Synthetic Training Data Generation and Domain Randomization for Object Detection in the Formula Student Driverless Framework

1st Rebecca Adam*  
Centre for Industrial Electronics, DME  
University of Southern Denmark  
Sønderborg, Denmark  
https://orcid.org/0000-0003-1080-1330

2nd Paulius Janciauskas  
Centre for Industrial Electronics, DME  
University of Southern Denmark  
Sønderborg, Denmark

3rd Thomas Ebel  
Centre for Industrial Electronics, DME  
University of Southern Denmark  
Sønderborg, Denmark  
https://orcid.org/0000-0001-8473-4471

4th Jost Adam*  
SDU Centre for Photonics Engineering, MCI  
University of Southern Denmark  
Odense, Denmark  
https://orcid.org/0000-0001-7177-3252

Abstract—Today industries strive toward using data-driven machine learning wherever applicable. Consequently, they require manually or automatically labeled training data sets. Currently, synthetically generating labeled training data sets belongs to the open challenges in machine learning across multiple application fields. In this paper, we propose employing a procedural pipeline combining BlenderProc with domain randomization to create prelabeled training data sets synthetically. Randomizing the domain using uncorrelated random background images, we ensure that the neural network applied for object detection purely learns the object features and is background-independent. Our proposed pipeline yields a solution to create sizeable prelabeled training data sets. We assess the pipeline performance for the application of cone object detection for the formula student driverless competition using no real training and a small real-world training data set for fine-tuning: We show that using the synthetically generated training data fine-tuned with a limited real training data set performs best for object detection. This transfer learning-based, fine-tuned solution also outperforms the benchmark training data set in detecting knocked-over cones that are neither present in the real nor the synthetic training data set. Consequently, by combining BlenderProc and domain randomization, we provide a solution for formula student teams to generate extensive training data for cone detection and other detection problems relevant to driverless.

Index Terms—synthetic training data, domain randomization, BlenderProc, driverless, formula student, transfer learning

I. INTRODUCTION

The awareness that recording and labeling vast amounts of data are necessary to employ machine learning is pervasive. It raises the question of how to replace manual labeling of categorical data features for machine learning or reduce the required training data sizes. Therefore, methods for synthetic training data are gaining popularity and attention across multiple application areas. Specifically, the field of autonomous driving requires object detection and hence desires labeled data sets. Student competitions like the formula student competition raise students’ interest in the field of driverless and constitute ideal development grounds for future driverless systems.

A. Related Work

1) BlenderProc: BlenderProc is a tool developed by the German aerospace center to synthetically create and render objects in scenes [1]. This tool builds upon the Blender software. Falling under the GNU General Public License (GPL) and allowing end-users to use, share and modify GPL software for commercial and educational purposes, Blender is a free, open-source 3D creation suite. It includes modeling, rendering, simulation, and compositing [2] and comprises a Python API.

2) Synthetic Data Generation and Domain Randomization: Generative Adversarial Networks are suitable for generating synthetic training data [3]. This method relies on utilizing a small number of real-world training images. Alternatively, data-independent approaches use pipelines to simulate or craft the complete training data as in [4]. The authors of [5, 6] assess synthetic data generation for car detection, employing the partly freely available Unreal Engine, usually used for gaming applications. Similar to [6], we propose and investigate a pipeline to generate synthetic data employing domain randomization (DR). Here, we use the combination of BlenderProc and DR instead of investigating the combination of the Unreal Engine, and DR. With this distinction to [6], we
C. Novel Contributions

For simplistic multi-colored cones, open-access databases like COCO [7] lack training data sets. Student driverless task realm has not been investigated, and synthetic data generation pipeline for cone detection (the formula student driverless task realm) has not been investigated, and open-access databases like COCO [7] lack training data sets for simplistic multi-colored cones.

B. Open Challenges

Open source freely available pipelines to create synthetic training data for object detection belong to the open challenges in detection problems. In particular, investigating a novel synthetic data generation pipeline for cone detection (the formula student driverless task realm) has not been investigated, and open-access databases like COCO [7] lack training data sets for simplistic multi-colored cones.

C. Novel Contributions

- We propose an open-source freely available pipeline for synthetic training data generation combining BlenderProc and domain randomization.
- We provide and assess a synthetically generated data set for cone detection for the formula student driverless.

II. PROCEDURAL PIPELINE FOR SYNTHETIC DATA

In this work, we propose a procedural pipeline for synthetic training data generation outlined in Fig. 1.

