This CVPR workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

DPPD: Deformable Polar Polygon Object Detection

Yang Zheng Oles Andrienko Yonglei Zhao Minwoo Park Trung Pham
NVIDIA
{yazheng, oandrienko, yongleiz, minwoop, trungp}@nvidia.com

Abstract

Regular object detection methods output rectangle bounding boxes, which are unable to accurately describe the actual object shapes. Instance segmentation methods output pixel-level labels, which are computationally expensive for real-time applications. Therefore, a polygon representation is needed to achieve precise shape alignment, while retaining low computation cost. We develop a novel Deformable Polar Polygon Object Detection method (DPPD) to detect objects in polygon shapes. In particular, our network predicts, for each object, a sparse set of flexible vertices to construct the polygon, where each vertex is represented by a pair of angle and distance in the Polar coordinate system. To enable training, both ground truth and predicted polygons are densely resampled to have the same number of vertices with equal-spaced raypoints. The resampling operation is fully differentable, allowing gradient back-propagation. Sparse polygon predicton ensures high-speed runtime inference while dense resampling allows the network to learn object shapes with high precision. The polygon detection head is established on top of an anchor-free and NMS-free network architecture. DPPD has been demonstrated successfully in various object detection tasks for autonomous driving such as traffic-sign, crosswalk, vehicle and pedestrian objects.

1. Introduction

Object detection, as one of the most popular computer vision tasks, typically predicts objects in rectangle bounding boxes. Boxes are able to describe locations and sizes, but not object shapes. Fig. 1 shows an example of crosswalk detection for autonomous driving, where precise crosswalk regions are needed. This can be achieved by an instance segmentation method which outputs a pixel-wise mask per object. However, pixel-level post-processing is computationally expensive, thus not suitable for real-time applications.

Alternatively, a method that detects objects as polygons is a better choice because 1) polygons can capture object shapes with high accuracy, and 2) a detection network can be run much faster than a segmentation network plus post-processing. Unfortunately, polygon detection is more complex than box detection in the variant numbers of vertices of arbitrary shapes. This creates difficulties when training a network to predict a fixed number of vertices. A common solution (*e.g.*, as done in previous methods such as PolarMask [25]) is that ground truth polygons are represented in Polar coordinates and approximated by a vector of distance values and a (fixed) vector of evenly-spacing angular values. The task becomes training a network to regress, for each object, a radius vector, together with the predefined uniformly emitted rays decoded back to polygon. A clear limitation of this method is that the quality of ground truth labels (and thus the quality of prediction) is bounded by the number of rays. Fig 2 shows that even with 64 rays, crosswalk regions are not well captured. Increasing the numbers of rays will increase the computational cost significantly.

In this work, we propose a novel *Deformable Polar Polygon Object Detection* method, namely *DPPD*. Unlike Polar-

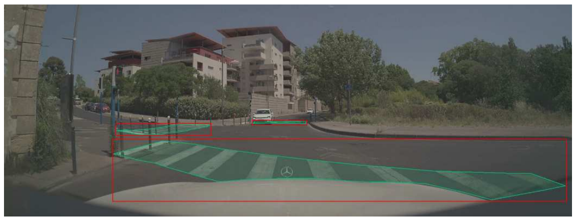


Figure 1. For crosswalk detection, polygon shapes are in green; bounding boxes are in red. It is clear that bounding boxes are unable to represent the crosswalk regions well as compared to polygons.

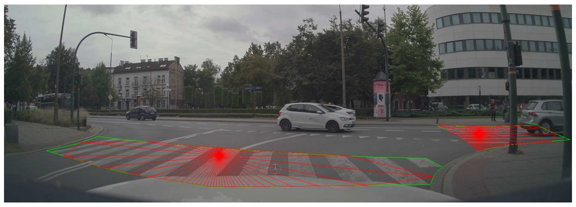


Figure 2. Example of approximating ground truth polygons (green) using polar polygons (red) with fixed angular bins. Notice that 64 rays is still insufficient to capture the actual crosswalk regions.

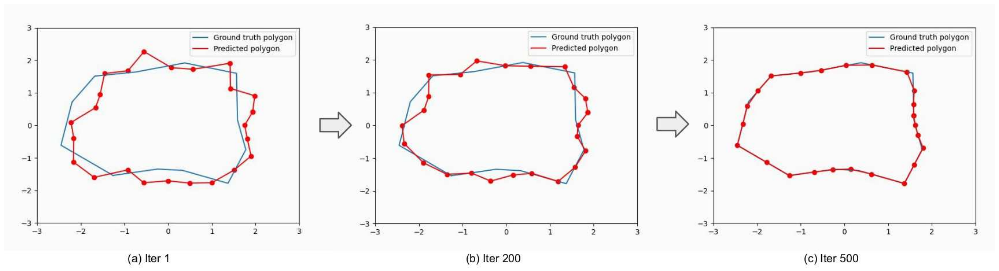


Figure 3. An illustration of the deformable polygon shape learning process. The target shape is in blue, and the predicted shape with 24 vertices is in red. From iteration step 1 to 200 to 500, the predicted shape is converging close to the target.

