

A Deep Learning Approach for Stem Base Detection of Row Crop Plants

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stefan Schweng, BSc

Matrikelnummer 11709472

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze

Mitwirkung: Dipl.-Ing. Dr. techn. Peter Riegler-Nurscher

Wien, 21. September 2023

Stefan Schweng


Markus Vincze

A Deep Learning Approach for Stem Base Detection of Row Crop Plants

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Stefan Schweng, BSc

Registration Number 11709472

to the Faculty of Informatics


at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze

Assistance: Dipl.-Ing. Dr. techn. Peter Riegler-Nurscher

Vienna, 21st September, 2023

Stefan Schweng



Markus Vincze

Erklärung zur Verfassung der Arbeit

Stefan Schweng, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. September 2023

Stefan Schweng

Danksagung

Ich möchte den Abschluss meiner Diplomarbeit als Anlass nehmen, mich von Herzen bei meiner gesamten Familie und meinen Freunden zu bedanken. Es sind diese Menschen in meinem Leben, auf die ich mich immer verlassen kann und auf die ich in allen Lebenslagen zurückkommen kann. Es gab und gibt viele Momente, Menschen und Dinge in meinem Leben, für die ich mich glücklich schätze und für die ich sehr dankbar bin.

Natürlich erhielt ich auch während der Erstellung dieser Arbeit Unterstützung an verschiedenen Stellen, für die ich dankbar bin. In diesem Zusammenhang möchte ich mich besonders bei Peter Riegler-Nurscher für die Betreuung meiner Arbeit bedanken. Weiterhin danke ich Wolfgang Pitzl, der mich bei der Durchführung der Experimente unterstützt hat. Ein besonderer Dank gebührt auch meinem Vater, Karl Schweng, der mich bei der Annotation von Maispflanzen tatkräftig unterstützt hat. Ebenso möchte ich Raphael Schmidt hervorheben, der mir mit seiner Expertise in den Bereichen Photoshop und 3D-Modellierung bei der Erstellung zusätzlicher Pflanzenmodelle wertvolle Hilfe geleistet hat. Zum Abschluss möchte ich mich bei Peter Riegler-Nurscher und Markus Vincze für die Korrektur meiner Arbeit bedanken.

Acknowledgements

I would like to take the completion of my diploma thesis as an opportunity to express my heartfelt gratitude to my entire family and friends. These are the people in my life whom I can always rely on and turn to in all circumstances. There have been and continue to be many moments, people, and things in my life that I consider myself fortunate for and for which I am very thankful.

Of course, I also received support at various stages during the creation of this work, for which I am grateful. In this regard, I would like to extend my special thanks to Peter Riegler-Nurscher for supervising my thesis. Furthermore, I would like to thank Wolfgang Pitzl, who supported me in conducting the experiments. A particular note of gratitude goes to my father, Karl Schweng, who provided valuable assistance in annotating corn plants. I would also like to highlight Raphael Schmidt, who, with his expertise in Photoshop and 3D modeling, greatly helped in creating additional plant models. Finally, I would like to thank Peter Riegler-Nurscher and Markus Vincze for proofreading my work.

Kurzfassung

Diese Arbeit stellt zwei auf Deep Learning basierende Methoden vor: *DeepRow* zur Erkennung von Pflanzreihen und *DeepStem* zur Erkennung der Stammansätze einzelner Pflanzen innerhalb von Pflanzreihen. Ein zentraler Schwerpunkt dieser Forschungsarbeit liegt auf der Entwicklung von Trainingsdatensätzen für die DeepRow- und DeepStem-Netzwerke. Konkret wird untersucht, ob die Leistung dieser Netzwerke durch die Ergänzung von Trainingsdatensätzen, die aus realen Aufnahmen von Maisfeldern bestehen, mit synthetisch erzeugten Bildern verbessert werden kann. Dabei wurden verschiedene Varianten von Datensätzen mit Farmlandbildern erstellt, einschließlich solcher, die zu 100% aus realen Bildern, zu 100% aus synthetischen Bildern oder aus gemischten Datensätzen bestehen. Die realen Bilder wurden manuell unter Verwendung des Annotationstools LABELIMAGE erstellt, während zur automatisierten Erzeugung und Annotation der synthetischen Bilder ein Modifikationstool des Computerspiels FARMING SIMULATOR 22 verwendet wurde. Diese Studie zielt darauf ab, die Wirksamkeit synthetischer Farmlandbilder zur Steigerung der Leistung der DeepRow- und DeepStem-Modelle zu untersuchen. Darüber hinaus untersucht ein Teil der Studie die Auswirkungen von verschiedenen virtuellen Pflanzen- und Bodentexturen bei der Generierung synthetischer Bilder.

Die Ergebnisse für DeepRow zeigen, dass die Ergänzung von realen Datensätzen mit synthetischen Bildern die Leistung positiv beeinflusst. Darüber hinaus trägt der gewählte Ansatz der Textur-Editierung positiv zur Genauigkeit der Pflanzreihen-Erkennung bei. Konkret führte ein Datensatz, der aus 20% realen Bildern und 80% textureditierten synthetischen Bildern bestand, zu einer Verbesserung von 9,06% bei den Metriken *Accuracy* und 6,4% bei *IoU*, im Vergleich zu einem Datensatz, der ausschließlich aus realen Farmlandbildern bestand.

