ half-way through training. We experimentally found that the performance of the model is not sensitive to the exact values of the hyper-parameters.

A sensible alternative could be to just re-weight the loss for various time-steps and use a constant ground-truth uncertainty. To show why this is a less attractive option, let us inspect the derivative of the KL divergence with respect to the predicted mean,

$$\frac{\partial D_{\text{KL}}}{\partial \mu} = \text{sgn}(\mu - \tilde{\mu}) \left(1 - \exp\left(-\frac{|\mu - \tilde{\mu}|}{b}\right)\right).$$

Re-weighting the KL divergence simply re-scales the derivative. However, by increasing the ground-truth uncertainty, the magnitude of the derivative is only reduced once the ratio between $|\mu - \tilde{\mu}|$ and $b$ becomes small, since the exponential term in Eq. (15) will go to one. Therefore, at the start of training, the network will be pushed to correct significant errors while ignoring small errors. By employing a curriculum on the uncertainty, we can ensure the predictions for the early
time-steps will be refined before the later time-steps. In the next section, we demonstrate through experimentation that utilizing an uncertainty curriculum is superior to re-weighting the loss.

4 Experiments

Our proposed method is evaluated and compared to state-of-the-art methods on two autonomous driving datasets: ATG4D [18] and NuScenes [4]. The ATG4D dataset contains $1.2 \times 10^6$ sweeps for training where the NuScenes dataset contains $2.7 \times 10^5$ training sweeps. ATG4D uses a 64-beam LiDAR resulting in a $2048 \times 64$ range image, and NuScenes uses a 32-beam LiDAR producing a range image with a resolution of $1024 \times 32$. For both datasets, we utilize a total of 5 input sweeps, and predict 3 seconds into the future at 0.5 seconds intervals. To train the network, we use the same optimizer and learning scheduling as [18].

Following [5], we measure object detection and motion forecasting performance for vehicles within a region of interest (RoI). For ATG4D, the RoI is $140 \times 80$ meters centered on the self-driving vehicle, and for NuScenes, the RoI is $100 \times 100$ meters. Detection performance is evaluated using the average precision (AP) metric where an intersection-over-union (IoU) of 0.7 with a ground-truth label is required for a detection to be considered a true positive. Furthermore, to evaluate the precision of the motion forecasting, we use the $L_2$ error between the ground-truth’s center and the prediction’s center at various time-steps. Since the $L_2$ error is affected by the sensitivity of the detector, [5] uses a fixed recall to evaluate the models. For ATG4D, the recall point is 80% at a 0.5 IoU, and for NuScenes, the recall point is 60% at a 0.5 IoU.

4.1 Comparisons with the State-of-the-Art

The results for our proposed approach and existing state-of-the-art methods on ATG4D are shown in Table 1. On this dataset, our method obtains better results compared to the existing methods in terms of AP and $L_2$ error at 0 and 1 seconds. LaserFlow as well as FaF [15], IntentNet [6], and NMP [29] predict trajectories in a single stage; however, SpAGNN [5] uses multiple stages to refine their predictions. When compared to their single stage version, our method outperforms SpAGNN by 8 cm at 3 seconds. The results on NuScenes are shown in Table 2. On this dataset, our method out-performs SpAGNN at 1 second, but has worse performance at 0 and 3 seconds. Currently, no single stage methods are available for comparisons on NuScenes. As future work, the graph neural network from SpAGNN can be incorporated into LaserFlow to refine its predictions.

4.2 Ablation Study

On ATG4D, we perform an ablation study on our multi-sweep fusion architecture, and the results are listed in Table 3. As shown in the table, naively applying the early fusion method employed by BEV methods to the RV, i.e. rendering
the previous sweeps in the current’s sweep coordinate frame, performs poorly at longer time horizons likely due to the distortion discussed in Section 3.1. Leveraging our proposed fusion method without the transformer network also under-performs at later time-steps showing the benefit of learning this transformation. Lastly, keeping the ego-motion feature in the global coordinate frame, by removing the rotation from Eq. 2, has a substantial impact on performance.

In addition, we conduct an ablation on our training procedure, and the results can be seen in Table 4. We observe that predicting only the mean of the distribution, i.e. removing the uncertainty estimate, significantly degrades the performance in terms of both object detection and motion forecasting. When predicting the full distribution, we see a reduction in performance at 3 seconds when the uncertainty curriculum is not utilized ($\alpha = 0$ in Eq. 14). Furthermore, applying a curriculum on the weight instead of the uncertainty performs worse at 3 seconds, which demonstrates the value of our proposed uncertainty curriculum.

