Fast and Compact Image Segmentation using Instance Stixels

Hehn, Thomas; Kooij, Julian F.P.; Gavrila, Dariu M.

DOI
10.1109/TIV.2021.3067223

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Intelligent Vehicles

Fast and Compact Image Segmentation Using Instance Stixels

Thomas Hehn, Julian Kooij, *Member, IEEE*, and Dariu Gavrila, *Member, IEEE*

***Abstract*—State-of-the-art stixel methods fuse dense stereo disparity and semantic class information, e.g., from a Convolutional Neural Network (CNN), into a compact representation of driveable space, obstacles and background. However, they do not explicitly differentiate instances within the same semantic class. We investigate several ways to augment single-frame stixels with instance information, which can be extracted by a CNN from the RGB image input. As a result, our novel Instance Stixels method efficiently computes stixels that account for boundaries of individual objects, and represents instances as grouped stixels that express connectivity. Experiments on the Cityscapes dataset demonstrate that including instance information into the stixel computation itself, rather than as a post-processing step, increases the segmentation performance (i.e., Intersection over Union and Average Precision). This holds especially for overlapping objects of the same class. Furthermore, we show the superiority of our approach in terms of segmentation performance and computational efficiency compared to combining the separate outputs of Semantic Stixels and a state-of-the-art pixel-level CNN. We achieve processing throughput of 28 frames per second on average for 8 pixel wide stixels on images from the Cityscapes dataset at 1792 × 784 pixels. Our Instance Stixels software is made freely available for non-commercial research purposes.**

***Index Terms*—Autonomous vehicles, image segmentation, machine vision.**

I. Introduction

SELF-DRIVING vehicles require a detailed understanding of their environment in order to react and avoid obstacles as well as to find their path towards their final destination. In particular, stereo vision sensors obtain pixel-wise 3D location information about the surrounding, providing valuable spatial information on nearby free space and obstacles. However, as processing should be as fast as possible, it is essential to find a compact and efficiently computable representation of sensor measurements which is still capable to provide adequate information about the environment [2], [3]. A common approach is to create a dynamic occupancy grid for sensor fusion [4] and tracking [5], which provides a top-down grid cell representation of occupied space surrounding the ego-vehicle. Still, directly aggregating depth values into an occupancy grid alone would disregard the rich semantic information from the intensity image, and the ability to exploit the local neighborhood to filter noise in the depth image.

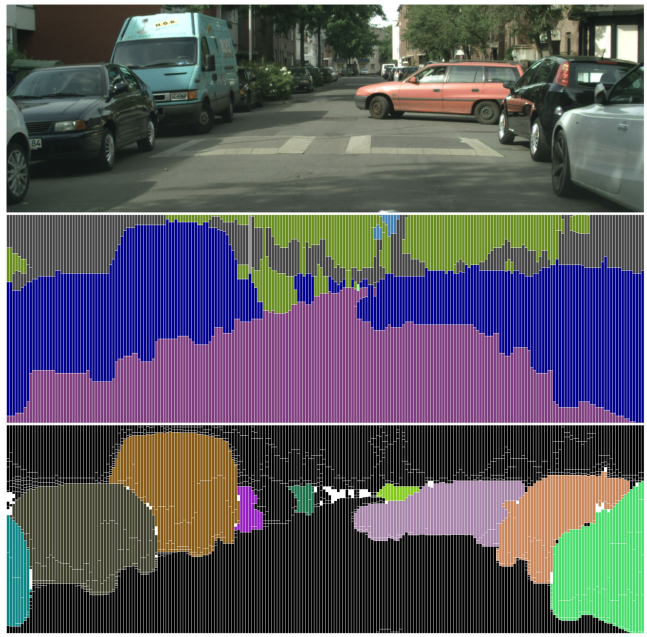


Fig. 1. Top: Input RGB image (corresponding disparity image not shown). Middle: Semantic Stixels [1] use a semantic segmentation CNN to create a compact stixel representation which accounts for class boundaries (stixel borders: white lines, arbitrary colors per class). Note that a single stixel sometimes covers multiple instances, e.g., multiple cars. Bottom: Our *Instance Stixels* algorithm also accounts for instance boundaries using additional information learned by a CNN and clusters stixels into coherent objects (arbitrary colors per instance).

A popular alternative in the intelligent vehicles domain is the "stixel" representation, which exploits the image structure to reduce disparity artifacts, and is computed efficiently [6]. By grouping pixels into rectangular, column-wise super-pixels based on the disparity information, stixels reduce the complexity of the stereo information. Later, class label information obtained from deep learning has been incorporated into the stixel computation and representation, so-called Semantic Stixels [1]. Yet, obstacles are still just a loose collection of upright "sticks" on an estimated ground plane, lacking object level information. For example, the car stixels in the middle row of Fig. 1 do not indicate where one car starts and its neighboring car ends.

Manuscript received April 20, 2020; revised September 16, 2020; accepted March 10, 2021. Date of publication March 18, 2021; date of current version April 21, 2022. This work was supported by Dutch Science Foundation NWO-TTW within the Sensing, Mapping and Localization Project (nr. 14892). *(Corresponding author: Dariu Gavrila.)*

The authors are with the Intelligent Vehicles Group, TU Delft 2628 CD Delft, The Netherlands (e-mail: t.m.hehn@tudelft.nl; j.f.p.kooij@tudelft.nl; d.m.gavrila@tudelft.nl).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TIV.2021.3067223.

Digital Object Identifier 10.1109/TIV.2021.3067223

This paper introduces an object level environment representation extracted from stereo vision data based on stixels. Our method improves upon state-of-the-art stixel methods [1], [7] that only consider semantic class information, by adding instance information extracted with a convolutional neural networks (CNN) from the input RGB image. This provides several benefits: First, we obtain better stixels boundaries around objects by fusing disparity, semantic, and instance information in the stixel computation. Second, stixels belonging to an object are connected vertically and horizontally (see bottom image of Fig. 1) by clustering them based on semantic and instance information. Third, the processing is more efficient than computing Semantic Stixels [1] and per-pixel instance labels separately.

II. Related Work

The idea of stixels, regarding objects as sticks standing perpendicular on a ground plane, was introduced by [6]. The stixel algorithm has found diverse applications in the autonomous driving domain. Stixels were used as an integral part of the pipeline for the Berthe Benz drive [8]. [9] develop a collision warning system using only stereo-based stixels and [10] used stixels to detect small unknown objects, such as lost cargo. The original idea was further extended in [11] to a multi-layer representation which used a probabilistic approach, i.e., stixels do not need to be connected to the ground plane anymore. In the multi-layer representation, stixels segment the entire image into rectangular super-pixels, classified as ground, object or sky. Additionally, a dynamic programming scheme was presented for efficient real-time computation of stixels. For even faster computation, this dynamic programming scheme was then also implemented for the Graphical Processing Unit (GPU) by [12]. In [13] stixels were compared with other super-pixel algorithms as basis for multi-cue scene labeling.

The general stixel framework offers various possibilities for extensions and modifications. For instance, [14] compared the effects of different methods for initial ground manifold modeling. Driven by the requirements of autonomous driving, [15] applied a Kalman filter to track single stixels. Stixel tracking was then further improved by [16]. Yet, stixels are generally tracked independently and not as parts of an object. In order to obtain object information [17]–[20] group the Dynamic Stixels based on shape cues and graph cuts and thus rely on tracking Stixels over time. Stixels are also applied in semantic scene segmentation with more general classes than ground, object and sky. For this purpose, semantic information can be obtained by using object detectors for suitable classes [21] or Random Decision Forest classifiers [22] and then including that information in the Stixel generation process. [1] extend this idea by incorporating the output of a Fully Convolutional Neural Network (FCN) in the probabilistic stixel framework. They named their approach Semantic Stixels. Based on Semantic Stixels and focusing on non-flat road scenarios, [7] generalize stixels to also model slanted surfaces, e.g., not strictly perpendicular to the road anymore, including piece-wise linear road surfaces.