Im Gegensatz dazu zeigte die Leistung der DeepStem-Modelle einen negativen Trend, wenn synthetische Bilder in den Trainingsdatensatz aufgenommen wurden. Darüber hinaus neigte der präsentierte Ansatz der Textur-Editierung dazu, die Leistungsmetriken für DeepStem-Modelle zu reduzieren.

Abstract

This work introduces two deep learning-based methods: *DeepRow* for the detection of crop rows and *DeepStem* for identifying the stem bases of individual plants within crop rows. A central focus of this research lies in the development of training datasets for the DeepRow and DeepStem networks. Specifically, we investigate whether the performance of these networks can be enhanced by supplementing training datasets consisting of real field images with synthetically generated images. Various variants of farmland image datasets were created, including those consisting of 100% real images, 100% synthetic images, and mixed datasets. Real images were manually annotated using the LABELIMAGE annotation tool, while a modification tool from the computer game FARMING SIMULATOR 22 was employed for the automated generation and annotation of synthetic images. This study aims to examine the effectiveness of synthetic farmland images in improving the performance of the DeepRow and DeepStem models. Additionally, a part of the study explores the effects of different virtual plant and ground textures in the generation of synthetic images.

Regarding DeepRow, the results demonstrate that augmenting real datasets with synthetic images positively influences performance. Furthermore, the chosen approach of texture editing contributes positively to crop row detection accuracy. Specifically, a dataset consisting of 20% real images and 80% texture-edited synthetic images yielded a 9.06% improvement in *Accuracy* and a 6.4% improvement in *IoU* compared to a dataset consisting solely of real farmland images.

In contrast, the performance of DeepStem models exhibited a negative trend when synthetic images were added to the training set. Additionally, the proposed approach of texture editing tended to reduce performance metrics for DeepStem models.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Challenge	2
1.2 Contributions	4
1.3 Results preview	5
1.4 Thesis outline	5
2 Background	7
2.1 Mechanical Weeding	7
2.2 Deep Learning	8
2.3 Farming Simulator	14
2.4 Literature Review	15
3 Dataset Generation	19
3.1 Real Dataset	19
3.2 Synthetic Dataset	22
4 Evaluation and Results	47
4.1 Dataset Definitions	47
4.2 Study of Deep Learning Methods	49
4.3 Evaluation on Deep Learning Models	52
5 Conclusion	59
5.1 Results	59
5.2 Future Work	60
List of Figures	63
List of Tables	67
	xv

Bibliography

Introduction

In order to increase agricultural yields, farmers aim for low weed density on their fields. A cheap and convenient way of achieving this is chemical weed control, i.e. the use of herbicides. However, several studies in the past two decades revealed a significant negative impact of herbicides and other types of pesticides on climate resilience [LBG⁺16], biodiversity [PFLCO⁺21] and especially the abundance of pollinators [MHW⁺20] in ecosystems around the world. Further, the use of herbicides in high quantities in agriculture for food production has a negative impact on human health [BPSG04].

In 2020 the European Commission approved a set of policy initiatives, referred to as *European Green Deal* [EUD22], with the aim to make Europe a net-zero emitter of greenhouse gases by 2050. One of these initiatives is the reduction of use and risk of chemical pesticides by 50% by 2030.

Due to the reasons mentioned above and the growing herbicide-resistance of weed [Pie10], the demand for organic food production is increasing, which makes it necessary to find and use alternatives to chemical weed control. One alternative approach for row crop is a process called *hoeing*, where metal tools are pulled through farmland in order to remove weed mechanically. One of the big challenges for applications like these is to make sure that weed gets removed efficiently while the crop stays unharmed.

Figure 1.1 shows a hoeing machine with camera control on a field. The metal tools, being pulled through the field, are mounted on a frame with a fixed spacing in between them. The camera, which is mounted on top of the device, is part of a computer vision system which processes farmland images in real-time in order to gather information on the crop row alignment. Further, this information can be used to automatically adjust the tool-frame's linear offset normal to the driving direction of the tractor.

Up to a certain level of weed coverage and variance in crop height, state-of-the-art systems, as shown in Figure 1.1, achieve significantly higher efficiency while enabling higher tractor speeds compared to hoeing approaches with manual tool-frame offset adjustment. Stem



Figure 1.1: Maize hoeing with camera control. (Source: <https://www.einboeck.at>, accessed on 12.10.2022)

base detection enables not only precise row guidance but also in-row weeding. With in-row weeding, tools are mechanically moved between individual plants within a row, allowing for even less weed coverage.

Under sub-optimal conditions however, there is still room for improvement for systems like these when it comes to detecting crop rows. The goal of this thesis is to utilize convolutional neural networks (CNNs) for stem base detection of row crop plants in order to overcome these limitations.

Figure 1.2 shows examples of maize plants at different growth stages and different intensities of weed coverage. The goal of this work is to detect the stem bases of maize plants as shown in Figure 1.3, which shows the visualization of ground truth bounding boxes for object detection.

