Voting for Voting in Online Point Cloud Object Detection

Dominic Zeng Wang
Mobile Robotics Group
University of Oxford, UK
dominic@robots.ox.ac.uk

Ingmar Posner
Mobile Robotics Group
University of Oxford, UK
ingmar@robots.ox.ac.uk

Abstract—This paper proposes an efficient and effective scheme to applying the sliding window approach popular in computer vision to 3D data. Specifically, the sparse nature of the problem is exploited via a voting scheme to enable a search through all putative object locations at any orientation. We prove that this voting scheme is mathematically equivalent to a convolution on a sparse feature grid and thus enables the processing, in full 3D, of any point cloud irrespective of the number of vantage points required to construct it. As such it is versatile enough to operate on data from popular 3D laser scanners such as a Velodyne as well as on 3D data obtained from increasingly popular push-broom configurations. Our approach is “embarrassingly parallelisable” and capable of processing a point cloud containing over 100K points at eight orientations in less than 0.5s. For the object classes car, pedestrian and bicyclist the resulting detector achieves best-in-class detection and timing performance relative to prior art on the KITTI dataset as well as compared to another existing 3D object detection approach.

I. INTRODUCTION

The sliding window approach to object detection is arguably the most frequently deployed paradigm for object detection in computer vision. However, it has been largely neglected so far for laser-based object recognition. In fact, the same paradigm seems to be equally applicable to a 3D point cloud as it is to a 2D image. The conceptual difference is not significant, one only needs to first discretise the space into a 3D voxel grid, and slide a window through all three-dimensions instead of two as in the case of images.

We conjecture that perhaps one disparaging factor has been the perceived computational burden introduced by the additional dimension, leading to a dismissal of sliding window approaches as impractical in 3D. To illustrate, assume we are interested in detecting objects in 3D within a point cloud spanning an area of $(100m)^2$ and a volume $10m$ high. If we discretise this volume into $20cm$ cells, we will have $12.5$ million grid cells. A naïve approach would place the corner of a detection window at each of these $12.5$ million cells and test whether it bounds an object of interest. Thus one would need to process ca. $12.5$ million windows (neglecting boundary conditions as they are irrelevant for the sake of this thought experiment). Even assuming (rather optimistically) that a single window can be processed within $1\mu s$, this implies a computational burden of $12.5s$ in order to process a single frame. With our proposed algorithm, the average computation time for such a case is less than $0.5$ seconds.

The key insight to achieving such efficiency is by noting that there is one fundamental difference to the structure of a 3D point cloud as compared to that of a 2D image: a 3D point cloud is sparse in that most of the space is unoccupied. We show in Section IV that there exists a duality between sliding window detection with linear classifiers and a voting scheme exercised only by the occupied cells, reducing the
amount of computation to a minimum while at the same time maintaining exact mathematical equivalence. This constitutes the key contribution of this work and results in a detector which we demonstrate to achieve best-in-class detection and timing performance relative to prior art on a number of classes on the KITTI dataset as well as compared to another existing 3D object detection approach.

II. RELATED WORKS

There exist a number of works in robotics exploring the detection of objects in 3D data. Teichman et al. [19] [18], for example, first segment input laser scans and then track the segments obtained in a sequence of scans. A boosting classifier is employed to classify each tracked sequence of segments into the classes car, pedestrian and bicyclist. Wang et al. [20] take a related approach, where a laser scan is first over-segmented into object parts, and SVM classifiers are trained to separate the segments into either foreground that belongs to objects of interest or the background. Then a clustering procedure is conducted on the foreground segments to obtain object instances. A second classification stage classifies the resulting clusters into the same set of classes car, pedestrian and bicyclist. Behley et al. [1] also take a segmentation-based approach. First, a hierarchy of segments are obtained from the input point cloud. The segments are then classified with a mixture model trained on bag-of-words descriptors. Redundant segments in the hierarchy are later removed in a greedy manner. Lai et al. [11] deploy the Kinect sensor to learn a sparse distance metric for recognising pre-segmented indoor objects with Group-Lasso regularisation.