Mask, our network predicts a small set of flexible polygon vertices directly, where each vertex has two degrees of freedom in Polar coordinates, *i.e*., radius and angle. Intuitively, starting from an initialized polygon, the network will deform it until it aligns well with the ground truth polygon (see Fig. 3). A question arises is how to compare and compute loss between ground truth and prediction where they are different in the number of vertices. Our idea is to densely resample both ground truth and prediction with a certain number of rays. We could resample as many rays as the memory allows. It's worth emphasizing that the resampling process only happens at the loss computation, thus does not affect the inference runtime, and that the resampling operation is differentiable, allowing gradient based optimization.

Note that DPPD is designed for polygon shape regression, and it is applicable for any successful object detection architecture. In this work, we establish a high-speed, single-shot, anchor-free, NMS-free detection network structure based on set prediction [20, 27] and build DPPD on top of it. This simple architecture escalates both training effectiveness and inference efficiency.

The main contributions of this work can be summarized as follows: 1) We propose a novel polygon detection method to detect objects with arbitrary shapes. Our method avoids a trade-off decision between network computation cost and polygon accuracy as in PolarMask; 2) a method to decode a regression vector into a valid polygon (*e.g*., vertices are in counter-clockwise order), a batching processing algorithm to resample sparse polygons to dense polygons with a minimum cost; 3) Inspired by the latest object detection development, we design a highly efficient single-stage, anchor-free and NMS-free polygon detection architecture; 4) Our proposed polygon detector surpasses previous polygon detection methods in both speed and accuracy when tested on autonomous driving perception tasks such as crosswalk, road sign, vehicle and pedestrian detection.

2. Related Work

Object Detection. Object detection has been evoluted from two-stage or one-stage anchor-based detectors [15, 16, 19], to anchor-free and NMS-free detectors [6, 23, 28], transformer-based detectors [3], and other variations [1, 9, 14, 17, 18]. In general, an object detector includes three major components: a backbone of series of convolutional blocks, a neck of multi-resolution feature pyramids [13], and a detection head. The detection head is usually divided into a classification head and a bounding box regression head. Our detection method follows the anchor-free, NMS-free approach, but the box regression head is replaced by a polygon regression head to predict object locations and boundaries.

Instance Segmentation. Instance segmentation produces pixel-level class-ids and object-ids. Mask R-CNN [8] introduces a detect-then-segment approach to breakdown this problem into two sequential sub-tasks. To overcome the expensive two-stage processing, YOLOACT [2] and its extended methods [4, 22, 24] construct a parallel assembling framework, by generating a set of prototype masks and predicting per-instance mask coefficients. However, regardless of model variations, the per-pixel segmentation mask output is always a huge burden for downstream real-time applications.

Polygon Detection. A series of work, such as ExtremeNet [29], ESE-Seg [26], PolarMask [25], and FourierNet [21], try to parameterize the contour of an object mask into fixed-length coefficients, given different decomposition bases. These methods predict the center of each object and the contour shape with respect to that center. PolarMask is the one closest to our method, which is built on top of FCOS [23] and utilizes depth-variant rays at constant angle intervals. Similarly, PolyYOLO [10] adopts the YOLOv3 [18] architecture, and modifies the perpendicular grid into circular sectors to detect polar coordinates of polygon vertices. Each circular sector is responsible to produce 1 or 0 vertex. The drawback of these methods is that the shape alignment quality is heavily bottlenecked by the pre-defined ray bases. Increasing the number of rays is possible to improve the quality, but meanwhile downgrading the speed performance. In contrast, our method is less dependent on the number of vertices. We found that as small as 12 vertices is sufficient to model variety of object shapes.

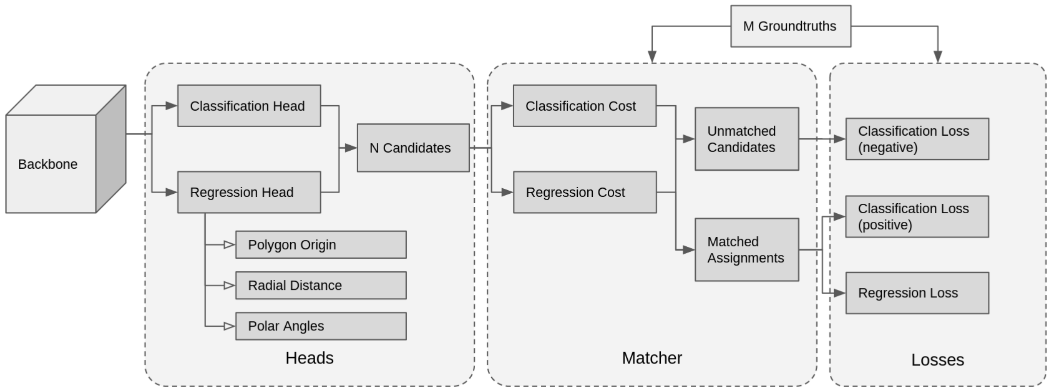


Figure 4. Overall network architecture and training pipeline. We establish a classification head and a regression head to generate N candidates. The N candidates are matched with M groundtruths, resulting in M pairs of prediction-target assignments. Based on the assignment, we compute positive classification and regression losses.