1.1 Challenge

As mentioned above, state-of-the-art computer vision systems have problems detecting crop rows correctly under sub-optimal conditions. Examples for sub-optimal conditions in this case are the following:

- a) high weed-pressure,
- b) high variance in crop height,
- c) similar height of crop and weed,



(a) Plants of age < 25 days with low weed coverage.



(b) Plants of age < 25 days and moderate weed coverage.



(c) Plants of age > 25 days and high in-row weed coverage.



(d) Plants of age > 25 days and low weed coverage.

Figure 1.2: Maize plants on a field at different growth stages and different levels of weed coverage.

- d) varying light conditions,
- e) outliers in a crop row,
- f) curved crop rows or
- g) bad image quality (e.g. dusty conditions during image acquisition).

One goal of this thesis is to investigate deep learning methods to detect stem bases of row crop plants with the goal to enable higher accuracy in crop row detection and to tackle the limitations mentioned above.

A pain point of using CNNs in computer vision is the need for large labeled datasets. Generating these datasets usually takes a lot of time, since every image of the set has to be manually labeled. Thanks to continuous progress in computer graphics, there is a growing trend to generate datasets synthetically from simulations [SHE⁺21] or computer games [SLS16], which is done in this work in order to overcome the limitations mentioned above.



Figure 1.3: Ground truth visualization of object detection for maize plant stem bases. Bounding boxes are shown in red.

1.2 Contributions

The goal of this work is to overcome the mentioned limitations of CNN-based methods for crop row detection by using synthetically generated datasets in the training process. More specifically, a state-of-the-art lane detection based (LDB) method - as implemented in the PYTORCHAUTODRIVE framework [FGT⁺22] - as well as the *PointRend* framework [KWHG19] for semantic segmentation are utilized in this work. LDB methods (i.e. the detection of rows instead of individual plants) do not support the idea of in-row weeding, but nevertheless, the effect of using synthetic images in the training process shall be evaluated in this work. Further, the YOLO [RDGF15] framework is used to evaluate the presented approach on object detection methods. PointRend and YOLO are used for detecting individual stem bases whereas the mentioned LDB method aims to detect crop rows.

The computer game FARMING SIMULATOR 22¹ comes with a tool, called GIANTS EDITOR, to create customized maps that can be used by the players during the game. These maps are virtual worlds where players can grow crops and perform a broad variety of other farming tasks virtually. For this work, GIANTS EDITOR and its *LUA* scripting interface are used to autonomously render farmland scenes showing row crop cultures and weed plants, create screenshots of the virtual scenes and generate annotation masks, which can be used for training neural networks.

The main software components needed by the game engine to render objects are described in so called *i3D* files, which either directly contain or reference to other files that serve

¹Farming Simulator 22: <https://www.farming-simulator.com> (accessed on May 20, 2023)

information about shape, dynamics and texture of the virtual object. In order to improve the performance of this approach, the textures of the virtual plants and farmland ground are edited according to information from real farmland imagery so that more realistic virtual scenes can be created.

In order to compare the performance of the implementations of this work to the baseline implementation [RNR22], real-time experiments on real farmland images are carried out. The implementations for stem base and crop row detection are evaluated on a dataset created from real farmland images.

The contributions in this work focus on CNN-based approaches for stem base detection of row crop plants and the capability of synthetic datasets to improve performance for the task of crop row and stem base detection. Additionally, investigations also focus on the effects of virtual texture editing when generating synthetic datasets. As in the work by Riegler-Nurscher and Rupp [RNR22], LDB methods are referred to as *DeepRow*, whereas methods for individual stem base detection are called *DeepStem*. Each CNN model used in this work is trained on different datasets consisting of either 100% real images, 100% synthetic images or mixed variants. Depending on the type of dataset the models are trained on, the methods are referred to as *Real-*, *Synthetic-* and *Mixed-DeepRow* or *DeepStem* respectively.

1.3 Results preview

Results in this work show that the performance of LDB models for crop row detection can be improved by augmenting datasets consisting of real farmland images only with farmland images from synthetic datasets. Furthermore, the results in this thesis show that increasing the variance in synthetic datasets by editing the textures of virtual maize plants and virtual farmland ground increases performance even further. More specifically, it is shown that for Mixed-DeepRow with a dataset consisting of 20% real images and 80% texture-edited synthetic images, the metrics *Accuracy* and *IoU* increase by 9.06% and 6.4% respectively, compared to Real-DeepRow.

Additional results in this thesis demonstrate that it is possible to use PointRend and YOLO networks for detecting stem bases of maize plants in crop rows and distinguishing maize plants from weeds on real farmland. However, in contrast to DeepRow, the performance of DeepStem tends to decrease with the presented approach of synthetic dataset augmentation.

1.4 Thesis outline

This thesis is divided into the chapters *Background* [2], *Dataset Generation* [3], *Evaluation and Results* [4] and *Conclusion* [5]. Chapter [2] summarizes theoretical background relevant in the context of this work, as well as relevant literature. In Chapter [3] the process of generating real and synthetic datasets is described. Details on the selection process of

1. INTRODUCTION

CNN architectures used in this work in addition to results and CNN inference examples are given in Chapter 4. Finally, Chapter 5 summarizes results and provides an outlook for possible future work.