A. Blender and BlenderProc:

We created the objects, blue, yellow, and orange cones, via Blender. The subsequent scene generation includes randomizing parameters for the object (here cones), the camera, and the lighting:
- Object cone: Color (blue, yellow and orange), numbers, position, and vertical Rotation
- Camera: Location, and Rotation
- Color resolution for rendering

To employ domain randomization, we set backgrounds for the renderer to be transparent. Then we render each scene twice using different camera and lighting parameters. Finally, BlenderProc annotates in COCO format [7].

B. Domain Randomization

Domain randomization was first introduced in [8] and conceptually targets domain-uncorrelated object feature learning as explained in [9]. The domain randomization purpose is to create synthetically trained models that generalize to real-world data by a sufficient degree of variation [10], [11]. A data set’s quality and quantity strongly influence the trained network’s prediction consistency. As previously mentioned, building a data set that fulfills these requirements is usually challenging because it requires labeling, a costly process, especially regarding segmentation. To tackle this issue, Trembly et al. [6] build an artificial data set based on artificially generated images, which contain the specific features the network requires to track, randomly combined with other features that can be considered as noise. While previous methods tried to generate images that recreate photo-realistic scenes artificially, domain randomization purposely creates images that do not search for fidelity. It forces the network to learn the essential object features of interest, bringing more consistency to the results. This process potentially leads to substantial improvements in terms of network efficiency, enabling better performances and recognition capabilities. Moreover, domain randomization highly facilitates the possibility of generating artificial training data, removing the necessity to mimic complicated realistic scenes. While the authors of [6] use the method to track cars in images and compare the performance of synthetically generated training data in randomized domains to real-world training data, the technique also finds its way into medical imaging, e.g., to segment cardiac intervention X-ray data, and train a network to track a tool in lungs’ airways [9], [7]. Here, we present the combination of artificially created objects (cones) with random, non-scenic background images: BlenderProc renders the cones background-free, and we subsequently superimpose the resulting images on randomly chosen background images (Fig. 2). To reduce the probability of incorrectly detected objects in case ground truth objects are missing in an image (false positive scenario), we also include domain randomized background images without objects, or cones, in them.

III. OBJECT DETECTION APPLICATION: CONES

A. Cone Detection for Formula Student

The driverless formula student competition requires that students build a fully autonomous race car. According to the competition guidelines, the vehicle should be able to detect different cone objects (orange, blue, yellow) and navigate through them as fast as possible. The orange and blue cones have a white stripe in the middle, and the yellow cones have a black stripe in the middle. In addition to the orange, blue, and yellow small cones, larger orange cones with two instead of a single white stripe define the race-track starting position and finish line. Following the FSOCO [12] principles, these larger orange cones are not labeled and detected as another class. Nevertheless, they introduce a particular challenge for the orange small cone detection.

Although cone detection belongs to the comparably easy detection tasks due to its simplistic shape, currently, open training databases like COCO lack labeled cone images for training neural networks.

B. Bounding Boxes for Object Detection

For all $M$ objects, Cartesian $x$, $y$ positions, box width $w$, and height $h$ define a bounding box $m \in \{1, \ldots, M\}$. To decide for two overlapping bounding boxes if the overlap is large enough to classify as a correct detection, we introduce the intersection over union (IoU). Let $N$ and $K$ denote the number of images and the number of classes used for object detection, respectively. Let $b_{gt} = [x, y, w, h]$ and $b_p = [\hat{x}, \hat{y}, \hat{w}, \hat{h}]$ denote
Fig. 1: The procedural Pipeline uses domain randomization together with BlenderProc to create labeled objects in a random background setting synthetically.

Fig. 2: Examples of labeled, synthetic images with random backgrounds (not showing annotations).

Fig. 3: Example of labeled real-world data.

ground truth and the predicted object bounding box areas, respectively. Similar as proposed in [13], [14] and applied in [6] we measure the accuracy of correct object detections by measuring the detections’ number of

- True positives (TP): A correctly predicted ground truth bounding box.
- False positives (FP): An incorrectly detected object; a matching ground truth object is missing.
- False negatives (FN): An existing but not predicted ground truth object.

In each image $n \in \{1, \ldots, N\}$ and in each class $k \in \{1, \ldots, K\}$ the number of TP, FP and FN detections is denoted by $TP_{k,n}$, $FP_{k,n}$, and $FN_{k,n}$, respectively. Furthermore, we categorize a correct detection for an object $m \in \{1, \ldots, M\}$ by requiring that the overlapping area, the intersection over union $\text{IoU}_m$ (Fig. 4)

$$\text{IoU}_m = \frac{\text{area}\{\hat{b}_{p} \cap b_{gt}\}}{\text{area}\{\hat{b}_{p} \cup b_{gt}\}}$$

exceeds a given threshold, e.g., $\text{IoU}0.5$.