A body of work in monocular object detection with 3D pose estimation [9] [7] also characterises objects by 3D bounding boxes identified by sliding a window in 3D. However, instead of building a 3D feature grid, detection is achieved by projecting the image fronto-parallel to each visible face of the object bounding cuboid and 2D features are then extracted for that face from the projected image. Similarly, another common approach to object detection in 3D is to select or exploit a single vantage point and to project the 3D data from a laser scanner onto an image plane to form a depth image. A 2D sliding window operation is then performed. As examples serve [14] [15] who in addition also take into account appearance information from a regular camera image. Lai et al. [10] detect objects in 3D point clouds generated from multiple views of RGB-D data by running a 2D sliding window detector on each view over both the RGB and the depth data. Then detections in 3D are obtained by first projecting 2D detection scores to the integrated 3D point cloud and fusing them using a voxel representation. In contrast, the approach we propose in this paper does not require any projection and operates solely on 3D data. Consequently, our approach is versatile enough to operate on data from popular 3D laser scanners such as a Velodyne as well as on 3D data obtained from increasingly popular push-broom configurations without having to assume a privileged vantage point.

Most closely related to our work is the sliding window approach to 3D object detection recently proposed by Song and Xiao [16], who share our aspiration of passing a 3D window through a voxel grid. In their work, computational tractability is achieved via the use of a 3D integral image. However, each window needs to be tested explicitly as to whether there are more occupied cells than a certain threshold. While this is more efficient than a naïve approach, it requires operations linear in the total number of cells in the 3D grid. In contrast, our approach is linear in the number of occupied cells, empty cells are bypassed altogether. Our approach is therefore computationally more efficient.

We are not the first to notice the duality between sliding window detection with linear classifiers and voting. Lehmann et al. [12] use a similar argument to justify the voting process in the Implicit Shape Model (ISM). In their framework named Principled Implicit Shape Model (PRISM), it is argued that the Implicit Shape Model is in fact equivalent to the sliding window detector – they are two sides of the same coin. However, there are three main differences of the derivation presented in Section IV to the PRISM framework: a) The “votes” in our derivation are not cast into a continuous search space, they vote directly for the discrete locations of the sliding window. b) There are no codebooks generated, feature vectors are not matched to any exemplars. Instead, votes are simply scalar products between weight and feature vectors. c) Finally, and most importantly, instead of a conceptual equivalence, what we demonstrate in this work, is an exact mathematical equivalence between sparse convolution and voting.

III. OVERVIEW

The steps required in our 3D sliding window detector are conceptually analogous to an image-based one. Figure 1 illustrates the process with a toy example – a small section of a real 3D laser scan containing an object of interest, a car in this case.

The input to detection is the 3D laser scan represented as a list of point locations, together with reflectance values. First, the point cloud is converted into a feature grid as follows. The 3D space is discretised according to a fixed grid size, and each occupied cell is converted into a fixed-dimensional feature vector. Cells that are not occupied by any points map to zero feature vectors. This definition is critical for exploiting the sparsity of the problem. For example, as an illustration, the middle left diagram of Figure 1 visualises the feature grid extracted over the section of point cloud shown at the top left of the same figure. Here each coloured sphere represents a feature vector extracted for an occupied cell, the absence of a sphere means the cell is unoccupied and therefore its feature vector is zero. Note the sparsity of the feature grid – coloured spheres only occupy a small subset of the entire grid.

Then conceptually, a 3D detection window of a fixed size is placed at one corner of the feature grid and slides down all three dimensions. At each window location, the feature vectors contained within its bounds are stacked up into a single long vector and passed to a classifier (for example, an SVM).
The classifier then decides whether the current location of the detection window bounds an object of interest by means of returning a detection score. Section IV is devoted to a key mathematical result that makes this step tractable.