Active Contour Model (ACM). In the classical computer vision area, the active contour model [11] has been used to describe object shape boundaries. The main idea is to minimize an dedicated internal and external energy functions. The external term is to control the contour shape fitting, and the internal term is to control the deformation continuity. Our DPPD is inspired by the ACM. The training loss jointly minimizes the shape fitting error between ground truth and prediction polygons and polygon smoothness.

3. Method

In this section, we first introduce the overall set prediction network architecture. We then describe the polygon detection head. And lastly, we discuss the training strategies.

3.1. Object Detection as Set Prediction

We adopt an anchor-free and NMS-free set prediction approach for our DPPD object detector due to its simplicity and efficiency. The network predicts a set of N candidates ($N \gg M$ number of ground truth objects). The N candidates are matched with M ground truth labels using a Hungarian matching algorithm, resulting in M pairs of prediction-target assignments. Classification and regression losses from these matches are computed to supervise the training. For unmatched candidates, only classification loss is computed.

Fig. 4 depicts the high-level network architecture and training pipeline. Followed by the network backbone and feature pyramid is a classification head and a regression head. The regression head predicts polygon origins (*i.e.*, object center) and vertices (*i.e.*, radial distance and polar angles). One grid cell in the feature map is responsible for detecting one polygon candidate.

3.2. Polygon Regression

3.2.1 Polar Representation

In the polar coordinates, each polygon is represented as one origin (object center) and k pairs of radial distances and polar angles. Distances and angles are defined w.r.t the object center. The network will output a $(2 + 2 * k)$-vector, where 2 values are for the polygon origin and $2 * k$ values are for k vertices. The contouring vertices, in the polar representation, are convenient to be organized a clockwise or counterclockwise order.

3.2.2 Polygon Decoding

The decoding process parses a regression output vector $[f_0, f_1, ..., f_{2*k+2}]$ to the corresponding polygon origin coordinates, radial distances, and polar angles, denoting as $[o_x, o_y, r_0, ..., r_{k-1}, a_0, ..., a_{k-1}]$.

Polygon origin In the fully convolutional set prediction framework, every grid cell at the feature map yields a candidate. To get the accurate location, we predict offsets w.r.t. the grid cell position. Formally, the polygon origin is decoded as:

$$\begin{cases} o_x = g_x + s_x * \sigma(f_0) \\ o_y = g_y + s_y * \sigma(f_1) \end{cases} \tag{1}$$

where (o_x, o_y) denote the polygon origin coordinates; (g_x, g_y) denote the grid cell coordinates; (s_x, s_y) denote the grid cell size; σ is a sigmoid activation function.

Radial distances The next k regression outputs $(f_2, ..., f_{k+2})$ are dedicated for k radial distances. The decoding function is:

$$r_i = \mu * e^{f_i}, i \in [2, k+2] \tag{2}$$

where μ is a prior knowledge of the radius scale. We apply an exponential activation to ensure the decoded radius is always positive.

Polar angles The last k output channels $(f_{k+2}, ..., f_{2*k+2})$ are responsible for the k polar angles. We predict angle deltas between adjacent vertices, and then decode using cumulative sum before normalizing them into $[0, 2\pi]$ range:

$$a_i = 2\pi * \frac{\sum_{j=k+2}^{i} e^{f_j}}{\sum_{j=k+2}^{2*k+2} e^{f_j}}, i \in [k+2, 2*k+2] \tag{3}$$

It can be seen clearly that unlike the previous methods such as PolarMask [25], which predefined polar angles (e.g., by shooting uniform rays from 0 to 360 degrees), our method predicts both polar angles and radial distances. This allows the network to deform the initial polygons as much as needed to match the ground truth polygons. In contrast, previous methods find difficulties to fit well the ground truth shapes unless a dense number of rays (*e.g.*, 360) is used.

Note that the angle decoder also differentiates DPPD against Poly-YOLO [10]. Poly-YOLO splits the polar coordinates into circular sectors where each sector is responsible for either 1 (exist) or 0 (non-exist) vertex. This design is unable to discriminate clustered vertices that fall into the same sector. Instead, DPPD predicts angle deltas, which could be small and large to handle arbitrary intervals.

3.3. Training

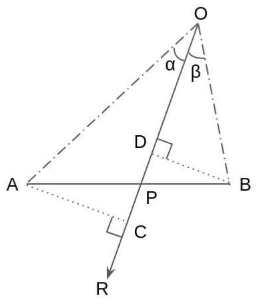
3.3.1 Ground Truths

Objects are often annotated using polygons. However, there is no consistent way to enforce all the objects having the same number of vertices and vertex distributions along the boundaries. Therefore, it is less reasonable to train a network to predict polygons with a fixed number of vertices and by directly comparing ground truth and predicted vertices as often done in the box regression problem.