Meanwhile, many more deep neural network architectures have been proposed in the computer vision literature to improve classification and image segmentation tasks on per pixel basis. For instance, Residual Neural Networks [23] facilitate training of deeper networks by learning residual functions. Dilated Residual Networks [24] (DRN) improve on this work by maintaining a higher resolution throughout the fully connected network, while working with the same receptive field. As a consequence, they are useful for applications that require spatial reasoning such as object detection or, as in our case, instance segmentation. In order to enforce consistency between semantic and instance segmentation, recently the term panoptic segmentation was introduced in [25] and has led to further improvement in the field [26]. Unfortunately, one cannot treat instance segmentation as a classification problem, as is done for semantic segmentation. A main reason is that the number of instances varies per image, which prohibits a one-to-one mapping of network output channels to instances. Instead of predicting instance labels directly, [27] trains a CNN to map each pixels in an image onto a learned low-dimensional space such that pixels from the same instance map close together. Object masks are then obtained in post-processing by assigning pixels to cluster centers in this space. [28] instead use supervised learning to map pixels to a specific target space, namely the 2D offsets from the given pixel towards its instance's center and then rely on clustering all pixels into instances. The Box2Pix method [29] uses 2D center offset predictions for instances, but instead of clustering they are associated with bounding boxes found through a bounding box detection branch. In order to avoid an additional bounding box detection branch, [30] learn a clustering bandwidth and confidence per pixel and thereby speed up the grouping of pixels to instances.

Our objective is to create efficient stixel representations rather than pixel-accurate instance segmentation in images, and to avoid overhead of clustering all pixels into instances before reducing them to a compact representation. Still, we follow insights from the work on per-pixel instance segmentation to improve stixel computation, deal with the unknown number of instances in an image, and enable the clustering of stixels into instances. Building upon our prior conference publication [31], the main contributions are thus summarized as:

- We present Instance Stixels, a method to include instance information into stixels computation, which creates better stixels, and allows grouping to instance IDs from a single stereo frame.
- We investigate three different ways to include the instance information, and show that adding the information into the stixel computation itself results in more accurate instance representations than only using it to cluster Semantic Stixels or alternatively assigning Semantic Stixels to instances using pixel-based methods. Further we compare the trade-off between computation speed and instance segmentation performance for these three variations to showcase the favorable properties of Instance Stixels.
- We investigate the use of a novel regularizer for Instance Stixels which replaces the former prior term in Stixels.

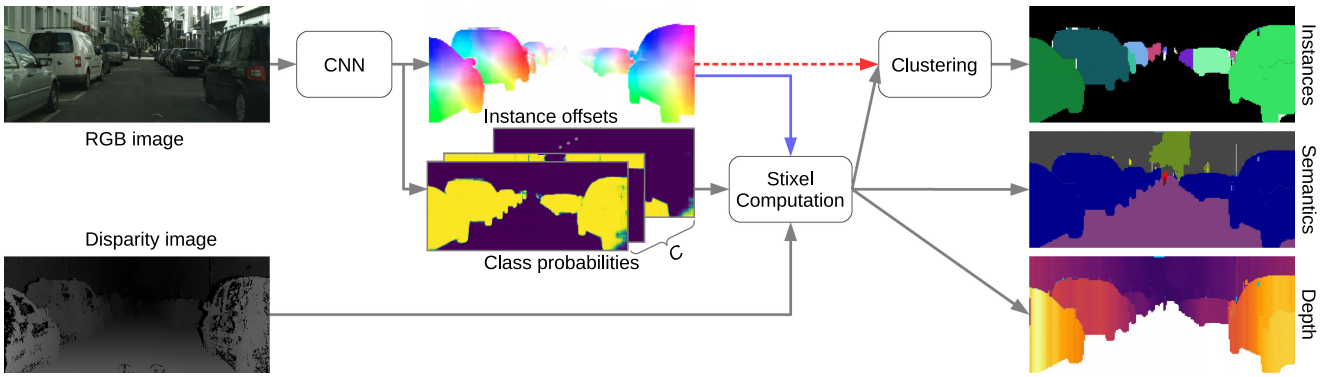


Fig. 2. Instance stixel pipeline applied to a *RGB* and *disparity* input image pair obtained from a stereo camera. The RGB image is processed by a Convolutional Neural Network (*CNN*) to predict *offsets* to the instance centers (HSV color coded) and per-pixel semantic *class probabilities* (visualized as color gradient). The class probabilities are fused with the disparity input image in the *Stixel computation* to provide a super-pixel representation of the traffic scene, which unifies *Semantics*, *Depth* and additionally *Instance* output (left images). In the baseline algorithm (*Semantic Stixels + Instance*, dashed red arrow) the obtained stixels are clustered based on the instance offsets to assign stixels to instances (not shown). In contrast, our proposed algorithm (*Instance Stixels*, blue arrow) fuses the instance offset information with the other two channels in the Stixel Computation. Subsequently, stixels are also clustered to form instances, but with improved adherence of stixels to instance boundaries (top right image, arbitrary colors).

This simplifies the model and leads to improved instance segmentation.

- Our entire implementation of the optimized pipeline for Semantic Stixels and Instance Stixels is provided as open-source to the scientific community for non-commercial research purposes.

III. Methods

This section will first briefly summarize the original disparity Stixel and Semantic Stixel formulations in Section III-A. Section III-B then explains how to integrate the instance information from a trained CNN into the stixel computation itself for improved stixel segmentation. Finally, Section III-C will discuss how the instance information can be used to cluster stixels belonging to the same object.

The clustering step could be applied to any stixel computation method. We therefore consider two options:

- Clustering stixels from a standard Semantic Stixels method [1], such that instance offset information is only considered here at this final clustering step. This baseline approach corresponds to the red arrow in Fig. 2. In our experiments we shall refer to this combination as the **Semantic Stixels + Instance** method.
- Clustering based on our novel instance-aware stixels computation from Section III-B, see the blue arrow in Fig. 2. We name this combination our novel **Instance Stixels** method.

Conceptually, Instance Stixels are a natural extension to Semantic Stixels as they extend disparity and semantic segmentation information with additional instance information to compute a compact representation from a stereo image pair. These stixels also receive an object id which groups them into instances.

A. Stixels

In the following, an outline of the derivation of the original Stixels and Semantic Stixels framework is presented. For a more detailed derivation, see [32] and [1].

1) Disparity Stixels: Following the notation of [32], the full stixel segmentation of an image is denoted as $L = \{L_u | 0 \leq u < W\}$ with W being the total number of stixel columns in the image. Thus, given a selected stixel width w, it follows that $W = \frac{\text{image width}}{w}$. The segmentation of column u contains $L_u = \{s_n | 1 \leq n \leq N_u \leq h\}$ contains at least one but at most height h stixels s_n. A stixel $s_n = (v_n^b, v_n^t, c_n, f_n(v))$ is described by the bottom and top rows, respectively v_n^b and v_n^t, that delimit the stixel. Additionally, a stixel is associated with a class $c_n \in \{g, o, s\}$ (i.e., ground, object, sky) and a function f_n which maps each row of the image to an estimated disparity value.

The aim is to find the best stixel segmentation L^* given a measurement (e.g., a disparity image) D, i.e., it maximizes the posterior probability

$$L^* = \arg\max_L p(L|D). \tag{1}$$

According to Bayes' rule, this can be rewritten as

$$p(L|D) = \frac{p(D|L)p(L)}{p(D)}. \tag{2}$$

Here, the normalization factor $p(D)$, constant in L, can be discarded in the maximization task. Since each column $u \in \{0, \ldots, W-1\}$ of the image is treated independently, the MAP objective can further be simplified:

$$L^* = \arg\max_L \prod_{u=0}^{W-1} p(D_u|L_u)p(L_u). \tag{3}$$

Here, $p(D_u|L_u)$ denotes the column's likelihood of the disparity data, and $p(L_u)$ is a prior term modeling the pairwise interaction of vertically adjacent stixels. This is explained in more detail in [11].

Assuming all rows are equally likely to separate two stixels, the column likelihood term can be written as product of individual terms for N_u stixels, $L_u = \{s_1, \ldots, s_{N_u}\}$. Since only disparity values of the rows within each stixel contribute to its likelihood, those terms can in turn be factorized over the rows $v_n^b \leq v \leq v_n^t$ of each stixel $n \in \{1, \ldots, N_u\}$. Hence, the final

objective is [32]:

$$L^* = \arg\max_{L} \prod_{u=0}^{W-1} \prod_{n=1}^{N_u} \prod_{v=v_n^b}^{v_n^t} p(d_v|s_n, v)p(L_u). \quad (4)$$

Here the term $p(d_v|s_n, v)$ includes different disparity models per geometric class. For sky stixels this model is simple: $f_{\text{sky}}(v) = 0$. The disparity of object stixels is assumed to be normally distributed around the mean stixel disparity $f_{\text{object},n}(v) = \frac{1}{v_n^t+1-v_n^b} \sum_{v'=v_n^b}^{v_n^t} d_v$. Furthermore, ground stixels rely on a previous estimation of the ground plane parameters α (the slope) and v_{horizon} (horizon estimate in the image), which can be obtained for example from v-disparity [33]. The assumed disparity model for ground stixels $f_{\text{ground}}(v) = \alpha(v_{\text{horizon}} - v)$ is then linear and the same for all columns. For details, we refer to [12].