C. You Only Look Once

Classical approaches to object registration first use regions in the image to determine object bounding boxes and, in a second step, carry out the classification [15]–[17]. This two-step process limits the processing speed, potentially hindering real-time applications. In this work, we use the latest version of You Only Look Once (YOLO), a single-step object detector proposed by [18]. YOLO treats object detection as a regression problem and uses a fully connected layer convolutional neural network approach to solve it. Due to this single-step approach, the YOLO architecture outperforms two-step competitor architectures speed-wise and qualifies for embedding [18]. Unlike in the preceding work, YOLO uses a grid to localize all objects in a single image at once. Each of the $I$ grid cells $i \in \{1, \ldots, I\}$ predicts $B$ bounding boxes $j \in \{1, \ldots, B\}$, each described by 5 parameters (cartesian coordinates $x$ and $y$, width $w$, height $h$, and confidence $c$).

$$\hat{b}_{i,j} = [\hat{x}_{i,j}, \hat{y}_{i,j}, \hat{w}_{i,j}, \hat{h}_{i,j}, \hat{c}_{i,j}]$$

$$c_{i,j} = P(\text{Object}) \cdot \text{IoU}_{i,j},$$
where \( P(\text{Object}) \) denotes the object probability and \( \text{IoU}_{i,j} \) is the IoU for grid cell \( i \) and bounding-box \( j \).

### D. Transfer Learning

Transfer Learning applies neural network models to different problems that it has originally been trained for. This technique can improve the overall model performance while lowering the computational complexity. We propose to transfer the YOLO model weights and biases trained on synthetic data as an initialization to a second similar YOLO architecture trained on a smaller real-data set.

### IV. Numerical Results

#### A. Performance Metrics

1) **Precision, Recall and the Average Precision:** Furthermore, we measure the precision and recall denoted by \( p_{k,n} \) and \( r_{k,n} \) for every class \( k \in \{1\ldots K\} \) and image \( n \in \{1\ldots N\} \).

\[
\begin{align*}
  p_{k,n} &= \frac{\text{TP}_{k,n}}{\text{TP}_{k,n} + \text{FP}_{k,n}}, \quad \text{(4)} \\
  r_{k,n} &= \frac{\text{TP}_{k,n}}{\text{TP}_{k,n} + \text{FN}_{k,n}}, \quad \text{(5)}
\end{align*}
\]

\[
\text{AP}_k = \sum_n (r_n - r_{n-1})p_n \approx \int_0^1 p(r)dr.
\]

Plotting the precision-over-recall values for all images enables us to approximate the area under the curve (AUC), or the average precision (AP) for class \( k \in \{1\ldots K\} \) (Fig. 4). In object detection, we target both high precision and recall indicated by a high AUC value. Furthermore, averaging over all \( K \) classes we obtain the mean average precision (mAP)

\[
m\text{AP} = \frac{1}{K} \sum_{k=1}^{K} \text{AP}_k.
\]

#### B. Pipeline Setup

We use the pipeline as outlined in Sec. II and Fig. 1. We employed the configurations summarized in Tab. I in the BlenderProc setup. As backgrounds we used free and openly available images from Pexels [2], with search query keywords to "Abstract, Pattern, Background" and weights as listed in Tab. II.

#### C. Experimental Setups

**First Setup:** To test how the combination of BlenderProc and domain randomization performs we employ the application of cone detection. For object detection we employ the You Only Look Once (YOLO) neural network architecture.

**Second Setup:** This setup is trained as in the first setup. The weights initialize the network to be trained in a fine-tuning step on the tiny real-world data set.

**Synthetic:** A synthetic large data set for training and a real-world data set is used for testing.

**Fine-tuned:** A synthetic large data set for training in a first step. In a second step, the resulting network model weights initialize the network to be trained in a fine-tuning step on the tiny real-world data set.

For the training and the hyper parameter tuning we utilized YOLOv5’s logging capabilities (https://wandb.ai/site).

**Second Setup:** This setup is trained as in the first setup. The test is carried out on a separate knocked-over-cones data set, neither present in the real-world, nor in the synthetic data set.

**Benchmark:** As a performance benchmark, we trained a YOLOv5s model using real images showing cones under different lighting, weather and environment conditions.