Experiments on multi-class object detection and motion forecasting are available in Appendix D. For additional qualitative results, refer to Appendix E.

4.3 Uncertainty Evaluation

The calibration of the predicted distributions on ATG4D is shown in Figure 4. An explanation on how the calibration plots are generated is available in [19]. From the figure, we observe that our proposed method can accurately learn the along and cross track distribution of the bounding box corners for all time-steps.

4.4 Runtime Performance

The total runtime of our proposed method on ATG4D is 60 ms. The only existing method to report timing is FaF [15], and they claim a runtime of 30 ms. However, their BEV method operates only on the 144 $\times$ 80 meter RoI. Our RV method operates at the full range of the LiDAR which is a circle with diameter of 240
Fig. 4: Calibration plots showing the reliability of the predicted distributions at different time-steps. A well-calibrated distribution will follow the dashed line.

meters, and its runtime is independent of the range of the sensor. Therefore, our method runs on an area approximately 4x the size in only 2x the time. For the purposes of autonomous driving, it is critical to consider the entire scene without disregarding any measurements provided by the sensor.

5 Conclusion

We proposed a novel method for 3D object detection and motion forecasting from the native range view representation of the LiDAR. Our approach is both efficient and competitive with existing state-of-the-art methods that rely on the bird’s eye view representation. Although we focused on demonstrating the effectiveness of the range view at these tasks, we believe our multi-sweep fusion and uncertainty curriculum could be successfully incorporated into a bird’s eye view or a hybrid view system to exploit the advantages of both representations.
References

1. Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., Savarese, S.: Social LSTM: Human trajectory prediction in crowded spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
2. Bansal, M., Krizhevsky, A., Ogale, A.: ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst. arXiv preprint arXiv:1812.03079 (2018)
3. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the International Conference on Machine Learning (ICML) (2009)
4. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Lioing, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: NuScenes: A multimodal dataset for autonomous driving. arXiv preprint arXiv:1903.11027 (2019)
5. Casas, S., Gulino, C., Liao, R., Urtasun, R.: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. arXiv preprint arXiv:1910.08233 (2019)
6. Casas, S., Luo, W., Urtasun, R.: IntentNet: Learning to predict intention from raw sensor data. In: Proceedings of the Conference on Robot Learning (CoRL) (2018)
7. Chai, Y., Sapp, B., Bansal, M., Anguelov, D.: MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. arXiv preprint arXiv:1910.05449 (2019)
8. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3D object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
9. Deo, N., Trivedi, M.M.: Convolutional social pooling for vehicle trajectory prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2018)
10. Djuric, N., Radosavljevic, V., Cui, H., Nguyen, T., Chou, F.C., Lin, T.H., Schneider, J.: Motion prediction of traffic actors for autonomous driving using deep convolutional networks. arXiv preprint arXiv:1808.05819 (2018)
11. Hong, J., Sapp, B., Philbin, J.: Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
12. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: PointPillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2010)
13. Lee, N., Choi, W., Vernaza, P., Choy, C.B., Torr, P.H., Chandraker, M.: DESIRE: Distant future prediction in dynamic scenes with interacting agents. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
14. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2017)
15. Luo, W., Yang, B., Urtasun, R.: Fast and furious: Real time end-to-end 3D detection, tracking and motion forecasting with a single convolutional net. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
16. Meyer, G.P.: An alternative probabilistic interpretation of the Huber loss. arXiv preprint arXiv:1911.02088 (2019)
17. Meyer, G.P., Charland, J., Hegde, D., Laddha, A., Vallespi-Gonzalez, C.: Sensor fusion for joint 3D object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2019)

18. Meyer, G.P., Laddha, A., Kee, E., Vallespi-Gonzalez, C., Wellington, C.K.: LaserNet: An efficient probabilistic 3D object detector for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

19. Meyer, G.P., Thakurdesai, N.: Learning an uncertainty-aware object detector for autonomous driving. arXiv preprint arXiv:1910.11375 (2019)

20. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI) (2015)

21. Sadeghian, A., Kosaraju, V., Sadeghian, A., Hirose, N., Rezatofighi, H., Savarese, S.: SoPhie: An attentive GAN for predicting paths compliant to social and physical constraints. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