In practice, the MAP problem equation (4) is written as an energy *minimization* problem by turning the product over probabilities into a sum of negative log probabilities, which is then solved efficiently through Dynamic Programming (DP) [12], [32]. DP will efficiently minimize the energy function

$$E(L_u) = \sum_{n=1}^{N_u} E_p(s_{n-1}, s_n) + E_d(s_n) \quad (5)$$

for many stixel hypotheses $L_u = \{s_1, \ldots, s_{N_u}\}$, which consists of unary terms $E_d(s_n)$ and pairwise energy terms $E_p(s_{n-1}, s_n)$. Intuitively, the unary energy term $E_d(s_n)$ describes the disparity deviation of the disparity models described above. The pairwise term for $n = 1$ reads $E_p(s_0, s_1)$ and is a special case since s_0 is not defined. In all other cases, this pairwise term only evaluates the plausibility of a given stixel segmentation. Note that this in particular means that this pairwise term is independent of the disparity data. We have omitted these details here for simplicity [32].

2) Semantic Stixels: The Semantic Stixels method [1] introduced an additional semantic data term to associate each stixel with one class label $l_n \in \{1, \ldots, C\}$. Thus, Semantic Stixels are characterized by $s_n = (v_n^b, v_n^t, c_n, f_n(v), l_n)$. First, a semantic segmentation CNN is trained on RGB images with annotated per-pixel class labels. Then, when testing on a test image, the softmax outputs $\sigma(p, l)$ for all semantic classes l of all pixels p are kept (note that in a standard semantic segmentation task, only the class label of the strongest softmax output would be kept). The unary data term $E_d(s_n)$ of the original disparity stixel computation is then replaced by $E_u(s_n) = E_d(s_n) + \omega_l E_l(s_n)$, thereby adding semantic information from the network activations,

$$E_l(s_n) = -\sum_{p \in \mathcal{P}_n} \log \sigma(p, l_n). \quad (6)$$

Here $\mathcal{P}_n$ are all pixels in stixel s_n, and ω_l a weight factor.

B. Instance Stixels

Instance Stixels expand the idea of Semantic Stixels by additionally training a CNN to output a 2D estimation of the position of the instance center for each pixel. This estimation is predicted in image coordinates, as proposed in [28], [29]. More specifically, the CNN predicts 2D offsets $\mathbf{\Omega}_p \in \mathbb{R}^2$ (i.e., x and y direction) per pixel, which are relative to the pixel's location in the image. As a consequence, for all pixels p belonging to the same instance j, adding their ground truth offset $\hat{\mathbf{\Omega}}_p$ to the pixel location (x_p, y_p) will result in the same instance center location

$$\hat{\boldsymbol{\mu}}_j = \hat{\mathbf{\Omega}}_p + (x_p, y_p). \quad (7)$$

We refer to such a network as the *Offset CNN* and an example of its output is visualized in Fig. 2. The ground truth instance centers are defined as the center of mass of the ground truth instance masks. Note that instances are commonly only considered for certain semantic classes, e.g., cars, pedestrians and bicycles. Let $\mathcal{I} \subset \mathbb{N}$ denote said set of instance relevant classes. For all other classes, the target offset is (0, 0).

Instance Stixels incorporate the Offset CNN prediction into the stixel computation. Let $\boldsymbol{\mu}_p$ denote the instance center estimate obtained from the CNN for some pixel p, and $\bar{\boldsymbol{\mu}}_n = \sum_{p \in \mathcal{P}_n} \boldsymbol{\mu}_p$ the mean over all pixels in an instance stixel $s_n = (v_n^b, v_n^t, c_n, f_n(v), l_n, \bar{\boldsymbol{\mu}}_n)$. We model the instance term depending on the center estimates of the pixels and the mean instance center of the current stixel hypothesis s_n:

$$E_i(s_n) = \begin{cases} \sum_{p \in \mathcal{P}_n} ||\boldsymbol{\mu}_p - \bar{\boldsymbol{\mu}}_n||_2^2, & \text{if } l_n \in \mathcal{I} \\ \sum_{p \in \mathcal{P}_n} ||\boldsymbol{\mu}_p - (x_p, y_p)||_2^2, & \text{otherwise.} \end{cases} \quad (8)$$

In other words, for instance classes, the instance term favors stixels which combine pixels that consistently point to the same instance center. For non-instance classes, i.e., $l_n \notin \mathcal{I}$, offsets $\mathbf{\Omega}_p = \boldsymbol{\mu}_p - (x_p, y_p)$ deviating from zero contribute to the instance energy term. Without this, classes with instance information would generally have higher energy and thus be less likely than the non-instance classes.

With the instance energy term, the unary energy becomes

$$E_u(s_n) = \omega_d E_d(s_n) + \omega_s E_s(s_n) + \omega_i E_i(s_n). \quad (9)$$

This also introduces weights ω_d and ω_i for the disparity and instance terms for more control on the segmentation.

A useful side effect is that each instance stixel receives a mean estimate of its instance center pixel coordinates, which will be used when clustering stixels into objects, discussed in Section III-C.

C. Clustering Stixels With Instance Information

We now describe how output from an Offset CNN can be used in a post-processing step to cluster stixels. Note the favorable computational complexity of grouping a low number of stixels rather than individual pixels as in conventional instance segmentation tasks, e.g., 2000 stixels vs. 1.4 M pixels.

First, the per-pixel offsets from the Offset CNN are aggregated into a per-stixel offset estimate by averaging the CNN's predictions over the pixels in the stixel (this is already done for Instance Stixels, as noted in Section III-B). Hence, each stixel is equipped with an estimate of its instance center in 2D image coordinates, as well as a semantic class label.

Then, the estimated instance centers and semantic class prediction are used to group stixels to form instances. Separately

for each semantic class, we aim to find clusters in the estimated instance centers. Note that this condition on the semantic class also qualifies Instance Stixels for panoptic segmentation. The final clustering is done using the DBSCAN algorithm [34] as it estimates the number of clusters (i.e., instances) and performs well when the data has dense clusters. DBSCAN has only two parameters: the maximum distance between neighboring data points ε and the minimum size, as in cardinality, γ of the neighborhood of a data point in order to consider this point a core point. Additionally, we introduce a size filter parameter which prevents stixels that are smaller (i.e., cover less rows) than ρ to be considered a core point. This modification prevents small stixels, which lie on the border of two instances, to merge those instances together. Nevertheless, they are assigned to one of those adjacent instances during the clustering procedure.

D. Unary Regularization

The original Stixel MAP formulation considers a prior term $p(L_u)$ equation (4) which models pairwise interactions of vertically adjacent stixels. The prior term contains detailed models of the expected segmentation. For example, it models the probability of a ground stixel to be found below a sky stixel and vice versa. In the end, the modelled probabilities are usually estimated heuristically.

At the same time, this prior term acts as a regularizer. Without this regularization effect, the resulting stixels tend to be very small simply to fit the data terms as well as possible. In an extreme case with stixels of a width of 1 pixel, this would lead to stixels of also height equal to 1 pixel, which means in the end that each stixel corresponds to a single pixel. Consequently, the stixel segmentation would not be any more compact than the pixel-wise representation.

We argue that this modeling of pairwise interactions is especially useful for disparity-based stixels, since there more detailed semantic information is missing. Instance Stixels however do extract semantic and instance information from the RGB images and thus this modeling may be unnecessary. Therefore, we propose to replace this prior term by a simple *unary regularization* term

$$E_p(s_n) = \frac{w_R}{v_n^t + 1 - v_n^b} \tag{10}$$

which penalizes small stixels. The regularization constant w_R is the only parameter that needs to be determined and is comparable to the different weighting factors of the data terms.

IV. Implementation

We provide an open source Instance Stixels implementation which has been optimized for computational performance on the Cityscapes dataset [35]. As input it requires the RGB and disparity image of a scene and outputs a set of stixels comprising information about 3D position, semantic class and instance label. Note that in general, Instance Stixels may also operate only on the RGB image without relying on an disparity image and as a result do not compute the depth of a stixel.