In contrast to image-based detectors, scale is not an issue here, because the absolute scale (in metres) is known in 3D. However, rotation is a problem. Assuming objects of interest are generally upright, that is, any rotation is constrained to be about the vertical axis, in order to be able to detect objects in arbitrary orientations, we discretise the full $360^\circ$ into $N$ orientation bins and run the same detection process $N$ times on the rotated point cloud for each orientation bin.

IV. SLIDING WINDOW, SPARSE CONVOLUTION AND VOTING

It is well known that a linear classifier in the case of sliding window detection is equivalent to convolution, and therefore standard techniques such as the Fast Fourier Transform (FFT) can be applied to compute the detection scores efficiently [3]. Unfortunately, this technique does not apply in our case in 3D. Sparsity in the spatial domain does not imply sparsity in the frequency domain, thus the Fourier transform of the sparse feature grid will be dense. We prove in this section that sparse convolution is mathematically equivalent to the process of voting. This leads to an efficient way of computing the detection scores fully exploiting the sparse nature of the problem to our advantage. We offer this as our main contribution in this work. Note, the technique we are about to describe is only applicable in the case of a linear classifier.

The feature grid is naturally four-dimensional – there is one feature vector per cell, and cells span a three-dimensional grid. Let us denote the $l$th feature at cell location $(i,j,k)$ by $f_{l_{ijk}}$. At times, we will find it convenient to refer to all features computed at location $(i,j,k)$ collectively as a vector $f_{ijk}$. To keep the presentation simple and clear, we refer to the tuple $(i,j,k)$ by a single variable, e.g. $\phi = (i,j,k)$. If the grid has dimensions $(N_x^G, N_y^G, N_z^G)$, we may define a set $\Phi = \{0, N_x^G\} \times \{0, N_y^G\} \times \{0, N_z^G\}$. Here the notation $[m,n)$ is to be understood as the standard half open interval defined over the set of integers, i.e. $[m,n) = \{q \in \mathbb{Z} : m \leq q < n\}$, and “$\times$” denotes the set Cartesian product. Thus $\Phi$ is the set of indices on the feature grid, and any $\phi \in \Phi$ index a particular cell of the grid. In this notation $f_{ijk}$ can be written in the cleaner form $f_{\phi}$ (this indexing notation is illustrated in Figure 2(a)).

Recall that by definition $f_{\phi} = 0$ if the cell at $\phi$ is not occupied. We can capture this concept by defining a set $\Phi^\star \subset \Phi$ that represents the subset of cell locations that are occupied. Thus $\phi \in \Phi \setminus \Phi^\star \implies f_{\phi} = 0$. The feature grid is sparse.

Similarly, if the dimensions of the detection window are $(N_x^W, N_y^W, N_z^W)$, then define the set $\Theta = \{0, N_x^W\} \times \{0, N_y^W\} \times \{0, N_z^W\}$ that is the set of indices local to a detection window, we may denote the weights of the linear classifier associated with a cell location in the window $\theta \in \Theta$ as $w_{\theta}$ (an example is also illustrated in Figure 2(a)). In contrary to the feature grid, the weights are dense.

To save us from worrying about boundary conditions, we define the feature vectors and weight vectors to be zero if its index is outside the bounds. This extends the set of indices in both cases (feature and weights) to the full $\mathbb{Z}^3$.

In addition to the set of indices on the feature grid $\Phi$, the set of indices in a detection window $\Theta$, we will need to define a third set of indices $\Psi$ that contains the indices of all detection window locations that may possibly receive a nonzero detection score. If we define the anchor point for a detection window to be at its local coordinates $\theta = (0, 0, 0)$ (cf. Figure 2(a)), then the set of all possible window locations can be captured by the set of all possible locations on the feature grid of a window’s anchor point. Thus the set of indices of the anchor points of all windows that can possibly receive a nonzero detection score is given by $\Psi = [1-N_x^W, N_x^G) \times [1-N_y^W, N_y^G) \times [1-N_z^W, N_z^G)$.