22. Sadeghian, A., Legros, F., Voisin, M., Vesel, R., Alahi, A., Savarese, S.: CAR-Net: Clairvoyant attentive recurrent network. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018)

23. Shi, S., Wang, X., Li, H.: PointRCNN: 3D object proposal generation and detection from point cloud. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

24. Tang, C., Salakhutdinov, R.R.: Multiple futures prediction. In: Proceedings of Advances in Neural Information Processing Systems (NIPS) (2019)

25. Vemula, A., Muelling, K., Oh, J.: Social attention: Modeling attention in human crowds. In: Proceedings of the IEEE international Conference on Robotics and Automation (ICRA) (2018)

26. Yan, Y., Mao, Y., Li, B.: SECOND: Sparsely embedded convolutional detection. Sensors (2018)

27. Yang, B., Luo, W., Urtasun, R.: PIXOR: Real-time 3D object detection from point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)

28. Yang, Z., Sun, Y., Liu, S., Shen, X., Jia, J.: STD: Sparse-to-dense 3D object detector for point cloud. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2019)

29. Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., Urtasun, R.: End-to-end interpretable neural motion planner. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

30. Zhao, T., Xu, Y., Monfort, M., Choi, W., Baker, C., Zhao, Y., Wang, Y., Wu, Y.N.: Multi-agent tensor fusion for contextual trajectory prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)

31. Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., Vasudevan, V.: End-to-end multi-view fusion for 3D object detection in LiDAR point clouds. In: Proceedings of the Conference on Robot Learning (CoRL) (2019)

32. Zhou, Y., Tuzel, O.: VoxelNet: End-to-end learning for point cloud based 3D object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
Appendix

A Network Architecture

The entire network architecture used by our proposed method to extract multi-sweep features and predict the motion of objects is depicted in Figure 5. The network is fully convolutional, and its input is the range view representation of the LiDAR. Since the height of the range image is significantly smaller than the width, downsampling and upsampling is only performed on the columns of the image, i.e., the number of rows in the image is constant throughout the network. The number of kernels used for each convolution is 16 in the multi-sweep fusion network and 64 in the backbone network. To allow the backbone network to compare features across sweeps, feature normalization which normalizes each channel independently, is not used in the multi-sweep fusion network. For more information on our approach to multi-sweep fusion, refer to Section 3.1.
B Convex Combination of Bounding Boxes

A bounding box is an oriented rectangle, which can be defined by four vertices, \( \{ v_1, v_2, v_3, v_4 \} \). A bounding box has the following properties:

\[
(v_1 - v_2)^T (v_3 - v_2) = 0, \tag{16}
\]

\[
l = \|v_1 - v_2\| > 0, \tag{17}
\]

\[
w = \|v_3 - v_2\| > 0, \tag{18}
\]

and

\[
v_4 = (v_1 - v_2) + v_3. \tag{19}
\]

The first property, Eq. (16), implies that the angle between \((v_1 - v_2)\) and \((v_3 - v_2)\) is \( \theta = \pm \frac{\pi}{2} \). Let us assume there is another bounding box defined by \( \{ \hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}_4 \} \), and our goal is to produce a new set of vertices through a convex combination of the two boxes:

\[
\hat{v}_i = \alpha v_i + (1 - \alpha) \hat{v}_i \tag{20}
\]

where \( \alpha \in [0, 1] \). For \( \{ \hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}_4 \} \) to be a valid bounding box it must satisfy the above properties (Eq. (16)-(19)). Under what circumstances will the new set of vertices satisfy all four properties? Let us start by considering the first property, Eq. (16).

\[
(\hat{v}_1 - \hat{v}_2)^T (\hat{v}_3 - \hat{v}_2) = (\alpha v_1 + (1 - \alpha) \hat{v}_1)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) - \tag{21}
\]

\[
(\alpha v_2 + (1 - \alpha) \hat{v}_2)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) + \tag{22}
\]

\[
(\alpha v_2 + (1 - \alpha) \hat{v}_2)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) + \tag{23}
\]

\[
(\alpha v_2 + (1 - \alpha) \hat{v}_2)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) + \tag{24}
\]

\[
(\alpha v_2 + (1 - \alpha) \hat{v}_2)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) + \tag{25}
\]

\[
(\alpha v_2 + (1 - \alpha) \hat{v}_2)^T (\alpha v_3 + (1 - \alpha) \hat{v}_3) + \tag{26}
\]