The first step in the Instance Stixel pipeline as depicted in Fig. 2 is the CNN which predicts for each pixel the probability of each semantic class and the 2D instance center offset vectors in pixels. On Cityscapes this results in an output depth of $19 + 2 = 21$ channels in total. Any standard semantic segmentation network architecture could be used as the basis for the Semantic Segmentation and Offset CNN by increasing the output depth by 2 channels and training those to predict instance offset vectors. In our implementation, we use Dilated Residual Networks [24] (DRN) as our underlying architecture due to their favorable properties for these tasks, as discussed in Section II. Furthermore, we exploit the fact that, unlike the general method presented in that paper, our implementation is computes stixels of a fixed width of 8 pixels and remove any upsampling layers in the DRN architecture. The implementation of the DRN is largely based on the PyTorch [36] code provided by the authors of [24]. In order to optimize CNN inference for efficiency, we make use of mixed precision capabilities of NVIDIA Volta GPUs using the Apex utilities [37] without loss of accuracy.

The second step in the pipeline consists of the actual stixel computation. For this purpose, we extended the open-source disparity Stixel CUDA-implementation introduced in [12]. Amongst other features, such as the computation of Semantic Stixels according to [1] and handling of invalid disparity measurements, our extension comprises the Instance Stixels presented here. Techniques to optimize for efficiency, such as the use of prefix sums (aka. cumulative sums), have been adapted and reused from the original implementation. [12] provides a detailed explanation of those ideas.

Lastly, the stixels are clustered based on the mean instance center estimate. To this end, we utilize the GPU-based DBSCAN implementation of cuML [38] and customize it to include the size filter ρ described in Section III-C.

In summary, all components are implemented on the GPU which reduces the effective number of required host-device copy operations to two, namely copying the RGB and disparity images to device memory and retrieving the resulting stixel segmentation from device memory. The source code of our implementation is available online.[1]

V. Experiments

A. Dataset, Metrics and Pre-Processing

The computation of stixels require a RGB camera image and the corresponding disparity image obtained from a stereo camera setup. We use the Cityscapes dataset [35] for our experiments, as it consists of challenging traffic scenarios. Further, it provides ground truth annotations for semantic and instance segmentation. The performance on these two tasks is evaluated using the standard Cityscapes metrics [35].

Semantic segmentation performance is measured by the Intersection-over-Union (IoU) $= \frac{TP}{TP+FP+FN}$, where TP, FP, and FN denote the number of true positives, false positives and false negatives over all pixels in the dataset split. An instance mask is considered correct if the overlap with its ground truth

[1]Code available at https://github.com/tudelft-iv/instance-stixels

mask surpasses a specific threshold. The Average Precision (AP) corresponds to an average over the precision for multiple thresholds. Average Precision ($AP^{50\%}$) only considers an overlap of at least 50% as true positive. The metric also allows to provide confidence score for each instance mask. We did not make use of this option and always set the confidence score to 1 for all compared algorithms.

The disparity images provided in the Cityscapes dataset exhibit noisy regions introduced due to bad disparity measurements at the vertical image edges and the hood of the car. Inaccurate disparity data may harm the performance of disparity based Stixels. Although Semantic Stixels are already more robust due to the second modality, we aim to suppress such effects. Therefore, we crop all images symmetrically (top: 120px, bottom: 120px, left: 128px, right: 128px) to ensure that our experiments are not influenced by disparity errors. Following [1], we are using the official validation as test set. Therefore, we split the official training set into a separate training *subtrain* and validation set *subtrainval* (validation cities: Hanover, Krefeld, Stuttgart).

B. Training the CNN

The CNN takes an RGB image as input and predicts the semantic class probabilities and two channels for the offset vectors. Thus, it is a single CNN that provides the output of the Semantic Segmentation and Offset CNN, which were discussed separately in Section III. For training, we construct a loss that allows us to steer the focus between consistency and accuracy of the prediction. Here, we consider a prediction consistent when all pixels of a ground truth instance mask point towards the *same* 2D position, i.e., all predictions for the instance center equation (7) are the same. Offset accuracy is directly measured by the deviation of each single pixel from the center of mass of the ground truth instance mask. We argue that, for the predicted offsets, consistency is more important than accuracy. This is best illustrated by an example: consider a single instance in an image and all predicted offsets of that instance do not point to the center of mass of the instance, but instead to a different single point. As a result, this prediction would be consistent, as all offsets point to the same point, and at the same inaccurate as that point does not match the ground truth instance mask's center of mass. Despite the fact that this single point is not the training target, the clustering on this inaccurate, but consistent prediction would work perfectly since all the pixels of the instance are mapped to a single point and thus form a distinct cluster. This observation holds for both the instance-aware stixel computation and the clustering. Nevertheless, enforcing a certain degree of accuracy avoids trivial solutions such as all pixel offsets in the image point to the same single point which would render clustering impossible.

Let $j \in \mathcal{J}$ denote all ground truth instance masks in an image, $\mathcal{P}_j$ all pixels of that mask and $\mathcal{P}_B$ all background pixels which are not part of any instance mask. For all pixels p the CNN predicts an offset $\mathbf{\Omega}_p$ and using equation (7) the predicted center $\boldsymbol{\mu}_p$ can be computed. Further, $\bar{\boldsymbol{\mu}}_j = \frac{1}{|\mathcal{P}_j|} \sum_{p \in \mathcal{P}_j} \boldsymbol{\mu}_p$ denotes the corresponding mean of the predicted centers and $\hat{\boldsymbol{\mu}}_j$ the center of mass of the ground truth instance mask. Our offset loss

$$\mathcal{L}_{\mathcal{O}} = \sum_{j \in \mathcal{J}} \left(\frac{\alpha_a}{|\mathcal{P}_j|_1} \sum_{p \in \mathcal{P}_j} ||\boldsymbol{\mu}_p - \hat{\boldsymbol{\mu}}_j||_1 + \frac{\alpha_c}{|\mathcal{P}_j|} \sum_{p \in \mathcal{P}_j} (\boldsymbol{\mu}_p - \bar{\boldsymbol{\mu}}_j)^2 \right) + \frac{\alpha_a}{|\mathcal{P}_B|} \sum_{p \in \mathcal{P}_B} ||\mathbf{\Omega}_p||_1 \tag{11}$$

thus comprises a consistency term based on $\bar{\boldsymbol{\mu}}_j$, an accuracy term based on $\hat{\boldsymbol{\mu}}_j$ and a background term. The weights α_a and α_c provide the means to find a favorable trade-off between those terms. The full loss $\mathcal{L} = \mathcal{L}_O + \mathcal{L}_S$ further includes a semantic loss $\mathcal{L}_S$, namely a 2D cross-entropy semantic segmentation loss, on the the first 19 semantic output channels.

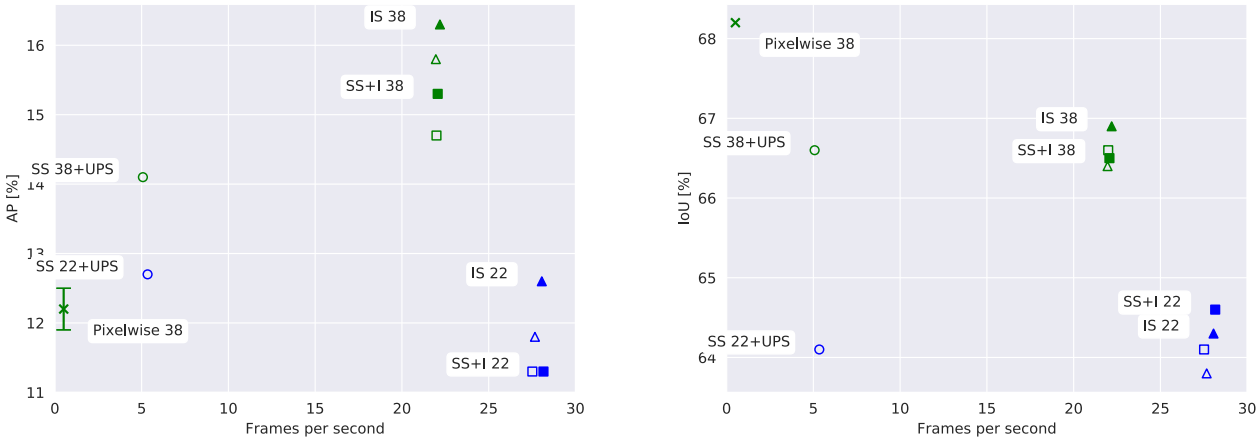
It is important to note that the output (*not* the input) of the CNN is downscaled by a factor 8. We also downscale the ground truth output by that factor for training. The reason for this is that upscaling, unless nearest neighbor upscaling is used, introduces interpolation errors that result in a smooth transition of the offset vectors between two instances. As a consequence, this would also result in an interpolation of the predicted means of two neighboring instances at pixels close to the borders, which in the end yields worse clustering results. To overcome this issue, we use nearest neighbor upscaling when passing the predicted images to the Stixel algorithms. The loss of resolution is compensated by the fact that our Stixels work at a resolution of width 8.