Background

The following sections give details on *Mechanical Weeding*, which is of practical relevance in the context of this thesis, theoretical background information on *Deep Learning* and the computer game FARMING SIMULATOR 22 together with the tool GIANTS EDITOR, which will be used for synthetic dataset generation. The last section in this chapter summarizes the results of relevant academic work in the form of a literature review.

2.1 Mechanical Weeding

Mechanical weeding is as an alternative to chemical weed treatment, i.e. the use of herbicides. Due to the growing resistance of weed against herbicides [Pie10] and the negative impact of herbicides on human health and the environment, as mentioned in Chapter I, this alternative becomes more and more relevant in practice, especially in the EU.

Conventional methods for mechanical weeding of row crops often relies on manual adjustment of the weeding tools. On the other hand, modern applications include camera guided hoeing machines as shown in Figure I.1, but also robots like the so-called *Farmdroid*¹, which utilizes Real-Time Kinematics (RTK) data to remember plant positions from the seeding procedure and uses metal tools to mechanically remove weed plants. Another robot called *Laserweeder*² uses camera based machine learning techniques to identify weed plants and removes them using high-powered lasers.

The deep learning approach presented in this thesis could be used to design applications like the ones mentioned above. Further, existing applications like *Farmdroid* could be improved by fusing RTK data with the real-time information about crop coordinates inferred by the deep learning algorithm.

¹<https://farmdroid.dk>, accessed on 03.08.2023

²<https://carbonrobotics.com>, accessed on 03.08.2023

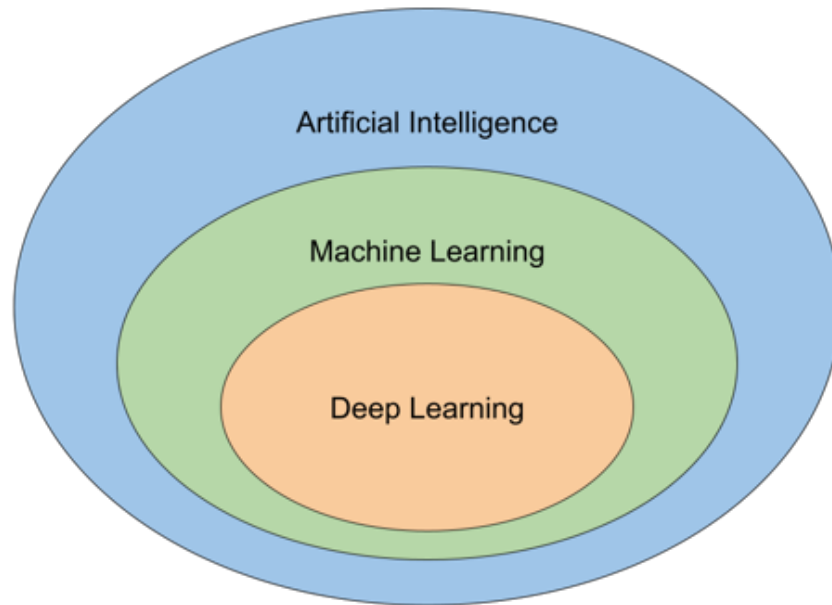


Figure 2.1: Subfields of Artificial Intelligence.

2.2 Deep Learning

Traditional image processing techniques were employed to segment regions of interest based on color, texture, or edge information. However, these methods often struggled with complex scenes and fine-grained details due to limited feature representations. Due to rapidly increasing performance and falling costs in computing power, artificial intelligence became more and more popular in various fields and especially in image analysis.

As shown in Figure [2.1](#), deep learning is a subfield of machine learning, which in turn is a subfield of artificial intelligence. It is based on deep CNNs, where the term *deep* originates from the fact that models of this class consist of many convolutional layers. These deep CNNs are usually trained on a vast amount of data. Two of the most common applications of deep learning include object detection and semantic segmentation. Both terms are introduced in the following subsections.

2.2.1 Object Detection

Object detection is a task in computer vision that involves identifying and localizing objects of interest within an image. It is the task of assigning class labels to objects but also provides bounding boxes to outline their locations. Figure [2.2](#) shows an example of a typical object detection problem.

Early object detection approaches were primarily based on handcrafted features and sliding window techniques. These methods, such as Histogram of Oriented Gradients