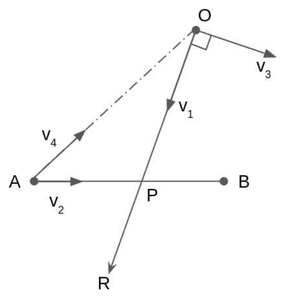
Instead, we propose a simple method to tackle this issue, in which both ground truth and prediction polygons are resampled to have the same number of points, namely *raypoints*. Bear in mind that this resampling process only happens during training, thus does not hinder inference latency. This simple idea seems overlooked in the past mainly because the resampling operation might be not efficient, and not differentiable for SGD based training. Below we will describe two resampling methods which are both efficient and differentiable.

3.3.2 Polygon Resampling

Given a polygon with k vertices, we want to upsample it with m ($m > k$) raypoints along equally distributed polar angles. This resampling process is a geometry problem, *i.e.*, to find intersections between k boundary segments and



(a) Triangle approach.



(b) Vector approach.

Figure 5. O is the polygon center, A and B are two adjacent vertices, $\vec{OR}$ is one ray emitted from O. The goal is to find the intersection point P between the segment AB and ray $\vec{OR}$. Refer to Sec. 3.3.2 for detailed derivation.

m rays. To simplify the computation, we assume the polygon is translated to its origin at $(0, 0)$. In polar coordinates, since rays are uniformly emitted with the same angle interval, the resampling output only consists a m dimensional radial distance vector $[r_0, r_1, ..., r_{m-1}]$. We provide two approaches to tackle this problem, triangle approach and vector approach.

Triangle Approach. In Fig. 5a, let A and B be two adjacent vertices and O be the origin. The task is to find the intersection point P between segment AB and ray $\vec{OR}$. The norm $|\cdot|$ notation is used for the segment length. Deriving from the triangle similarity between $\triangle ACP$ and $\triangle BDP$:

$$w = \frac{|AP|}{|BP|} = \frac{|AC|}{|BD|} = \frac{|OA|\sin(\alpha)}{|OB|\sin(\beta)}. \tag{4}$$

we compute a length ratio w between $|AP|$ and $|AB|$. Then the point P coordinates are calculated as:

$$(P_x, P_y) = (\frac{A_x + wB_x}{1+w}, \frac{A_y + wB_y}{1+w}). \tag{5}$$

The coordinates of O, A, B and their segment lengths are known from the decoding outputs. $\vec{OR}$ is one of the m equal-spaced resampling rays. Given a list of polygon vertices with their sorted polar angles, we can easily find, for each ray, a pair of neighboring vertices A, B via linear search so that the ray locates between $\vec{OA}$ and $\vec{OB}$.

Vector Approach. In Fig. 5b, for each known segment AB and ray $\vec{OR}$, let define the following 4 vectors:

$$\begin{cases} \vec{v_1} = (\vec{OR}_x, \vec{OR}_y) \\ \vec{v_2} = (B_x - A_x, B_y - A_y) \\ \vec{v_3} = (-\vec{OR}_y, \vec{OR}_x) \\ \vec{v_4} = (O_x - A_x, O_y - A_y) \end{cases} \tag{6}$$

Since P is the intersection point by $\vec{OR}$ and AB, we formulate the vector representation of $\vec{OP}$ and $\vec{AP}$:

$$\begin{cases} \vec{OP} = O + t_1\vec{v_1}, t_1 \in [0, \infty) \\ \vec{AP} = A + t_2\vec{v_2}, t_2 \in [0, 1] \end{cases} \quad (7)$$

where t_1 and t_2 are the fractional scale of units along $\vec{OR}$ and $\vec{AB}$. Finding the intersection is to solve t_1 and t_2 from Eq. (7). We constrain $t_1 \in [0, \infty)$ to ensure P is along the positive direction of ray $\vec{OR}$, and $t_2 \in [0, 1]$ to ensure P is inside segment AB. The mathematical solution is:

$$\begin{cases} t_1 = \dfrac{|\vec{v_2} \times \vec{v_4}|}{\vec{v_2} \cdot \vec{v_3}} \\ t_2 = \dfrac{\vec{v_4} \cdot \vec{v_3}}{\vec{v_2} \cdot \vec{v_3}} \end{cases} \quad (8)$$

where the operator $\cdot$ is the dot product and $\times$ is the cross product. Suppose $\vec{v_1}$ is decomposed into $[v_{1x}, v_{1y}]$, combining Eq. (7), the coordinates of P is written as:

$$(P_x, P_y) = (O_x + t_1 v_{1x}, O_y + t_1 v_{1y}). \quad (9)$$

Note that the triangle approach assumes vertices have been ordered ascendingly. For predictions, we decode polar angles in counter-clockwise order, which naturally satisfies the assumption. However, for ground truth encoding, this is not guaranteed for concave shapes. Therefore, the triangle approach is only used for prediction decoding. On the other hand, the vector approach has no requirements on the vertex order, and it works for both convex and concave shapes. However, the cross-product calculation expense much memory in the practical implementation, therefor vector approach is only used for ground truth encoding.