In practice, we found that training the *drn_d_38* architecture with $\alpha_a = \alpha_c = 1e-4$ and the *drn_d_22* architecture with $\alpha_a = 1e-5$ and $\alpha_c = 1e-4$ worked well. We minimize the loss function using the Adam optimization [39] (learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$). Further, we apply zero mean, unit variance normalization based on the training data to the input data and use horizontal flipping to augment our training data. The networks were trained for 500 epochs and with a batch size of 20 images. From these 500 epochs, we chose the best performing model for each architecture based on the semantic IoU on the validation set.

C. Hyperparameter Optimization

The stixel algorithms we evaluate offer several hyperparameters that require tuning: the weighting of the data terms for the Stixel computation ω_d, ω_s and ω_i, as well as the DBSCAN parameters ϵ, γ and ρ. The stixel framework provides more parameters from which we set the stixel width to 8 pixels throughout our experiments. Remaining parameters are set based on recommendations from [32] and [12]. For the *Pixelwise* baseline we only need to tune ϵ and γ. Additionally, in this baseline the large number of data points requires the clustering algorithm to process the data in batches which leads to non-deterministic results.

The parameter tuning is performed using Bayesian optimization [40] on the *subtrainval* validation set for 100 iterations. The score is computed as Semantic IoU $+1.5\cdot$ Instance AP. We weighted Instance AP higher as this is our main focus. The



(a) Instance performance vs. frames per second.

(b) Semantic performance vs. frames per second.

Fig. 3. Trade-off between segmentation performance and processing speed. Each data point represents the average performance of an algorithm on the Cityscapes validation set (all classes, cropped to 1792x784 pixels). The colors indicate different CNN architectures (drn_d_22 or drn_d_38), the symbols differentiate the base algorithm to obtain the instances (*triangle:* Instance Stixels, *square:* Semantic Stixels + Instance, *circle:* Semantic Stixels + UPSNet, *cross:* Pixelwise. If the symbol is filled with color, the unary regularization term was used instead of the pairwise energy term in the stixel computation (Section III-D).

optimization is performed separately for each algorithm unless noted otherwise.

D. Comparison of Algorithmic Variations

To analyze the capabilities of our proposed method, we vary four different aspects of computing stixels with instance information.

1) *Pixelwise:* In this baseline setup, the pipeline as shown in Fig. 2 is run entirely without stixels, by removing the *Stixel Computation*. The semantic class is determined according to the largest class probabilities. During the clustering step, pixels of the same semantic class are clustered based on their predicted instance centers.
2) *SS+UPS:* Represents the combination of state-of-the-art methods to augment stixels with instance information. Based on a separate instance segmentation method, a stixel is assigned to an instance by majority vote of the pixel-level prediction. For this purpose, we utilize the following state-of-the-art methods: a pretrained instance segmentation method called UPSNet [26] and Semantic Stixels [1]. On pixel-level, UPSNet achieves AP performance of 33.1% on the cropped validation set.
3) Semantic Stixels + Instance vs. Instance Stixels (*SS+I* vs. *IS*): Corresponds to setting $\omega_i = 0$, which resembles Semantic Stixels [1]), versus $\omega_i > 0$ in the stixels computation (see 9).
4) Pairwise vs. unary: Describes whether the stixel computation takes the pairwise term into account or instead regularizes the height of a stixel based on the unary regularization term as described in Section III-D.
5) *drn_d_22* vs. *drn_d_38:* Denotes the different base architectures of the Dilated Residual Network [24] used to predict semantic probabilities and instance offsets. The architecture *drn_d_38* is deeper and requires more memory.

Due to the fact that our *subtrainval* set overlaps with the training set of the UPSNet, we cannot use the *subtrainval* set for hyperparameter tuning. Hence, we use the same weights for the Semantic Stixels of *SS+UPS* as the corresponding *SS+I*.

1) Processing Speed Vs. Segmentation Performance: In the following we compare the different stixel methods for instance segmentation regarding the trade-off of segmentation performance and processing speed. The main indicators for segmentation performance are the instance AP and the semantic IoU as described in Section V-A. Processing speed is measured as the number of frames the pipeline can process per second. Here, to compute the frames per second we average the processing time of the frames in the validation set, which takes into account the processing time of all three modules (*CNN*, *Stixel Computation* and *Clustering*, see Fig. 2), but neglects data loading and visualization. All frames are processed sequentially on a NVIDIA Titan V GPU.

Fig. 3 illustrates the trade-off between processing speed and instance as well as semantic performance in a compact manner. Table I extends the figure by providing further segmentation metrics and also the complexity of the image representation as the average number of stixels per frame on the official Cityscapes validation set.

In terms of segmentation performance, the illustrations show that the choice of the network architecture of the CNN, indicated by the color of the points, has the most prominent effect (green: *drn_d_38* and blue: *drn_d_22*). For both segmentation metrics, even the best algorithm based on *drn_d_22* performing worse than the worst stixel algorithm based on *drn_d_38*. Within the same architecture however, Instance Stixels (*IS*) generally perform better than Semantic Stixels + Instance (*SS+I*) in terms of instance AP, but not always in terms of semantic IoU. Further, for both algorithms (*IS* and *SS+I*), using the unary regularization term (filled symbols) surpasses its pairwise counterpart (non-filled symbols) or at least remains on par. Interestingly, the CNN architecture choice also affects the comparison in instance AP of Instance Stixels and Semantic Stixels + UPSNet (*SS+UPS*). For *drn_d_22*, *IS 22* with unary regularization achieves similar instance AP as *SS+UPS 22*. For *drn_d_38*, *SS+UPS* obtains

TABLE I
PERFORMANCE OF THE PIXELWISE BASELINE AND DIFFERENT VARIATIONS OF STIXEL ALGORITHMS THAT PROVIDE INSTANCE SEGMENTATION (ROWS) WITH RESPECT TO VARIOUS METRICS (COLUMNS). RESULTS ARE COMPUTED ON THE CITYSCAPES VALIDATION SET (ALL CLASSES, CROPPED TO 1792X784 PIXELS). BEST RESULTS PER METRIC ARE HIGHLIGHTED IN **BOLDFACE**. * THE RESULTS OF THE PIXELWISE BASELINE WERE AVERAGED OVER THREE RUNS AND ARE REPORTED WITH THE CORRESPONDING STANDARD DEVIATION. ALL OTHER ALGORITHMS ARE CONSISTENT OVER MULTIPLE RUNS. ** THE RESULTING SEGMENTATION IS REPRESENTED AS $1792 \cdot 784 = 1\,404\,928$ PIXELS, SINCE NO STIXELS ARE INVOLVED HERE

	CNN	Unary regularization	AP [%]	AP^{50} [%]	IoU [%]	cat IoU [%]	FPS	Avg. number of stixels
Pixelwise	drn_d_38	-	12.5 ± 0.3*	25.3 ± 0.7*	**68.2**	**85.0**	0.5	1404928**
SS+I	drn_d_22	-	11.3	25.4	64.1	80.2	27.5	2095
SS+I	drn_d_22	✓	11.3	25.7	64.6	81.8	**28.2**	1765
SS+I	drn_d_38	-	14.7	30.3	66.6	80.8	22.0	**1270**
SS+I	drn_d_38	✓	15.3	31.6	66.5	81.3	22.1	4795
SS+UPS	drn_d_22	-	12.7	28.5	64.1	80.2	5.3	2095
SS+UPS	drn_d_38	-	14.1	30.8	66.6	80.8	5.1	**1270**
IS	drn_d_22	-	11.8	26.3	63.8	79.9	27.7	1384
IS	drn_d_22	✓	12.6	26.8	64.3	81.1	**28.1**	2673
IS	drn_d_38	-	15.8	31.1	66.4	80.5	22.0	1421
IS	drn_d_38	✓	**16.3**	**32.4**	66.9	81.9	22.2	2278

worst instance AP of all stixel methods. The semantic IoU of *SS+UPS* is limited by its *SS+I* counterpart by construction. Overall, Instance Stixels based on the *drn_d_38* architecture and using the unary regularization outperforms all other stixel-based algorithms in both segmentation metrics. Only the *Pixelwise* algorithm surpasses this performance in the semantic IoU, but not the instance AP. The same observations generally also hold for the extended instance and semantic segmentation metrics $AP^{50\%}$ and the category IoU [35] as listed in Table I.