\[
= (\alpha^2 v_1^T v_3 + (1 - \alpha) v_1^T \hat{v}_3 + (1 - \alpha) \hat{v}_1^T v_3 + (1 - \alpha)^2 \hat{v}_1^T \hat{v}_3) - \tag{27}
\]

\[
(\alpha^2 v_1^T v_3 + (1 - \alpha) v_1^T \hat{v}_3 + (1 - \alpha) \hat{v}_1^T v_3 + (1 - \alpha)^2 \hat{v}_1^T \hat{v}_3) - \tag{28}
\]

\[
(\alpha^2 v_2^T v_3 + (1 - \alpha) v_2^T \hat{v}_3 + (1 - \alpha) \hat{v}_2^T v_3 + (1 - \alpha)^2 \hat{v}_2^T \hat{v}_3) - \tag{29}
\]

\[
(\alpha^2 v_2^T v_3 + (1 - \alpha) v_2^T \hat{v}_3 + (1 - \alpha) \hat{v}_2^T v_3 + (1 - \alpha)^2 \hat{v}_2^T \hat{v}_3) - \tag{30}
\]

\[
= \alpha^2 [(v_1 - v_2)^T (v_3 - v_2)] + (1 - \alpha)^2 [(\hat{v}_1 - \hat{v}_2)^T (\hat{v}_3 - \hat{v}_2)] + \tag{31}
\]

\[
\alpha (1 - \alpha) [(v_1 - v_2)^T (\hat{v}_3 - \hat{v}_2) + (\hat{v}_1 - \hat{v}_2)^T (v_3 - v_2)] \tag{32}
\]

\[
= \alpha (1 - \alpha) [(v_1 - v_2)^T (\hat{v}_3 - \hat{v}_2) + (\hat{v}_1 - \hat{v}_2)^T (v_3 - v_2)]. \tag{33}
\]

When \( \alpha = 0 \) or \( \alpha = 1 \), Eq. (16) is clearly satisfied. By definition of the dot product,

\[
(v_1 - v_2)^T (\hat{v}_3 - \hat{v}_2) = \|v_1 - v_2\| \|\hat{v}_3 - \hat{v}_2\| \cos \phi = l \hat{w} \cos \phi \tag{34}
\]
and
\[(\mathbf{v}_1 - \mathbf{v}_2)^T (\mathbf{v}_3 - \mathbf{v}_2) = ||\mathbf{v}_1 - \mathbf{v}_2|| ||\mathbf{v}_3 - \mathbf{v}_2|| \cos \psi = \tilde{l} \tilde{w} \cos \psi \tag{35}\]

where $\phi$ is the angle between $(\mathbf{v}_1 - \mathbf{v}_2)$ and $(\mathbf{v}_3 - \mathbf{v}_2)$, and $\psi$ is the angle between $(\mathbf{v}_1 - \mathbf{v}_2)$ and $(\mathbf{v}_3 - \mathbf{v}_2)$. Let us define the angle between $(\mathbf{v}_1 - \mathbf{v}_2)$ and $(\mathbf{v}_1 - \mathbf{v}_2)$ as $\rho$, and the angle between $(\mathbf{v}_3 - \mathbf{v}_2)$ and $(\tilde{\mathbf{v}}_3 - \tilde{\mathbf{v}}_2)$ as $\omega$. Note that $\rho + \psi = \theta$, $\phi - \omega = \theta$, $\psi + \omega = \theta$, and $\phi - \rho = \tilde{\theta}$. If $\theta$ and $\tilde{\theta}$ have the same sign, then
\[l \tilde{w} \cos \phi + \tilde{l} \tilde{w} \cos \psi = l \tilde{w} \cos (\theta + \omega) + \tilde{l} \tilde{w} \cos (\theta - \omega) = \pm \left( \tilde{l} \tilde{w} - l \tilde{w} \right) \sin \omega \tag{36}\]

and $\rho = \omega$. If $\theta$ and $\tilde{\theta}$ have the opposite signs, then $\rho = \omega \pm \pi$ and
\[l \tilde{w} \cos \phi + \tilde{l} \tilde{w} \cos \psi = l \tilde{w} \cos (\theta + \omega) + \tilde{l} \tilde{w} \cos (\theta - \omega) = \pm \left( \tilde{l} \tilde{w} + l \tilde{w} \right) \sin \omega. \tag{37}\]