3.3.3 Polygon Regression Losses

A common way to measure the loss between two shapes is based on their intersection-over-union (IoU). For general polygons, there is no exact closed-form solution. Fortunately, with the polar representation and the above resampling strategy, computing losses between ground truth and prediction polygons becomes easier. Formally, the polygon shape regression loss $\mathcal{L}_{reg}$ is a weighted sum of three components: polygon origin loss $\mathcal{L}_o$, polar IoU loss $\mathcal{L}_{iou}$ and internal smoothness loss $\mathcal{L}_{sm}$:

$$\mathcal{L}_{reg} = w_1\mathcal{L}_o + w_2\mathcal{L}_{iou} + w_3\mathcal{L}_{sm} \quad (10)$$

Polygon origin loss. The origin loss $\mathcal{L}_o$ measures the difference between two polygons' centers. We employ smooth-l_1 loss (l_1^s) for the absolute distance error. Also the losses are normalized by the ground truth size. Formally, let $[\hat{x}_o, \hat{y}_o]$ be the ground truth center, $[\hat{w}, \hat{h}]$ be the polygon width and height, $[x_o, y_o]$ be the prediction center, the origin loss is computed as:

$$\mathcal{L}_o = \frac{l_1^s(o_x, \hat{o}_x)}{\hat{w}} + \frac{l_1^s(o_y, \hat{o}_y)}{\hat{h}} \quad (11)$$

Polar IoU loss. The polar IoU loss $\mathcal{L}_{iou}$ measures shape difference between the two polygons regardless of their locations. Let $[\hat{r}_0, \hat{r}_1, ..., \hat{r}_{m-1}]$ and $[r_0, r_1, ..., r_{m-1}]$ be the ground truth and prediction radial distances respectively, the polar IoU loss is computed as:

$$\mathcal{L}_{iou} = \log \frac{\sum_{i=0}^{m-1} \max(r_i, \hat{r_i})}{\sum_{i=0}^{m-1} \min(r_i, \hat{r_i})} \quad (12)$$

Smoothness loss. The smoothness loss is added to reduce the shape oscillation, similar to the internal energy of the classical active contour model [11]. Let $[r_0, r_1, ..., r_{m-1}]$ be the prediction radial distances, $d^1 r$ and $d^2 r$ be the 1st and 2nd ordered differences, the smoothness loss is computed as:

$$\mathcal{L}_{sm} = \frac{\sum_{i=1}^{m-1} d^1 r_i}{m-1} + \frac{\sum_{i=1}^{m-1} d^2 r_i}{m-1} \quad (13)$$

4. Experiment

We conduct experiments using two datasets: our in-house autonomous driving dataset, and the public Cityscapes [5] dataset. Since one major application of the polygon detector is for autonomous driving perception, we use our in-house dataset for the primary investigation. We focus on road-sign and crosswalk detections where accurate object boundaries are important for subsequent tasks such as localization, sign recognition. We examine the effectiveness of polygon detectors against bounding box detectors, and thoroughly compare DPPD with PolarMask [25] in terms of both accuracy and speed performance. To further demonstrate the generic effectiveness, we benchmark DPPD results on Cityscapes, which covers more dynamic instances like vehicles and pedestrians. We also explore ablation studies for the model design.

4.1. DPPD on Internal Datasets

The in-house dataset statistics for road sign and crosswalk are summarized in Table 1, where all labels are given as polygon vertices. In this experiment, we establish DPPD polygon head on top of a FPN augmented AlexNet network backbone. The number of channels for different CNN blocks are 32, 128, 256, 512 respectively. The input image size is 960×480. The feature maps at stride 8 and 32 are used for the crosswalk and road-sign tasks separately.

4.1.1 Polygon vs Bounding Box

We first demonstrate the polygon detector is substitute for the bounding box detector. Typically, road signs are detected

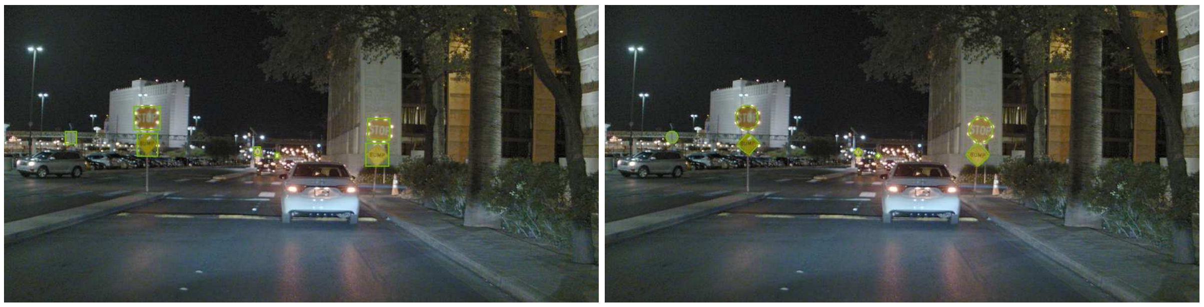


Figure 6. Two examples to visualize bounding box (left) and DPPD polygon (right) detections on traffic-signs. The polygon detector is better to capture the object shape.

Table 1. Number of instances for traffic-sign and crosswalk polygon detection datasets.