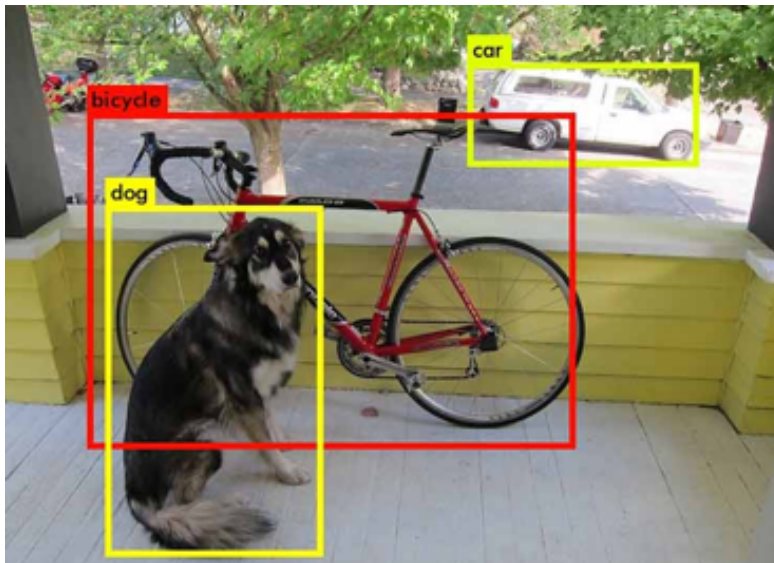


Figure 2.2: Example for Object Detection. (Source: <https://towardsdatascience.com>, accessed on 04.08.2023)

by Dalal and Triggs [DT05] were limited by the need for shallow feature representations and high computational costs.

State-of-the-Art Techniques

The introduction of Region-based Convolutional Neural Networks (R-CNN) by Girshick et al. [GDDM13], marked a breakthrough in object detection. R-CNN and its variants, such as Fast R-CNN and Faster R-CNN, brought end-to-end trainable networks that combined region proposals and object classification.

You Only Look Once (YOLO) by Redmon et al. [RDGF15] is another pioneering approach in object detection. YOLO treats object detection as a regression problem, directly predicting class probabilities and bounding box coordinates in one pass through the network, i.e. one evaluation per image. This real-time detection technique gained popularity due to its simplicity and high inference speed.

2.2.2 Semantic Segmentation

Semantic segmentation is another fundamental task in computer vision that aims to assign a semantic label to each pixel in an image. It plays an important role in scene understanding, enabling machines to perceive and interpret visual information at a pixel level.

In earlier days of computer vision, semantic segmentation was a set of traditional image processing techniques to segment regions of interest based on color, texture, or edge

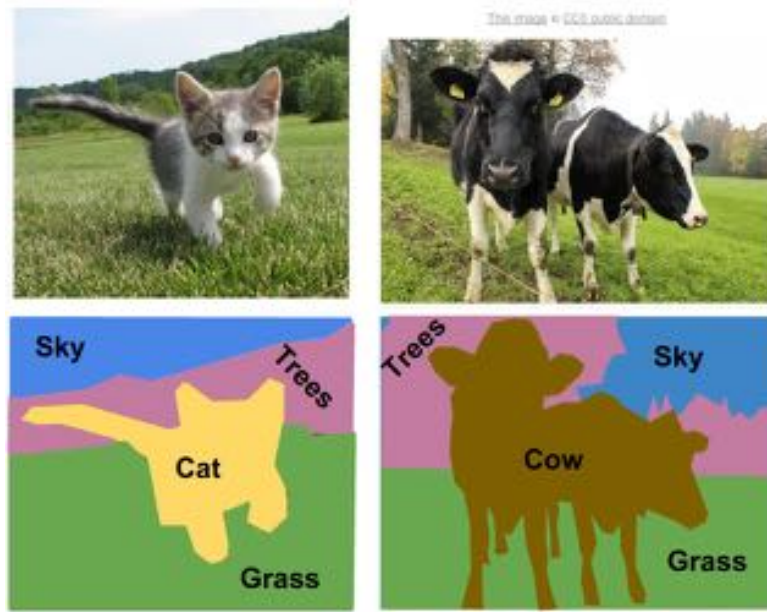


Figure 2.3: Example for Semantic Segmentation, (Source: <https://www.superannotate.com>, accessed on 04.08.2023)

information. However, these methods often struggled with complex scenes and fine-grained details due to limited feature representations.

In recent years, the introduction of deep learning revolutionized semantic segmentation. Deep learning paved the way for end-to-end trainable models capable of learning hierarchical features from image data. This breakthrough empowered researchers to create more sophisticated and accurate segmentation networks.

State-of-the-Art Techniques

A state-of-the-art approach in semantic segmentation is a method called *Fully Convolutional Networks* (FCNs), which was introduced by Long et al. [LSD15]. FCNs employ *deconvolution* (also referred to as *backwards convolution*) to upsample feature maps and generate pixel-wise predictions. The introduction of skip connections and residual blocks further enhanced the performance and reduced the semantic gap between feature maps of different resolutions.

Following FCNs, various so-called *encoder-decoder* architectures gained popularity. In general, the *encoder* part of such a model is used to extract features and put them into feature maps at different scales. In order to end up with a pixel-wise representation with the same dimension as the input image the decoder network uses deconvolution to upsample the feature map representation. Another encoder-decoder model, called U-Net (Ronneberger et al. [RFB15]), improved performance and reduced training time

by introducing skip connections between the encoder and decoder to combine low-level and high-level features, achieving precise boundaries in the segmentation.