In the derivations that follow, we will consistently use an index $\phi$ to index a feature vector on the feature grid, $\theta$ to index a weight vector in the detection window, and $\psi$ to index a window location (anchor point position on the grid). We are now in a position to derive the main result of this section.

Theorem 1. The score $s_\psi$ for the detection window with origin placed at grid location $\psi$ can be written as a sum of votes from occupied cells that fall within the detection window.

Proof: We begin by writing down the explicit form for the detection score $s_\psi$ according to the linear classifier

$$s_\psi = \sum_{\theta \in \Theta} f_{\psi+\theta} \cdot w_{\theta} \cdot,$$

where “$\cdot$” denotes the vector dot product. Since $w_{\theta} = 0$ whenever $\theta \notin \Theta$, the summation can be extended to the entire
Different, we can also imagine votes being cast from
voted by its contributing occupied cells. This is illustrated in
a second view, in that each detection window location is
particular window locations. This view of the voting process
in support of the existence of an object of interest at those
Fig 2(a). Indeed, we can also imagine votes being cast from

$Z^3$, then after a change of variables $\phi = \psi + \theta$ we arrive at

$$s_\psi = \sum_{\theta \in Z^3} f_{\psi+\theta} \cdot w_{\theta} \tag{2}$$

$$= \sum_{\phi \in Z^3} f_{\phi} \cdot w_{\phi-\psi} \tag{3}$$

$$= \sum_{\phi \in \Phi} f_{\phi} \cdot w_{\phi-\psi} \tag{4}$$

$$= \sum_{\phi \in \Phi^*} f_{\phi} \cdot w_{\phi-\psi} \cdot \tag{5}$$

Equation (4) follows from Equation (3) because $f_{\phi} = 0 \ \forall \phi \notin \Phi$, and Equation (5) then follows from Equation (4) because $f_{\phi} = 0$ for unoccupied cells by definition.

Now, we note again, that $w_{\phi} = 0 \ \forall \theta \notin \Theta$, this implies that the summation in Equation (5) further reduces to

$$s_\psi = \sum_{\phi \in \Phi^*} f_{\phi} \cdot w_{\phi-\psi} \cdot \tag{6}$$

where $\Gamma_{\psi} = \{ \phi \in Z^3 : \phi - \psi \in \Theta \} = \{ \phi \in Z^3 : \exists \theta \in \Theta, \phi = \psi + \theta \}$.

If we now define the vote from the occupied cell at location $\phi$ to the window at location $\psi$ as $v_{\phi, \psi} = f_{\phi} \cdot w_{\phi-\psi}$, Equation (6) becomes

$$s_\psi = \sum_{\phi \in \Phi^* \cap \Gamma_{\psi}} v_{\phi, \psi} \cdot \tag{7}$$

This completes the proof.

Theorem 1 gives sliding window detection on a sparse grid a second view, in that each detection window location is voted by its contributing occupied cells. This is illustrated in Figure 2(a). Indeed, we can also imagine votes being cast from each occupied cell for different detection window locations in support of the existence of an object of interest at those particular window locations. This view of the voting process is summarised by the next corollary.

**Corollary 1.** The three-dimensional score array $s$ can be written as a sum of arrays of votes one from each occupied cell.

**Proof:** First, we note that $s$ is a function that maps elements in $Z^3$ to real numbers (the detection scores at different window locations), that is $s : Z^3 \rightarrow \mathbb{R}$.