Task	Training	Evaluation
Road sign	377.4K	39.6K
Crosswalk	260.2K	36.1K

as bounding boxes. To swap it with a polygon model, we ensure that the new polygon model has non-negative effects on both detection accuracy and inference latency.

In this experiment, we set the number of predictable vertices $k = 12$, and the number of resampling rays $m = 180$. To compare the polygon against the box model, we convert polygons to their envelop bounding boxes, and evaluate both detectors based on the box metrics. Results are listed in Table 2. It shows that the polygon model is on-par with the box model with a very slight regression, although the polygon model was not trained to detect boxes. Nevertheless, the polygon model produces tighter alignments to the objects, which are qualitatively visualized in Fig. 6.

The speed performance is device-dependent. We measure the runtime inference latency (in ms) when the model is deployed to TensorRT and run in FP16 precision on two hardware platforms: 2080 Titan GPU, and NVIDIA DRIVE Orin chip. The polygon detector runs $0.09ms$ slower on GPU, while $0.13ms$ faster on the chip. We consider the inference time is comparable on both platforms. This is expected because only the last regression layer is changed, whereas the change of regression channels is minor w.r.t. the total number of model parameters.

4.1.2 DPPD vs PolarMask

Next we emphasize the benefits of DPPD against the state-of-the-art polygon detectors. We pick PolarMask as our competitor, and use the crosswalk detection task for this experiment. In DPPD, we set the number of prediction vertices $k = 36$, and the number of resampling rays $m = 360$. In PolarMask, we experimented different number of rays,

Table 2. Traffic-sign results. Box vs. polygon detector. Accuracy metrics precision (P), recall (R), F1-score ($F1$) are in percentages. The speed is measured as latency times on Titan GPU and Orin Chip devices, in ms.

Detector	P	R	$F1$	GPU	Chip
BBox	74.91	58.68	65.81	1.54	3.18
Polygon	74.54	57.31	64.80	1.63	3.05

Table 3. Crosswalk results. PolarMask vs. DPPD. The accuracy is evaluated based on polygon IoU directly. The speed is measured as latency times on GPU and Chip devices, in ms.

Detector	P	R	$F1$	GPU	Chip
PolarMask-36	61.58	38.00	47.00	-	-
PolarMask-64	66.72	45.46	54.08	1.91	3.24
DPPD-36	77.27	50.46	61.05	1.28	2.68

i.e., 36 and 64. Since crosswalk shapes are more complex, the matching criteria is based on polygon-to-polygon IoU directly. Metrics results and visualizations are shown in Table 3 and Fig. 7 respectively.

In terms of accuracy, DPPD is superior than PolarMask in all metrics, even the number of predictable vertices is less in DPPD (36) than PolarMask (64). The first reason is that PolarMask ground truth are approximated radius length along pre-defined rays, which are not guaranteed to reach ”real” vertices (*e.g.*, Fig. 2). The second reason is that DPPD predicts, for each vertex, both radial distance and polar angle, which allows greater capability to achieve high-quality shape regression.

In terms of the speed, DPPD also costs less inference time than PolarMask on both platforms. We claim it is benefitted from the design of separate training and inference features. Dense polygons for training facilitates the accuracy, and sparse polygon for inference boosts the runtime speed.

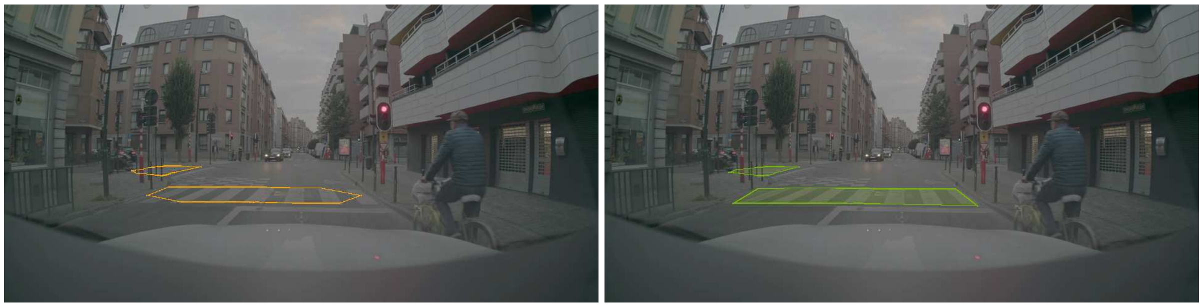


Figure 7. Two examples to visualize 64-vertex PolarMask (left) and 36-vertex DPPD (right) results on crosswalks. DPPD expenses less vertices but achieves tighter alignment with the groundtruth.