Yet another significant invention to focus on multi-scale features is called *atrous convolutions*, which was introduced by Google researchers in the so-called *DeepLab* model series. The first version of this series *DeepLabv1* was introduced by Chen et al. [CPK+15]. In contrast to standard convolution, atrous convolution uses a kernel with strides inside the kernel window to capture multi-scale contextual information to achieve higher accuracy. The second version *DeepLabv2* [CPK+16] incorporated the idea of *Spatial Pyramid Pooling* and combined it with the concept of atrous convolution. This basically means that atrous convolution is used to put features from different scales (by using different kernel window strides) in a feature map of a fixed size, which has proven to make feature recognition even more scale-invariant. To incorporate global context information *DeepLabv3* [CPK+18] adds a separate channel of global average pooling on the last feature map of the model's backbone.

2.2.3 Metrics in Image Analysis

This subsection gives an introduction to metrics used to evaluate the performance of object detection and semantic segmentation applications. The basis of every metric are so-called *ground-truth* labels, which indicate the actual class and position of instances of the objects of interest. In object detection this is done using bounding boxes, while for semantic segmentation pixel-wise annotation are used for labelling.

A very common measure is Intersection over Union (IoU), which is illustrated in Figure 2.4. In the context of object detection one of the squares would correspond to the model's prediction and the other one to a ground truth label. For semantic segmentation the same principle is used, but with free form pixel-wise annotations instead of fixed geometric shapes like squares or rectangles. IoU values are usually provided in the range $[0, 1]$, where a value of 0 corresponds to non-overlapping predictions and ground truth labels and 1 indicates exact overlap.

Object Detection Metrics

In the context of object detection, the IoU metric allows - together with an *IoU-threshold* β - the introduction of following definitions:

- True Positive (TP): $\text{IoU} \geq \beta$ with a ground truth label,
- False Positive (FP): $\text{IoU} < \beta$ for all ground truth labels.

Further, a False Negative (FN) is defined as a ground truth label that has no overlap with a prediction of $\text{IoU} \geq \beta$. Hence, the threshold β defines at which degree of overlap between predicted bounding box and ground truth a prediction is classified as TP, FP or FN.

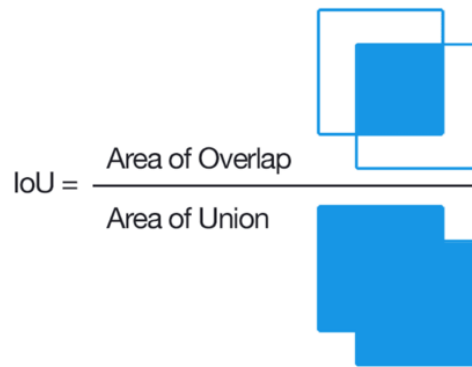


Figure 2.4: Calculation of Intersection over Union, (Source: <https://www.superannotate.com>, accessed on 04.08.2023)

In order to determine which object class is shown in a predicted bounding box, each of those boxes calculates a so-called *confidence score* for each object class, which provides a measure for the likelihood that a certain class is present in the considered bounding box. The object class with the highest confidence score gets selected to determine the class of the predicted object. In order to avoid predictions with low confidence scores another threshold is introduced, the *confidence-threshold*. If the selected class for a predicted bounding box has a confidence score lower than the confidence-threshold, the prediction (i.e. the bounding box) is discarded.

TP, FP and FN allow the definition of two more common measures called *Precision* and *Recall*. Precision is defined by the following formula

$$\text{Precision} = \frac{TP}{TP + FP}, \quad (2.1)$$

whereas Recall is defined as

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (2.2)$$

In other words, precision is the ratio of correctly predicted positive instances to the total predicted positive instances. It indicates the accuracy of positive predictions. On the other hand, recall is the ratio of correctly predicted positive instances to the total actual positive instances. It indicates the ability of the model to capture all relevant instances.

The metrics of precision and recall allow the definition of the so-called *Precision-Recall Curve* (P-R curve). P-R curves give insight on the trade-off between precision and recall, an example for a single object class is shown in Figure 2.5. The curve is plotted by varying the confidence-threshold. This finally allows the definition of the *Average Precision* (AP) metric, which is a value in the interval $[0, 1]$ and equals the area under the P-R curve. Another metric is called *Mean Average Precision* (mAP), which equals the mean value of all AP values across object classes. However, since only one object class is considered in this work, results in Chapter 4 will only pertain the AP metric.

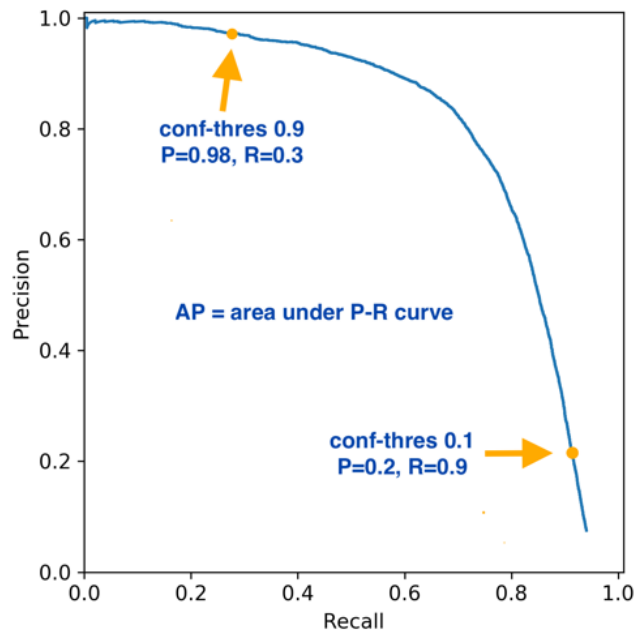


Figure 2.5: Example of a Precision-Recall-Curve for a single object class. (Source: <https://github.com/ultralytics>, accessed on 18.08.2023)