With this view in mind, we turn our attention back to Equation (5), with our previous definition of the vote $v_{\phi, \psi} = f_{\phi} \cdot w_{\phi-\psi}$, we arrive at

$$s_\psi = \sum_{\phi \in \Phi^*} v_{\phi, \psi} \cdot \tag{8}$$

Now, $v$ is defined for each $\phi, \psi \in Z^3$. Given a fixed $\phi$, with some abuse of notations, we define a function $v_{\phi} : Z^3 \rightarrow \mathbb{R}$ such that $v_{\phi}(\psi) = v_{\phi, \psi}$ for all $\psi \in Z^3$. It is now obvious that the three-dimensional score array $s$ can be written as

$$s = \sum_{\phi \in \Phi^*} v_{\phi} \cdot \tag{9}$$

We now turn our attention to the structure of the 3D array $v_{\phi}$. By definition, $v_{\phi}(\psi) = v_{\phi, \psi} = f_{\phi} \cdot w_{\phi-\psi}$, this implies that $v_{\phi}(\psi) = 0$ whenever $\phi - \psi \notin \Theta$. Recall that $\phi$ specifies the index of the occupied cell where the votes originate from, and $\psi$ the window location a vote is being cast to, this means only windows at locations satisfying $\phi - \psi \in \Theta$ receive possibly a non-zero vote from the cell. Now given a fixed $\phi$, define the set $\Lambda_{\phi} = \{ \psi \in Z^3 : \phi - \psi \in \Theta \} = \{ \psi \in Z^3 : \exists \theta \in \Theta, \psi = \phi - \theta \}$. Then the argument above limits the votes from cell $\phi$ to the subset of window locations given by $\Lambda_{\phi}$. Referring to Figure 2(b), the red spheres in the figure represents the location of the occupied cell where the votes originate from, and blue cubes indicate window locations that will receive votes from $\phi$, that is, the set $\Lambda_{\phi}$. $\Lambda_{\phi}$ thus includes all window locations whose origins are located in a window of cells that has the same dimensions as the detection window but going backwards from the cell location $\phi$.

With the insight of the structure of voting gained, Corollary 1 readily translates into an efficient algorithm – Algorithm 1 – to compute the array of detection scores $s$ by voting.

### Algorithm 1: Computing the score array given weights of the classifier and the feature grid via voting. See text for details.

```
Function ComputeScoreArray(w,f)
    Input: Weights of the classifier w and the feature grid f.
    Output: The array of detection scores s.
    // Initialise the score array with zero values.
    for $\psi \in \Psi$ do
        $s_{\psi} \leftarrow 0$;
    end
    // Begin voting.
    for $\phi \in \Phi^*$ do
        for $\theta \in \Theta$ do
            $s_{\phi-\theta} \leftarrow s_{\phi-\theta} + f_{\phi} \cdot w_{\theta}$;
        end
    end
    return s;
end
```

V. Feature Extraction

For our feature representation, we compute three shape factors [21] on the scatter of points within the occupied cell, the mean and variance of the reflectance values of points contained in the cell, and a binary occupancy feature that is 1 for a cell that is occupied and 0 if it is not. This gives a total of 6 features for each cell.

These simple features may not appear to be descriptive when considering just a single cell. However, considering an object
is described by a collection of cells (and that the relative positions of these cells do matter), the overall descriptive power of these apparently simple features can be rich.

We stress here that designing the best features for a sliding window detector in 3D is not the main focus of this work. However, the simple feature set we have chosen gives a good detection performance as is demonstrated in Section VII-B.

VI. NON-MAXIMUM SUPPRESSION

To remove duplicate detections, we apply a non-maximum suppression procedure analogous to the standard techniques commonly applied in Computer Vision [2, 6, 13]. Specifically, we follow the greedy approach described in [6].

The non-maximum suppression proceeds as follows. All window locations (at all angles of orientation) with a detection score higher than a threshold σ are first sorted in descending order of their detection scores. Then they are taken one-by-one in that order, and compared with the current list of accepted window locations (initialised to be empty). A window location is accepted and added to the list of accepted windows if it doesn't overlap with any of the previously accepted object windows by more than a given threshold. The overlap between two object windows is computed as the ratio of the volume of the intersection over the volume of the union.