As a result, Eq. (16) is satisfied whenever $w = k\pi$ where $k \in \mathbb{Z}$. Furthermore, if $\theta$ and $\tilde{\theta}$ have the same sign, then it will also be satisfied when $\frac{1}{w} = \frac{l}{w}$ regardless of $\omega$. Next, let us consider the second property, Eq. (17),
\[||\mathbf{\tilde{v}}_1 - \mathbf{\tilde{v}}_2|| \tag{38}\]
\[= \sqrt{(\mathbf{\tilde{v}}_1 - \mathbf{\tilde{v}}_2)^T (\mathbf{\tilde{v}}_1 - \mathbf{\tilde{v}}_2)} \tag{39}\]
\[= \sqrt{(\alpha l)^2 + \left( (1 - \alpha) l \right)^2 + \alpha(1 - \alpha) \left( \tilde{l} \tilde{w} \cos \rho + \tilde{l} l \cos \rho \right)} \tag{40}\]

where $\rho$ is the angle between $(\mathbf{v}_1 - \mathbf{v}_2)$ and $(\mathbf{\tilde{v}}_1 - \mathbf{\tilde{v}}_2)$ as before. When $\rho = \pm \pi$,
\[||\mathbf{\tilde{v}}_1 - \mathbf{\tilde{v}}_2|| = \sqrt{(\alpha l)^2 + \left( (1 - \alpha) l \right)^2 - 2\alpha(1 - \alpha) \tilde{l} l} = \sqrt{\left( \alpha l - (1 - \alpha) \tilde{l} \right)^2} \tag{41}\]

which equals zero only when $\alpha l = (1 - \alpha) \tilde{l}$. When $-\pi < \rho < \pi$,
\[\sqrt{(\alpha l)^2 + \left( (1 - \alpha) l \right)^2 + \alpha(1 - \alpha) \left( \tilde{l} \tilde{w} \cos \rho + \tilde{l} l \cos \rho \right)} > \sqrt{\left( \alpha l - (1 - \alpha) \tilde{l} \right)^2} \tag{42}\]

since $\cos \rho > -1$. Therefore, Eq. (17) is satisfied whenever $-\pi < \rho < \pi$ or when $\alpha l \neq (1 - \alpha) \tilde{l}$. For the third property, Eq. (18),
\[||\mathbf{\tilde{v}}_3 - \mathbf{\tilde{v}}_2|| \tag{43}\]
\[= \sqrt{(\mathbf{\tilde{v}}_3 - \mathbf{\tilde{v}}_2)^T (\mathbf{\tilde{v}}_3 - \mathbf{\tilde{v}}_2)} \tag{44}\]
\[= \sqrt{(\alpha w)^2 + \left( (1 - \alpha) \tilde{w} \right)^2 + \alpha(1 - \alpha) \left( \tilde{w} \tilde{w} \cos \omega + \tilde{w} \tilde{w} \cos \omega \right)} \tag{45}\]

where $\omega$ is the angle between $(\mathbf{v}_3 - \mathbf{v}_2)$ and $(\mathbf{\tilde{v}}_3 - \mathbf{\tilde{v}}_2)$. Like Eq. (17), Eq. (18) is satisfied whenever $-\pi < \omega < \pi$ or when $\alpha w \neq (1 - \alpha) \tilde{w}$. Finally, let us consider
Claim. The KL divergence between each of the marginal distributions when all dimensions of the joint distributions become the sum over the KL divergence between the fourth property,

\[
(\hat{v}_1 - \hat{v}_2) + \hat{v}_3 = ((\alpha v_1 + (1 - \alpha)\hat{v}_1) - (\alpha v_2 + (1 - \alpha)\hat{v}_2)) + (\alpha v_3 + (1 - \alpha)\hat{v}_3)
\]

so Eq. (19) is always satisfied.

Therefore, \{\hat{v}_1, \hat{v}_2, \hat{v}_3, \hat{v}_4\} is a valid bounding box whenever \(\alpha = 0\) or \(\alpha = 1\). When \(\alpha \in (0, 1)\), the bounding box is valid whenever \(\rho = \omega = 0\). Recall that \(\rho = \omega\) when \(\theta\) and \(\hat{\theta}\) have the same sign. If \(\rho = \pm \pi\), then \(\alpha l \neq (1 - \alpha)\hat{l}\) for it to be valid. Likewise, if \(\omega = \pm \pi\), then \(\alpha w \neq (1 - \alpha)\hat{w}\). Furthermore, the bounding box is valid when \(-\pi < \rho = \omega < \pi\) and \(\frac{l}{\hat{l}} = \frac{w}{\hat{w}}\). Therefore, there are several conditions where the convex combination of bounding boxes will result in a valid bounding box, but it is not guaranteed.