Semantic Segmentation Metrics

Two common metrics in the context of semantic segmentation are *Accuracy* and the so-called *F1* score. In order to define these metrics, the following pixel-based definitions are necessary. In semantic segmentation a pixel is classified as

- True Positive (TP), if it is correctly classified as part of the class,
- False Positive (FP), if it is incorrectly classified as part of the class,
- True Negative (TN), if it is correctly classified as not part of the class or
- False Negative (FN), if it is incorrectly classified as not part of the class.

Based on the definitions above *Accuracy* is defined as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \quad (2.3)$$

The equations for *Precision* (2.1) and *Recall* (2.2) are reused in order to define the *F1* score as follows:

$$F1 = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall}. \quad (2.4)$$



Figure 2.6: Synthetic image of a maize field from FARMING SIMULATOR 22

2.2.4 Synthetic Datasets

As mentioned above, the basis of every metric in image analysis are ground truth labels. Generating ground truth label masks is often a manual and time consuming task. In many cases human experts have to annotate objects of interest by hand.

In this thesis, datasets consisting of synthetic images from the computer game FARMING SIMULATOR 22. More specifically, the tool GIANTS EDITOR, provided by the game developers *Giants Software* is used to automate the process of taking scenes from virtual farmlands and extract stem base coordinates of maize plants from the game engine. Further, the extracted coordinates are used to generate annotation masks in an automated way. Figure [2.6](#) shows an example of screenshot taken from FARMING SIMULATOR 22.

Besides the obvious benefit of saving time, this process also has the advantage of the ability to parameterize the virtual environment in the simulator. This ability ranges from deciding on types, positioning and frequency of weeds in the field to lighting conditions by changing the suns position and other changes of physical conditions.

2.3 Farming Simulator

FARMING SIMULATOR 22 is a computer game where players take on the role of a farmer and experience various aspects of agricultural life. The game aims to provide a realistic farming experience, allowing players to manage a farm, cultivate crops and use a wide range of farming machinery and equipment.

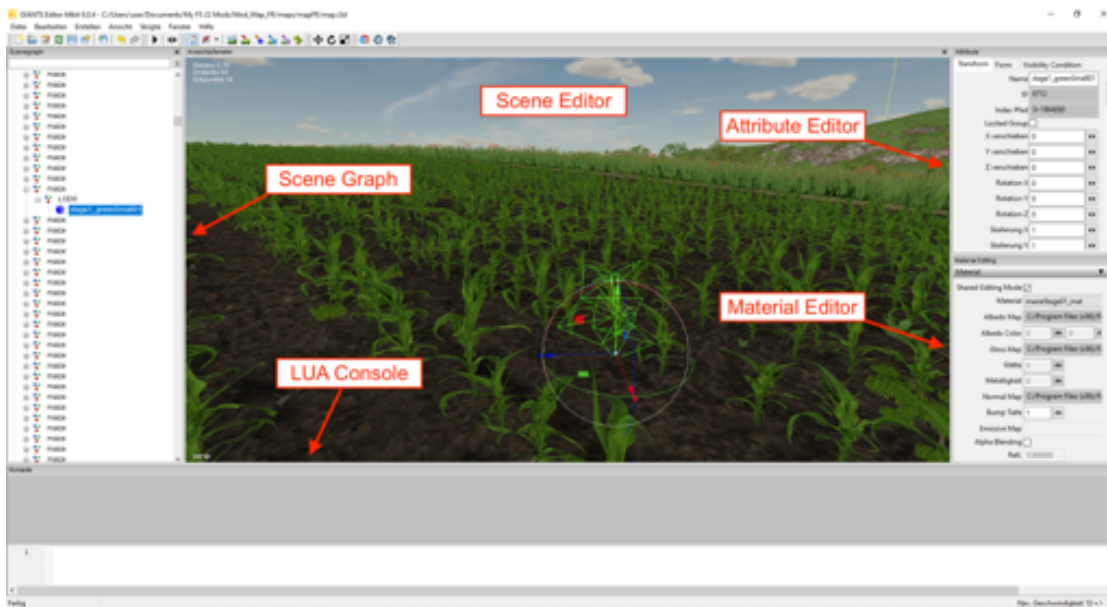


Figure 2.7: Screenshot GIANTS EDITOR

The tool GIANTS EDITOR is usually used by gamers to create their own virtual maps for the game. For this thesis it is used to automate the screenshot and labeling process. The generated datasets are used to train the deep learning models discussed in this thesis with synthetic images only or in addition to real farmland images.