VII. EVALUATION

To facilitate supervised learning, we take advantage of the publicly available KITTI dataset [8]. The object detection benchmark from the KITTI dataset supplies synchronised camera and Velodyne frames, with objects annotated in both image and laser data. Specific to our interests is that the annotations in the laser data are given as complete oriented 3D bounding boxes bounding the object of interest in a canonical orientation.

We evaluate the performance of the proposed 3D sliding window detector on the classes car, pedestrian and bicyclist, and demonstrate its superior performance in terms of its absolute performance based on common evaluation metrics and the relative performance compared quantitatively with both an existing segmentation-based approach to 3D object detection [20], and results published on the KITTI object benchmarking website.

A. Training

The standard KITTI object detection benchmark contains a labelled training set and a labelled test set. However, the labels on the test set are held back for evaluation purposes. Since what we are interested in here is a fair evaluation of the performance of the sliding window detector on 3D data, whereas KITTI is primarily a vision dataset, we create our own training and test datasets from the labelled data in KITTI that is publicly available (i.e. the original “training” dataset) by randomly splitting it into two parts and then test the detector’s performance based on metrics that are more suitable to evaluating detections in 3D (cf. Section VII-B).

Specifically, we randomly split the 7481 labelled frames available into 80/20 proportions for training and testing respectively. The numbers of frames contained in the resulting training and test sets, together with other information, are tabulated in Table I.

For training the linear SVM classifier, we use the LIBLIN-EAR library [5]. An initial set (equal to the number of positive examples) of negative examples are randomly sampled from the training data taking care not to overlap with any positive examples. Taking this initial set of training examples, we adopt the standard hard negative mining technique from image-based object detectors (e.g. [17, 2, 6]). Specifically, a classifier is first trained on the initial training set. After training, the classifier is applied back on all the training frames. All false positive detections from this classifier on all the training frames are collated, and sorted in descending order of the detection score. The first N (or all of the false positives if there are less than N of them) are then taken and added to the set of negative examples. The classifier is then retrained with this updated training set and this process may iterate for a predefined number of rounds. In all our experiments that follow, we fix N to be 10000 and conduct 20 rounds of hard negative mining.

A disadvantage of sliding window approaches is that artefacts may be introduced during the discretisation process. Because window locations are only searched on the discretised feature grid (and the discretised angle of rotation), it is unlikely an object is captured in the detection window in precisely its canonical pose. However, the positive examples for training are extracted from manual labels, the objects contained are therefore centred and facing forward. To compensate for this discrepancy, for each positive example, we randomly sample 10 slightly translated and rotated (about the vertical axis) versions of it, and append them to the set of positive examples for training.

B. Evaluation Strategy

The object labels provided by the KITTI dataset on the 3D laser data are comprehensive in the sense that, as well as obvious object instances, challenging objects that are heavily occluded or very sparsely sampled due to being at a large distance from the sensor are also included. The included objects may at times be as challenging as being described by only a handful of laser measurements (see, for example, the left column of Figure 3).

This motivates us to divide the labelled object instances into different difficulty levels similar to the original KITTI specification [8], to respect the complete set of labels from the dataset at the same time not to place unreasonable demands to the detection system.

The original KITTI specification is tailored specifically to vision-based detection systems. Here, we first take a closer look into the dataset for the types of labelled object instances provided in the 3D laser data, and based on that, devise suitable criteria for dividing the objects into the easy, moderate and hard difficulty levels.
Fig. 3. Examples of labelled object instances from the training set of different difficulties. Left column: hard (but not moderate), instances containing numbers of measurements \( m < 50 \). Middle column: moderate (but not easy), instances containing numbers of measurements \( 50 \leq m < 150 \). Right column: easy, instances containing numbers of measurements \( m \geq 150 \). See text for details.