C Independence and Kullback-Leibler Divergence

In Section 3.3 we claim that the Kullback-Leibler (KL) divergence between two joint probability distributions becomes the sum over the KL divergence between each of the marginal distributions when all dimensions of the joint distributions are assumed to be independent. We provide a proof for this claim below.

Claim. The KL divergence between \(p(x, y)\) and \(q(x, y)\) is the following:

\[
D_{KL} (p(x, y)\|q(x, y)) = D_{KL} (p(x)\|q(x)) + D_{KL} (p(y)\|q(y))
\]

when \(p(x, y) = p(x)p(y)\) and \(q(x, y) = q(x)q(y)\).

Proof.

\[
D_{KL} (p(x, y)\|q(x, y)) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x, y) \log \frac{p(x, y)}{q(x, y)} dx dy
\]

\[
= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x)p(y) \log \frac{p(x)p(y)}{q(x)q(y)} dx dy
\]

\[
= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(x)p(y) \left( \log \frac{p(x)}{q(x)} + \log \frac{p(y)}{q(y)} \right) dx dy
\]

\[
= \int_{-\infty}^{\infty} p(y) \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx dy + \int_{-\infty}^{\infty} p(x) \int_{-\infty}^{\infty} p(y) \log \frac{p(y)}{q(y)} dy dx
\]

\[
= \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx + \int_{-\infty}^{\infty} p(y) \log \frac{p(y)}{q(y)} dy
\]

\[
= D_{KL} (p(x)\|q(x)) + D_{KL} (p(y)\|q(y))
\]
D Multi-Class Object Detection and Motion Forecasting

Our method utilizes the range view representation of the LiDAR; therefore, it operates at the full range of the sensor without voxelization. As a result, it is suitable for detecting both large and small objects, e.g. vehicles as well as pedestrians and bicycles, with a single network. In Table 5, we list the object detection and motion forecasting performance of our multi-class model on ATG4D. In this case, the region-of-interest (RoI) is a circle with a diameter of 240 meters centered on the self-driving vehicle, which encapsulates the entire range of the LiDAR. To evaluate object detection, we require pedestrian and bicycle detections to have an intersection-over-union (IoU) with a ground-truth of at least 0.5 to be considered a true positive. To evaluate motion forecasting, we use a recall point of 80% at a 0.5 IoU for pedestrians and a recall point of 60% at a 0.5 IoU for bicycles. For more information on the evaluation procedure, refer to Section 4.

From Table 5, we observe that adding multiple classes does not significantly impact the model’s ability to detect and predict the motion of vehicles. Of the three classes, bicycles are the hardest to detect due to their rarity in the training set. However, the $L_2$ error for bicycles is the lowest at longer time horizons because their speed is lower than vehicles and their movement is more consistent than pedestrians. None of the existing work detects and predicts motion for multi-classes or operates at the full range of the sensor so we cannot show performance comparisons, but these results show the ability of our approach to extend to multiple classes.

| Model         | Vehicle | Bicycle | Pedestrian |
|---------------|---------|---------|------------|
|               | AP (%)  | $L_2$ (cm) | AP (%)  | $L_2$ (cm) | AP (%)  | $L_2$ (cm) |
| Vehicle Only  | 74.6    | 24 38 111 | -         | -         | -         | -         |
| Multi-Class   | 74.4    | 24 38 112 | 57.7      | 15 22 50  | 72.4      | 12 27 72  |

E Qualitative Results

Figure 6 and 7 depict qualitative results for our vehicle only and multi-class models. Vehicle predictions are shown in orange, bicycles are in maroon, and pedestrians are in purple. The ground-truth for each class is depicted in green. Although our proposed method predicts the future position of the object’s bounding box, for visualization purposes, we only render the future center of the bounding box. In addition, we visualize one standard deviation of the predicted trajectory uncertainty at each time-step. These results demonstrate our method’s ability to accurately detect and forecast the motion of objects of various sizes at different distances traveling at a variety of speeds. Furthermore, these results illustrate our approach’s capacity to reliably estimate its uncertainty.
Fig. 6: Qualitative results for our vehicle only model.
Fig. 7: Qualitative results for our multi-class model.