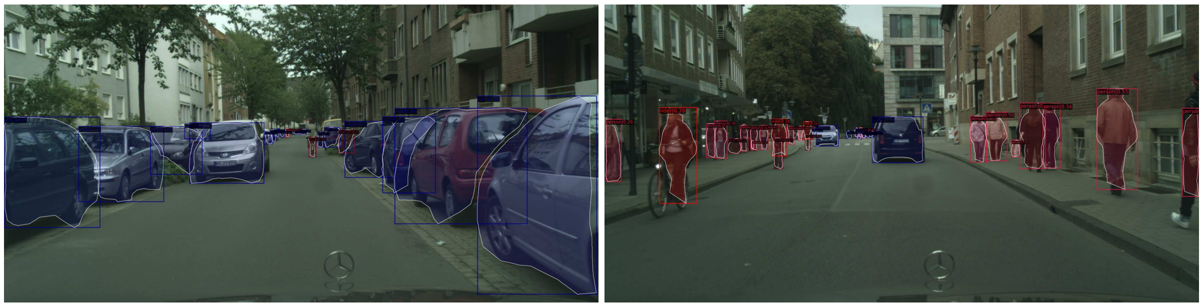


Figure 8. Qualitative visualizations of DPPD results on Cityscapes.

4.2. DPPD on Public Dataset

To examine the generic effectiveness, we experiment DPPD on the public Cityscapes [5] dataset, which covers more dynamic objects like vehicles, pedestians, bicycles, *etc*. We pick Poly-YOLO (an improved version of PolarMask and YOLOv3) as our main competitor. As the Poly-YOLO used DarkNet-53 as the backbone, we select Resnet50 as the backbone for our DPPD for a fair comparison. Input resolution is 1024×2048. we set the number of prediction vertices $k = 12$, and the number of resampling rays $m = 360$. Results of DPPD is based on the official Cityscapes evaluation metrics at the instance level, tested on the validation split. Results of other methods are borrowed from the Poly-YOLO paper [10].

As shown in Table 4, for the accuracy, DPPD surpasses the state-of-the-art polygon method Poly-YOLO, even using a smaller number of vertices predicted (12 vs. 24). Qualitative visualization are shown in Fig. 8. For the speed, DPPD runs at 56.1 FPS, measured on a Tesla V100 GPU. Since it is hard to reproduce competitors latency from the exact same environment, we mark it with a * symbol to indicate reference purpose only.

Poly-YOLO is built as an extension of YOLOv3. Comparing with DPPD, there are many fundamental differences such as the backbone structure, target assignment mechanisms, loss functions, *etc*. Considering the polygon head itself, due to the fact of circular grid splits, Poly-YOLO is still learned from approximated labels. Poly-YOLO is trained for boxes and polygons jointly, and it is claimed benefited from the auxiliary task learning. However, it predicts the object center for both box and polygon, which is not guaranteed well aligned (see Fig. 9). On the other hand, DPPD is a polygon-only detector. The resampling process allows DPPD to learn from the real polygon shape without any approximation, the angle decoding method enables its capability to fit for both sparse and dense vertices, and the object center encoded as the geometry centroid.

Table 4. Benchmark on Cityscapes. Poly-YOLO predicts 24 vertices, whereas DPPD predicts 12 vertices. YOLOv3 and Poly-YOLO use DarkNet53 backbone, whereas MaskRCNN and DPPD use ResNet50 backbone.

Method	Detector	AP	AP_{50}	FPS
YOLOv3 [18]	Box	10.6	26.6	26.3
MaskRCNN [8]	Mask	16.4	31.8	6.2
Poly-YOLO [10] (24)	Polygon	8.7	24.0	21.9
DPPD (12)	Polygon	11.96	27.31	56.1*

4.3. Ablation Studies

Admittedly, the optimization of model structure, data augmentation, and other fantastic training strategies could further improve the overall performance. But in this sub-

Table 5. Ablation studies for DPPD design choices.

Exp.	Dataset	Origin Finding	Angle Decoder	# of Vertices	# of Rays	P	R	$F1$	AP	AP_{50}
1	Crosswalk	box center	bin offsets	36	360	69.19	50.22	58.20	-	-
2	Crosswalk	vertices mean	bin offsets	36	360	50.13	49.89	50.01	-	-
3	Crosswalk	geo centroid	bin offsets	36	360	72.02	54.54	62.08	-	-
4	Cityscapes	geo centroid	bin offsets	18	360	-	-	-	11.10	25.80
5	Cityscapes	geo centroid	cumsum	18	360	-	-	-	12.01	27.45
6	Cityscapes	geo centroid	cumsum	12	120	-	-	-	10.38	24.17
7	Cityscapes	geo centroid	cumsum	12	180	-	-	-	11.54	27.29
8	Cityscapes	geo centroid	cumsum	12	360	-	-	-	11.96	27.31
9	Cityscapes	geo centroid	cumsum	18	360	-	-	-	12.01	27.45
10	Cityscapes	geo centroid	cumsum	36	360	-	-	-	11.67	26.83

section, we focus on the key components involved in the polygon detector regression head. Specifically, we discuss the design choices of polygon origin finding methods, angle decoding methods, number of prediction vertices, number of resampling rays.

Polar Origin Finding. The polygon polar origin holds two regression targets. To find the groundtruth, we considered three methods: (i) the mean of all vertices; (ii) the bounding box center that covers the polygon; (iii) the polygon shape geometry centroid. However, (i) and (ii) are unable to guarantee that the polar origin is located within the polygon boundary (see Fig. 9), which voids the model effectiveness. Therefore, (iii) geometry centroid is the only choice. We run experiments on a subset of crosswalk objects and compared the three methods, shown as Exp. 1-3 in Table 5.