The proposed sliding window detector is implemented as a C++ library. Noting that the computation for each orientation bin (cf. Section VII-A) is completely independent of each other, hence it falls within the “embarrassingly parallelisable”

C. Detection Performance

The proposed sliding window detector is trained with the training set according to the procedure outlined in Section VII-A, and evaluated on the test set. There are only three parameters to the detector, the grid size \( \delta \), the number of angular bins \( N \) and the overlap threshold \( t_o \) for non-maximum suppression (cf. Section VI). In all our experiments, we set \( \delta = 0.2m \), \( N = 8 \), and \( t_o = 0.01 \) for the car class, \( t_o = 0.5 \) for the pedestrian class, and \( t_o = 0.1 \) for the bicyclist class. The class-dependent \( t_o \) parameters are chosen to reflect expected proximity between objects of that class.

D. Timing

The proposed sliding window detector is implemented as a C++ library. Noting that the computation for each orientation bin (cf. Section III) is completely independent of each other, hence it falls within the “embarrassingly parallelisable”

![Table I: Details of data splits for training and testing.](image-url)

![Figure 3: Examples of labelled object instances from the training set of different difficulties.](image-url)
paradigm. We therefore take full advantage of modern CPUs’ multi-core architectures by treating the computation for each orientation bin as an independent job unit, which may be executed on different threads.

We evaluate the timing aspects of our implementation on a MacBook Pro equipped with a quad-core 2.8GHz Intel i7 CPU and 16GB of RAM. We take the same classifiers trained in Section VII-C and select the detection threshold for each object class from the Precision-Recall curve evaluated on the test dataset at the easy difficulty level (cf. Figure 4) at balanced values of Precision and Recall. Then the obtained classifiers are applied over the test set from the original KITTI benchmark (i.e. the test set whose ground truth labels are not publicly available). Timing statistics are collected, and tabulated in Table II. The computation time per frame averages to under 500ms for all three object types.

### E. Comparison with an Existing Approach on 3D Object Detection

Although standard procedures for benchmarking vision-based detectors exist on the KITTI dataset, a corresponding benchmarking standard is currently missing for laser. It is difficult to obtain fair quantitative comparison with existing approaches to object detection in 3D. In this section, we quantitatively compare the sliding window object detector proposed with one existing approach to object detection in 3D, the segmentation-based detector described in [20].

To ensure a fair comparison with the existing results quoted in [20], we follow exactly the same evaluation procedure and use exactly the same evaluation dataset on which results reported in [20] are obtained. In particular, each oriented object box as output from the sliding window detector is converted to a corresponding object segment by taking all points that fall within the window. Figure 5 presents the Precision-Recall curves in blue of the sliding window detector evaluated in this way, compared with the results quoted in Table II of [20] for the three different detection schemes proposed in that work. As may be noted from the figure, the sliding window detector outperforms the segmentation-based detector by a significant margin on all three object categories. Because the segmentation-based detector is purely shape-based, it does not use appearance information from the reflectance values, to compare the two approaches on a common footing, we also include in Figure 5 the Precision-Recall curves, in red, of a variant of the sliding window detector trained with only the shape-based features from our feature set. Though the performance compares less favourably with using the full feature set as one would expect, it still outperforms the segmentation-based detector.

### F. Comparison with Other Laser-Only Methods Evaluated on the KITTI Dataset

As mentioned in the previous section, while it is currently difficult to benchmark the proposed approach with other laser-based detection methods in its native form due to the lack of an established 3D object detection benchmark, standard benchmarking for vision-based detectors is readily available for the KITTI dataset.

As an attempt to broaden the reach of our comparative studies, we have therefore submitted results from the proposed 3D detector to the 2D KITTI object benchmark by projecting the obtained 3D detections to the image plane with the provided calibration parameters.

We must stress here, although evaluated following the same procedures on the same test dataset, any comparison with other published results of vision-based method is inevitably heavily biased. Our submission does not take advantage of any image data, and the evaluation criteria followed are according to the original difficulty specification defined with respect to vision. Because of the inherent difference in sensor modality, attempting to compare laser-based and vision-based methods on a completely fair basis is challenging if not impossible. For example, what is difficult in appearance in vision may not be difficult in laser and vice versa.