In Tab. III we list the number of manually labeled images/cones. The annotations employing supervisely were manually supported and corrected following the guidelines in
TABLE I: BlenderProc Parameter Configuration

<table>
<thead>
<tr>
<th>Cartesian Room Coordinates for</th>
<th>Object in m</th>
<th>Camera in m</th>
<th>Lighting in m</th>
<th>Intensity in W m⁻²</th>
<th>Render Sample #</th>
<th>Img. #</th>
<th>Cone #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>∈ U([-15, -8, 0], [15, 8, 3])</td>
<td>∈ U([-10, -15, 1], [40, 15, 7])</td>
<td>∈ U([-10, -10, -10], [10, 10, 10])</td>
<td>0.1-5</td>
<td>1-20</td>
<td>1200</td>
<td>9142</td>
</tr>
</tbody>
</table>

TABLE II: Background Query Keywords

<table>
<thead>
<tr>
<th>Query</th>
<th>Number of Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>2936</td>
</tr>
<tr>
<td>Pattern</td>
<td>1464</td>
</tr>
<tr>
<td>Background</td>
<td>660</td>
</tr>
</tbody>
</table>

TABLE III: Real-World Data Set: Classes

<table>
<thead>
<tr>
<th>Images</th>
<th>Total cones</th>
<th>Blue cones</th>
<th>Yellow cones</th>
<th>Orange cones</th>
</tr>
</thead>
<tbody>
<tr>
<td>330</td>
<td>1948</td>
<td>879</td>
<td>853</td>
<td>216</td>
</tr>
</tbody>
</table>

TABLE IV: Real-World Data Set (Train-Test-Split)

<table>
<thead>
<tr>
<th>Image #</th>
<th>Cone #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>330</td>
</tr>
<tr>
<td>Training</td>
<td>200</td>
</tr>
<tr>
<td>Validation</td>
<td>65</td>
</tr>
<tr>
<td>Testing</td>
<td>65</td>
</tr>
</tbody>
</table>

V. PERFORMANCE RESULTS

Real, Synthetic and Fine-tuned (Setup 1): We show the results for the first experimental setup, comparing the detection performance for training on real-world data versus training on synthetically generated data and fine-tuned training in Fig. 5. The results indicate that real training outperforms synthetic training, while both approaches yield high mAPs. If we fine-tune the synthetically trained network using a tiny real-world training, while both approaches yield high mAPs. If we fine-tune the synthetically trained network using a tiny real-world training, we ensure that the neural network applied for object detection purely learns the object features and is background-independent. Our results show that using the synthetically generated training data, fine-tuned with a limited real-world training data set, performs best for object detection. This fine-tuned solution also outperforms the benchmark training data set in detecting knocked-over cones neither contained in the real-world data set nor the synthetic training data set. Our proposed pipeline consequently constitutes a solution to generate extensive training data for cone detection. Furthermore, our study showcases a general concept to overcome the necessity of extensive real-world data labeling efforts in any situation where synthetic object generation is possible, e.g., but not limited to driverless.

VI. CONCLUSION

This contribution introduces a procedural pipeline, combining BlenderProc and domain randomization, to create synthetic, prelabelled training data sets for the formula student driverless challenge. Randomizing the domain using uncorrelated random background images, we ensure that the neural network applied for object detection purely learns the object features and is background-independent. Our results show that using the synthetically generated training data, fine-tuned with a limited real-world training data set, performs best for object detection. This fine-tuned solution also outperforms the benchmark training data set in detecting knocked-over cones neither contained in the real-world data set nor the synthetic training data set. Our proposed pipeline consequently constitutes a solution to generate extensive training data for cone detection. Furthermore, our study showcases a general concept to overcome the necessity of extensive real-world data labeling efforts in any situation where synthetic object generation is possible, e.g., but not limited to driverless.

REFERENCES

Fig. 5: (a-d): Comparing the mAPs for IoU $\geq 0.5$ shows: Using a tiny real data set for fine-tuning (transfer learning) performs close to the benchmark real data set. Purely synthetic data shows the second-best close to optimal performance. That the tiny-real data set is not sufficient to yield a competitive performance highlights the value of the synthetic training with and without fine-tuning. Plot e) suggests our proposed workflow, to apply transfer learning via first training on the synthetic and afterwards on the fine-tuning tiny data set.

Fig. 6: Knocked-over cone detection (top: fine-tuned synthetically trained outperforms bottom: real-world data trained)