Angle Decoding. The flexible angle prediction is one major distinction of DPPD. Comparing against PolarMask [25], this is the second degree of freedom of each vertex; comparing against Poly-YOLO [10], it breaks the angle sector constraint. Using the Cityscapes data, we experiment two angle decoding methods: (i) predict an offset within each angle bin; (ii) the angle cumulative summation. According to Exp. 4-5 in Table 5, (ii) obtains higher accuracy than (i).

Predictable Vertices and Resampling Rays. The number of predictable vertices and resampling rays are important hyperparameters to construct the DPPD head. We experiment six combinations of vertices and rays on Cityscapes. Results are listed in Table 5. Increasing the number of rays (Exp.6 - 8) results in the accuracy improvement, this is expected since we have denser representation of the polygon shape. On the other hand, increasing the number of vertices (Exp.8 - 10) is less effective. Keep increasing it will lead to excessive model complexity and hence lower generalization capability. But this is a good indicator that DPPD is able to achieve promising results even with less regression outputs.

5. Conclusion

In this paper, we present DPPD, a deformable polygon detector that stands intermediately between object detection and instance segmentation. The polygon detector is able to describe precise object shape information, while retaining fast runtime inference speed. Our polygon detection method is able to predict object shapes with high accuacy without a need to use an excessively large number of vertices as done in the previous methods. This is possible due to our novel polygon training strategy, in which both ground truths and predictions (not necessarily having the same number of vertices) are up-sampled so that both have the same number of raypoints. The upsampling (resampling) is efficient and differentiable, which is required for training. From the experiments using both in-house autonomous driving dataset and public dataset, DPPD outperforms PolarMask and Poly-YOLO in terms of both accuracy and speed for many detection tasks such as road sign, crosswalk, car, pedestrian.

Although DPPD is designed as a 2D polygon detector, it is also applicable in 3D perception. Many state-of-the-art 3D detectors adopt BEV [12] or rangeview [7] representations for runtime efficiency. We could place DPPD on top of any detection architecture for complex shape understanding.

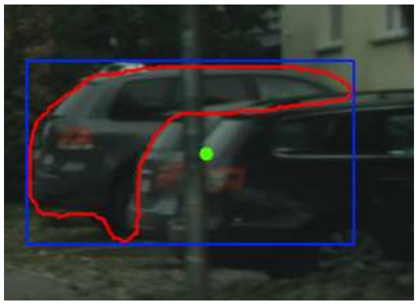


Figure 9. Example of polygon polar origin finding. If the vehicle object is partially occluded, the bounding box center (green) is outside of the polygon shape boundary (red).

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 2

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019. 2

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[4] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. Blendmask: Top-down meets bottom-up for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8573–8581, 2020. 2

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Scharwächter, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset. In *CVPR Workshop on the Future of Datasets in Vision*, volume 2. sn, 2015. 5, 7

[6] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6569–6578, 2019. 2

[7] Lue Fan, Xuan Xiong, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. Rangedet: In defense of range view for lidar-based 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2918–2927, 2021. 8

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 2, 7

[9] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510. IEEE, 2018. 2

[10] Petr Hurtik, Vojtech Molek, Jan Hula, Marek Vajgl, Pavel Vlasanek, and Tomas Nejezchleba. Poly-yolo: higher speed, more precise detection and instance segmentation for yolov3. *Neural Computing and Applications*, 34(10):8275–8290, 2022. 2, 4, 7, 8

[11] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988. 3, 5

[12] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*, pages 1–18. Springer, 2022. 8

[13] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 2

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 2

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2

[17] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 2

[18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 2, 7

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 2

[20] Hamid Rezatofighi, Tianyu Zhu, Roman Kaskman, Farbod T Motlagh, Javen Qinfeng Shi, Anton Milan, Daniel Cremers, Laura Leal-Taixé, and Ian Reid. Learn to predict sets using feed-forward neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):9011–9025, 2021. 2

[21] Hamd Ul Moqeet Riaz, Nuri Benbarka, and Andreas Zell. Fouriernet: Compact mask representation for instance segmentation using differentiable shape decoders. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 7833–7840. IEEE, 2021. 2

[22] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. In *European conference on computer vision*, pages 282–298. Springer, 2020. 2

[23] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019. 2

[24] Yuqing Wang, Zhaoliang Xu, Hao Shen, Baoshan Cheng, and Lirong Yang. Centermask: single shot instance segmentation with point representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9313–9321, 2020. 2

[25] Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Xuebo Liu, Ding Liang, Chunhua Shen, and Ping Luo. Polarmask: Single shot instance segmentation with polar representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12193–12202, 2020. 1, 2, 4, 5, 8

[26] Wenqiang Xu, Haiyang Wang, Fubo Qi, and Cewu Lu. Explicit shape encoding for real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5168–5177, 2019. 2

[27] Yan Zhang, Jonathon Hare, and Adam Prugel-Bennett. Deep set prediction networks. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[28] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 2

[29] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krahenbuhl. Bottom-up object detection by grouping extreme and center points. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 850–859, 2019. 2