1The submitted results may be viewed on the KITTI object benchmarking website [http://www.cvlibs.net/datasets/kitti/eval_object.php](http://www.cvlibs.net/datasets/kitti/eval_object.php) under the method name Vote3D.
Recall
0 0.2 0.4 0.6 0.8 1
Precision
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1 Car
All Features
Without Appearance Features
Seg-Based: N-Class
Seg-Based: F/B Binary
Seg-Based: F/B N-Class

Recall
0 0.2 0.4 0.6 0.8 1
Precision
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1 Pedestrian
All Features
Without Appearance Features
Seg-Based: N-Class
Seg-Based: F/B Binary
Seg-Based: F/B N-Class

Recall
0 0.2 0.4 0.6 0.8 1
Precision
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1 Bicyclist
All Features
Without Appearance Features
Seg-Based: N-Class
Seg-Based: F/B Binary
Seg-Based: F/B N-Class

Fig. 5. Precision-Recall curves generated by applying the sliding window detectors over the evaluation dataset in [20]. Included in the same figures are variants of the detectors with the appearance features turned off for fair comparison. Results reported in Table II of [20] are plotted on common axes as points.

### TABLE III

<table>
<thead>
<tr>
<th>Method</th>
<th>Rank</th>
<th>Car Moderate</th>
<th>Car Easy</th>
<th>Car Hard</th>
<th>Pedestrian Moderate</th>
<th>Pedestrian Easy</th>
<th>Pedestrian Hard</th>
<th>Bicyclist Moderate</th>
<th>Bicyclist Easy</th>
<th>Bicyclist Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vote3D (Ours)</td>
<td>20</td>
<td>47.99%</td>
<td>56.80%</td>
<td>42.57%</td>
<td>17</td>
<td>35.74%</td>
<td>44.48%</td>
<td>33.72%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSoR</td>
<td>22</td>
<td>26.13%</td>
<td>34.79%</td>
<td>22.69%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mBoW [1]</td>
<td>23</td>
<td>23.76%</td>
<td>36.02%</td>
<td>18.44%</td>
<td>18</td>
<td>31.37%</td>
<td>44.28%</td>
<td>30.62%</td>
<td>10</td>
<td>21.62%</td>
</tr>
</tbody>
</table>

However, currently two (and only two) other laser-only methods exist on the benchmarking website that published their detection results in a similar manner. Table III is an excerpt of the table of published results on the KITTI website that puts our submitted results (Vote3D) in direct comparison to these methods. For each category, the table lists the rank of the method in the benchmark, and the average precision for the moderate, easy and hard difficulty levels respectively (submissions are ranked according to the average precision evaluated on the moderate difficulty in the KITTI benchmark, hence the ordering). The CSoR submission is an anonymous submission, and evaluated on the car class only. The number in bold in each column denotes the top entry of that column (higher rank or higher average precision). As can be noted from the table, the proposed sliding window detector compares favourably winning on all columns.

### VIII. CONCLUSIONS

The sliding window approach to object detection, while ubiquitous in the Computer Vision community, is largely neglected in 3D laser-based object detectors. This may be due to its perceived computational inefficiency. In this paper we demonstrate that by fully exploiting the sparsity of the problem, exhaustive window searching in 3D can be made extremely efficient. We prove the mathematical equivalence between sparse convolution and voting and devise an efficient algorithm to compute exactly the detection scores at all window locations. The result is a fast, effective method applicable to generic 3D point clouds constructed from single or multiple vantage points for a variety of object types. The voting scheme proposed here enables the processing of a point cloud containing over 100K points in less than 0.5s while achieving state-of-the-art detection performance relative to prior art on a number of object categories from the KITTI dataset as well as compared to another existing 3D object detection approach.

### ACKNOWLEDGEMENTS

This work was supported by the EPSRC Leadership Fellowship Grant EP/I005021/1, and EPSRC Grant EP/J012017/1. The authors would like to thank Paul Newman for his support and many helpful discussions.

### REFERENCES