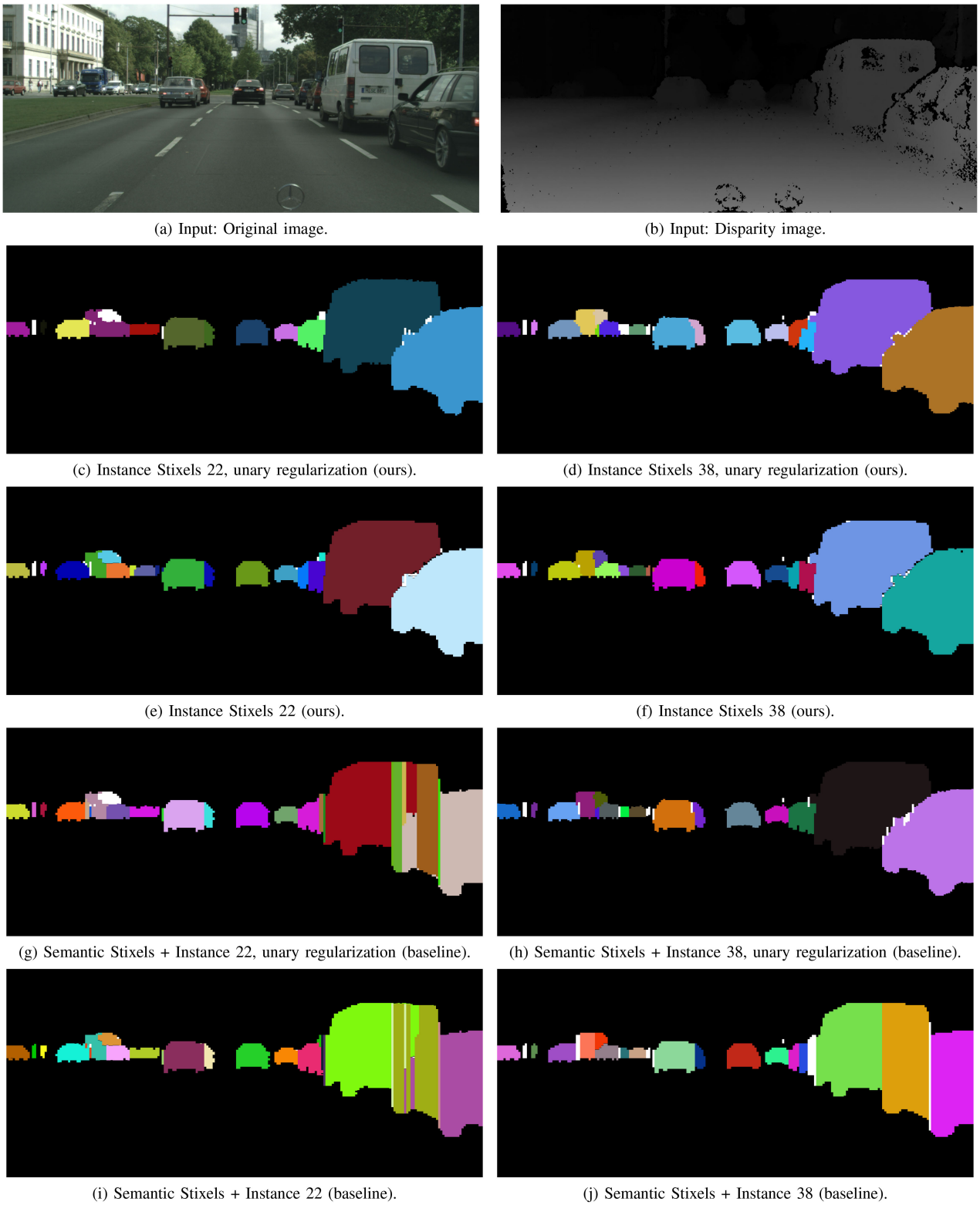
To a certain degree, segmentation performance comes at a trade-off regarding processing speed. Notably, the speed is mainly determined by the choice of the CNN as well. The *Pixelwise* pipeline is by far the slowest algorithm for these tasks at only 0.5 frames per second. Stixel methods based on *drn_d_38* are favorable compared to methods relying on UPSNet, but not as fast as methods based on *drn_d_22*. Among the same architecture the differences in processing speed are only minor and are listed in Table I. Additional analysis showed that the processing speed is steady over all frames, regardless of the number of instances or stixels in an image. The complexity of the segmentation, quantified by the average number of stixels per frame, varies between algorithms exhibiting no obvious correlation. Among the Instance Stixels the highest average number stixels per frame is at most 2673.

2) Qualitative Analysis: The consequences of the different algorithm variations as described in Section V-D are depicted in Fig. 4 when applied on an real traffic scene image from the *subtrainval* set. Figs. 4(a) and 4(b) show the input data. Instance segmentation results in the left column (4 4(c), 4(e), 4(g) and 4(i)) based on the drn_d_22 architecture show in general more errors than in the right column (4 4(d), 4(f), 4(h) and 4(j)) which is based on the drn_d_38 architecture. Especially Figs. 4(g), 4(i) and 4(j) show several stixels which overlap two instances.

Fig. 5 visualizes the full results (3D position, semantic and instance segmentation) of Instance Stixels (*drn_d_38*, unary regularization) on three scenes (columns). Based on the input RGB images (top row), the CNN predicts the offset vectors (center rows). The offset vectors are visualized in HSV color space, where the hue indicates the direction and the saturation the magnitude of the offsets. The fourth row shows the segmentation of the scenes. The overlaid colors illustrate the semantic class per pixel, whereas the white contours around objects mark the borders of instances. The bottom row shows top down views of the scene based on the per stixel disparity information and location within the input image. In these illustrations the road and the sidewalk are illustrated as polygons. Their boundaries are based on the ground plane estimation. Sky stixels are discarded and non-instance stixels are drawn as circles. Their radius indicates the size of the respective stixel in the image. Stixels of the same instance are connected by a line. Per column, we only connect the stixel that are closest to the ego-vehicle. As a result of our instance segmentation, we can also filter outliers. Specifically, we do not include stixels that are further than 3 meters away from the mean top down position of the instance. Also, we removed the stixel artefacts of the Mercedes-Benz Star from the top down view based on their position in the image.

VI. DISCUSSION

Results presented in Section V-D show that adding the instance term, that distinguishes *IS* and *SS+I*, increases instance AP. Minor drawbacks in terms of semantic IoU may be due to hyperparameter optimization which values instance AP more than semantic IoU. Despite the increased instance AP, the segmentation for far away objects (e.g., the truck in the left-hand part of Fig. 4(a)) and tightly overlapping objects (e.g., pedestrians in the left-hand part Fig. 5(a)) remain challenging. Overall, the choice of the CNN appears to be more important to the segmentation than the effect of the instance term. Notably, using a state-of-the-art pixelwise instance segmentation CNN, such as UPSNet [26], and combining it with Semantic Stixels falls behind significantly in terms of processing speed. UPSNet requires on average 0.15 seconds of processing time per frame which on its own yields only 6.6 frames per second. Compared to the pixelwise UPSNet result, the instance AP of *SS + UPSNet* has decreased by more than 50%. A drop in overall accuracy is likely, since the stixels group pixels along predefined coarse column borders and thus inherently decrease the granularity of the prediction. Further, *SS* do not consider the instance term introduced for *IS*, thus stixels may overlap two different instances. UPSNet cannot change this afterwards which leads



(a) Input: Original image.

(b) Input: Disparity image.

(c) Instance Stixels 22, unary regularization (ours).

(d) Instance Stixels 38, unary regularization (ours).

(e) Instance Stixels 22 (ours).

(f) Instance Stixels 38 (ours).

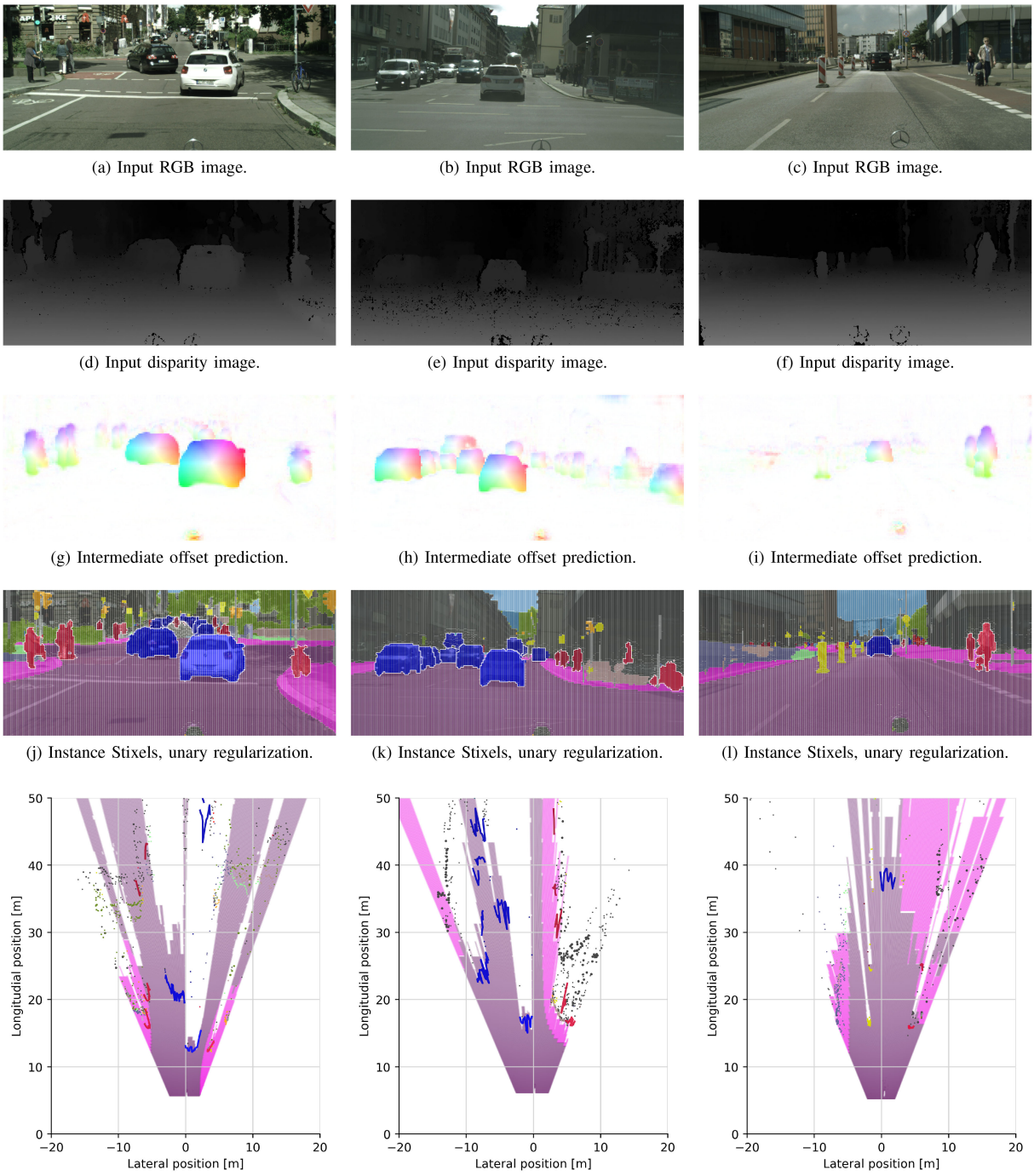
(g) Semantic Stixels + Instance 22, unary regularization (baseline).

(h) Semantic Stixels + Instance 38, unary regularization (baseline).

(i) Semantic Stixels + Instance 22 (baseline).

(j) Semantic Stixels + Instance 38 (baseline).

Fig. 4. Qualitative analysis of instance segmentation results from *Semantic Stixels + Instance* (baseline) and *Instance Stixels* (proposed algorithm) using different architectures as well as comparing the pairwise energy term and the unary regularization. Fig. 4(a) and 4(b): The input RGB and disparity image. Below, the left column shows instance segmentation results obtained using the drn_d_22 architecture as basis for the CNN. Likewise, the right column Instances are indicated by arbitrary colors. White areas denote stixels that cannot be assigned to specific instances, but their predicted semantic class is an instance class. Black areas indicate that the predicted semantic class is not an instance class.



(a) Input RGB image. (b) Input RGB image. (c) Input RGB image.

(d) Input disparity image. (e) Input disparity image. (f) Input disparity image.

(g) Intermediate offset prediction. (h) Intermediate offset prediction. (i) Intermediate offset prediction.

(j) Instance Stixels, unary regularization. (k) Instance Stixels, unary regularization. (l) Instance Stixels, unary regularization.

(m) Instance Stixels 38, unary regularization. (n) Instance Stixels 38, unary regularization. (o) Instance Stixels 38, unary regularization.

Fig. 5. Illustration of stixel segmentations including spatial top down view of the scene. Each column is a separate scene, the top two rows show the corresponding inputs, the center row shows the offsets predicted by the CNN and the bottom two rows visualize the output of Instance Stixels (*drn_d_38*, unary regularization). In the third row from the top, the overlaid color indicates the semantic class of a stixel, whereas the white contour around objects indicate the segmented instances. The last row shows the top view of the scene. Instances are visualized as lines, road and sidewalk stixels are plotted as polygons based on the obtained ground plane estimation. Stixels of class sky are discarded in this illustration. All remaining stixels (e.g., buildings and poles) as points and their radius indicates the stixels size.

to worse performance. Lastly, a pixelwise clustering approach shows weak instance segmentation performance at a runtime of 0.5 FPS that is dominated by the clustering algorithm suffering from the large amount of points.

The benefits of a purely stixels-based instance segmentation method however is not only observed in processing speed, but also in term of segmentation complexity. Pixelwise methods result in more than 1.4 million independent predictions. Our Instance Stixels on average require between 1384 and 2673 stixels per frame to describe the same amount of pixels. This means Instance Stixels reduce the complexity of the representation by factors between $525.6\times$ - $1015.1\times$.

Aside from image segmentation, Instance Stixels provide position estimates in 3D space. As a result, top down views of a scene can be extracted, similar to a grid map. In contrast to a grid map, our representation is continuous and does not discretize 3D space. In this top down representation, imperfect disparity measurements, become apparent, for example in that the back of cars do not appear as straight lines. Further, it also show the inaccuracies of the ground plane and horizon estimation, which is here based on v-disparity [33]. In the stixel model, stixels above the horizon cannot be classified as ground. This leads to artefacts as seen on the road behind the two cars in Fig. 5(j). As the ground plane estimation is crude, the polygons of the road stixels overlap sometimes with stixels of obstacles. Combining Instance Stixels with LiDAR measurements as shown in [41] may improve both, the depth estimation and the ground plane estimation. As this is an orthogonal approach, not related to instance segmentation, we made use of our object based representation to for example filter outliers in the depth measurements of a single object.

The rich information about both, the static and dynamic surrounding, contained in Instance Stixels can benefit subsequent utilization in an autonomous driving pipeline. For example, Instance Stixels provide a rich and efficient representation for path planning, object tracking, and mapping.

VII. Conclusions

This paper introduced Instance Stixels to improve stixel segmentation by considering instance information from a CNN, and performing a subsequent stixel clustering step. Our experiments showed multiple benefits of including the instance information already in the segmentation step, opposed to only clustering Semantic Stixels. First, quantitative and qualitative analysis show that Instance Stixels adhere better to object boundaries. Second, Instance Stixels provide more accurate instance segmentation than Semantic Stixels augmented with instance information from a pixel-level instance segmentation network. Third, Instance stixels still preserve the favorable stixel characteristics in terms of compactness of the segmentation representation (on average less than 2673 stixels per image) and computational efficiency (up to 28 FPS at a resolution of 1792x784). In future work, the integration of additional sensor modalities as shown in [41] and temporal information to enforce consistency are potential research directions.

Acknowledgment

The authors would like to thank the authors of [12] and [24] for kindly providing their code and pre-trained CNN models to the scientific community.

References

[1] L. Schneider *et al.*, "Semantic stixels: Depth is not enough," in *Proc. IEEE Intell. Veh. Symp.*, 2016, pp. 110–117.
[2] S. Sivaraman and M. M. Trivedi, "Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 4, pp. 1773–1795, Dec. 2013.
[3] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila, "Eurocity persons: A novel benchmark for person detection in traffic scenes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1844–1861, Aug. 2019.
[4] D. Nuss, T. Yuan, G. Krehl, M. Stübler, S. Reuter, and K. Dietmayer, "Fusion of laser and radar sensor data with a sequential Monte Carlo Bayesian occupancy filter," in *Proc. IEEE Intell. Veh. Symp.*, 2015, pp. 1074–1081.
[5] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1331–1342, Dec. 2011.
[6] H. Badino, U. Franke, and D. Pfeiffer, "The stixel world-a compact medium level representation of the 3d-world," in *Proc. 31st DAGM Symp. Pattern Recognit.*, Berlin, Heidelberg, Germany: Springer-Verlag, 2009, pp. 51–60.
[7] D. Hernandez-Juarez *et al.*, "Slanted stixels: Representing San Francisco's steepest streets," in *Proc. Brit. Mach. Vis. Conf. 2011*, Sep. 2017, pp. 87.1–87.12, doi: 10.5244/C.31.87.
[8] J. Ziegler *et al.*, "Making bertha drive - an autonomous journey on a historic route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, Summer 2014.
[9] W. Sanberg, G. Dubbelman, and P. de With, "From stixels to asteroids: Towards a collision warning system using stereo vision," in *Proc. IS&T Int. Symp. Electron. Imag.*, vol. 2019, no. 15, 2019, pp. 34-1–34-7, doi: 10.2352/ISSN.2470-1173.2019.15.AVM-034.
[10] S. Ramos, S. Gehrig, P. Pinggera, U. Franke, and C. Rother, "Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling," *IEEE Intell. Veh. Symp.*, 2017, pp. 1025–1032.
[11] D. Pfeiffer and U. Franke, "Towards a global optimal multi-layer stixel representation of dense 3D data," *Proc. British Mach. Vis. Conf.*, 2011, pp. 51. 1–51. 12.
[12] D. Hernandez-Juarez, A. Espinosa, J. C. Moure, D. Vázquez, and A. M. López, "GPU-accelerated real-time stixel computation," *IEEE Winter Conf. Appl. Comput. Vis.*, 2017, pp. 1054–1062.
[13] M. Cordts, T. Rehfeld, M. Enzweiler, U. Franke, and S. Roth, "Tree-structured models for efficient multi-cue scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1444–1454, Jul. 2017.
[14] N. H. Saleem, H. Chien, M. Rezaei, and R. Klette, "Effects of ground manifold modeling on the accuracy of stixel calculations," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, pp. 3675–3687, Oct. 2019.
[15] D. Pfeiffer and U. Franke, "Efficient representation of traffic scenes by means of dynamic stixels," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2010, pp. 217–224.
[16] B. Günyel, R. Benenson, R. Timofte, and L. Van Gool, "Stixels motion estimation without optical flow computation," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Heidelberg, Germany: Springer, 2012, pp. 528–539.
[17] F. Erbs, A. Barth, and U. Franke, "Moving vehicle detection by optimal segmentation of the dynamic stixel world," in *Proc. IEEE Intell. Veh. Symp.*, 2011, pp. 951–956.
[18] F. Erbs, B. Schwarz, and U. Franke, "Stixmentation-probabilistic stixel based traffic scene labeling." in *Proc. Brit. Mach. Vis. Conf.*, 2012, pp. 1–12.
[19] F. Erbs, B. Schwarz, and U. Franke, "From stixels to objects—A conditional random field based approach," in *Proc. IEEE Intell. Veh. Symp.*, 2013, pp. 586–591.
[20] F. Erbs, A. Witte, T. Scharwächter, R. Mester, and U. Franke, "Spider-based stixel object segmentation," in *Proc. IEEE Intell. Veh. Symp.*, 2014, pp. 906–911.
[21] M. Cordts, L. Schneider, M. Enzweiler, U. Franke, and S. Roth, "Object-level priors for stixel generation," in *Proc. German Conf. Pattern Recognit.*, 2014, pp. 172–183.
[22] T. Scharwächter and U. Franke, "Low-level fusion of color, texture and depth for robust road scene understanding," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2015, pp. 599–604.
[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[24] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 472–480.
[25] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollar, "Panoptic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 9396–9405.
[26] Y. Xiong *et al.*, "Upsnet: A unified panoptic segmentation network," in *Comput. Vis. Pattern Recognit.*, 2019, pp. 8818–8826.
[27] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," in *Proc. Deep Learn. Robot. Vis., Workshop Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1–2.
[28] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7482–7491.

[29] J. Uhrig, E. Rehder, B. Fröhlich, U. Franke, and T. Brox, "Box2pix: Single-shot instance segmentation by assigning pixels to object boxes," in *Proc. IEEE Intell. Veh. Symp.*, 2018, pp. 292–299.
[30] D. Neven, B. D. Brabandere, M. Proesmans, and L. V. Gool, "Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 8829–8837.
[31] T. M. Hehn, J. F. P. Kooij, and D. M. Gavrila, "Instance stixels: Segmenting and grouping stixels into objects," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2019, pp. 2542–2549.
[32] D. Pfeiffer, "The stixel world," Ph.D. dissertation, Mathematisch-Naturwissenschaftliche Fakultät II, Humboldt-Univ., Berlin, Germany, 2012.
[33] R. Labayrade, D. Aubert, and J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through v-disparity representation," in *Proc. IEEE Intell. Veh. Symp.*, 2002, pp. 646–651.
[34] M. Ester *et al.* "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discov. Data Mining*, pp. 226–231, 1996.
[35] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 3213–3223.
[36] A. Paszke *et al.*, *Advances in Neural Information Processing Systems 32*, Red Hook, NY, USA: Curran Associates, Inc., 2019, pp. 8024–8035, [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[37] NVIDIA Corporation, Apex: A pytorch extension: Tools for easy mixed precision and distributed training in pytorch, (visited on: 2019–11-26), [Online]. Available: https://github.com/NVIDIA/apex
[38] R. D. Team, *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. [Online]. Available: https://rapids.ai
[39] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic gradient descent," in *Proc. ICLR: Int. Conf. Learn. Representations*, 2015.
[40] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.
[41] F. Piewak, P. Pinggera, M. Enzweiler, D. Pfeiffer, and M. Zöllner, "Improved semantic stixels via multimodal sensor fusion," in *Proc. German Conf. Pattern Recognit.*, 2018, pp. 447–458.

Thomas Hehn received the bachelor's and master's degrees in physics from Heidelberg University, Heidelberg, Germany, in 2014 and 2017, respectively. He is currently working toward the Ph.D. degree with the Delft University of Technology, Delft, The Netherlands. His research focuses on computer vision for autonomous driving. In 2018, he was the recipient of the Best Paper Award of the German Conference on Pattern Recognition for his research on decision tree algorithms done at the Heidelberg Collaboratory for Image Processing.

Julian Kooij (Member, IEEE) received the Ph.D. degree in 2015 in artificial intelligence from the University of Amsterdam, Amsterdam, The Netherlands, where he worked on unsupervised machine learning and predictive models of pedestrian behavior. In 2013, he was with Daimler AG, Stuttgart, Germany, on path prediction of vulnerable road users for highly-automated vehicles. In 2014, he joined the Computer Vision Lab, Delft University of Technology, Delft, The Netherlands, and since 2016, he has been an Assistant Professor with the Intelligent Vehicles Group, part of the Cognitive Robotics Department with the same university. His research interests include developing novel probabilistic models and machine learning techniques to infer and anticipate critical traffic situations from multimodal sensor data.

Dariu Gavrila (Member, IEEE) received the Ph.D. degree in computer science from the University of Maryland, College Park, MD, USA, in 1996. In 1997, he was with Daimler R&D, Ulm, Germany, where he became a Distinguished Scientist. In 2016, he moved to the Delft University of Technology, Delft, The Netherlands, where he since heads the Intelligent Vehicles Group, as a Full Professor. His research interests include sensor-based detection of humans and analysis of behavior, recently in the context of the self-driving cars in urban traffic. He was the recipient of the Outstanding Application Award 2014 and the Outstanding Researcher Award 2019, both from the IEEE Intelligent Transportation Systems Society.