2.3.1 Giants Editor

Figure 2.7 shows a screenshot of GIANTS EDITOR. It provides the user with a graphical interface called the *Scene Editor* which can be used to arrange objects on the 3D map per drag and various other windows to interact with the virtual environment, also referred to as the *Scene*.

In the course of this thesis the most relevant way of interacting with the scene is the *LUA* scripting interface. *LUA* scripts are utilized to automate the random arrangement of maize and weed plants, as well as the creation of screenshots at different views of the maize field. In addition, this interface is used to store information about the virtual maize plants' positions, which in turn is used in a post-processing step to create annotation masks for training the deep learning models.

2.4 Literature Review

This section serves as an overview for relevant literature on crop row detection and stem base or plant position estimation respectively. In the initial literature review a significantly higher number of scientific articles on crop row detection than on stem

base detection was found. This is the reason why there shall be an implementation for crop row detection in the course of this thesis, since it enables better comparison to existing approaches. Further, the lack of academic work in the area of stem base detection additionally yields the need for research in this area.

Zhang et al. [ZLZ⁺18] introduce and apply a new vegetation index to farmland RGB images, followed by a two-staged thresholding process in order to segment maize, weed and background. From this segmentation a binary image is obtained with all the pixels classified as maize. After applying a dilation process to fill gaps between maize pixels in the binary image, feature points (i.e. maize plant positions) are extracted and the number of crop rows is determined via two separate processes, each based on the vertical projection method [JKDC08]. The last step of the proposed method, the detection of crop rows, aims to find feature points belonging to the same crop row by clustering them based on distance and angle constraints between the feature points. In order to refine the found point sets the authors use the Floyd algorithm [Flo62] to find the shortest path from the bottom to the top of the image for each of the sets. Finally, the least squares method is applied on the final point sets to end up with straight-line equations representing the detected crop rows. The proposed method shows outstanding performance under high weed-pressure conditions and superior performance in direct comparison with the Hough Transform method [Hou62]. The angle error between detected and ground truth crop rows was less than 0.5°.

In [BHC18] Bah et al. use farmland images made from an unmanned aerial vehicle (UAV) to detect crop rows in order to detect inter-row weeds. The images of detected weeds are used to train CNNs, which can then be used to detect weeds on other fields. Like in [ZLZ⁺18] the authors of this method apply a vegetation index on the RGB data as a pre-processing step in order to segment plants and background (i.e. soil, shadows and stones). The pixels classified as plants are transformed to binary images. The binary images are then skeletonized before applying the Hough Transform method in order to detect crop rows.

Haug et al. [HBMO14] apply sliding window feature extraction followed by non-maximum suppression on multi-spectral image data in order to obtain a probability map for plant stems in the processed image. Their implementation aims to detect both, use plant and weed stems. The described sliding window approach calculates feature vectors, consisting of statistical and geometrical features, from image patches which show biomass in their center. The extracted feature vectors are passed to a Random Forest classifier to gather stem probabilities for each patch. Their implementation was tested and evaluated with images from a carrot farm, which resulted in an stem detection rate of 80.4% with a mean position error of 1.88mm.

In [KJ12] Kiani et al. implement a crop detection and positioning method based on an artificial neural network (ANN) for a mechanical weeding application. The authors apply a vegetation index on digital camera images to segment plants and background, like in [ZLZ⁺18] and [BHC18]. The dataset for training consists of 180 images from which seven different shape features of plants have been extracted. With the help of discriminant

analysis the number of features needed for crop and weed classification is reduced from seven to four. For the classification itself an ANN is trained to distinguish between crop and weed. The classified crop pixels are then used to calculate corresponding centroids, which are finally used as an estimation of the crop's stem position. The system's classification accuracy reached a detection rate of 98.9%. The proposed method achieved crop position errors of less than 1.5 cm.

Fu et al. [EXLX14] introduce a method to detect stems of potted tomato plants based on data acquired with the stereo vision system *Kinect*. The depth information is used to segment the potted plant and its background. The background pixels are removed from the corresponding 2D-RGB image and the remaining pixels are skeletonized after transforming the RGB to a binary image. Finally, the Progressive Probabilistic Hough Transform is applied to detect plant stems in the image. The outcomes of this algorithms are used to extract textures of the plant's stems from the original 2D-RGB images. These textures are then used to generate virtual 3D models of tomato plants using OpenGL. The plant models are generated based on the *L-system* [PL90] algorithm.

Dataset Generation

This chapter covers implementation details on the generation of datasets. Real dataset generation is described in Section 3.1, whereas synthetic dataset generation is covered in Section 3.2.

3.1 Real Dataset

In order to create accurate annotations for real farmland images for all deep learning frameworks mentioned in Section 1.2, one has to distinguish three different types of annotations. These three types are annotations for

- semantic segmentation,
- object detection and
- lane detection based methods.

The dataset containing real farmland images is referred to as *Real Dataset* (RD) in this work.

3.1.1 Semantic Segmentation Annotations

In this work, semantic segmentation annotations differentiate two classes: *maize* and *background*. The maize class denotes image coordinates corresponding to the emergence of a maize plant from the ground. These coordinates are marked with filled circles. Figure 3.1 shows a visualization of such annotations. Note that this figure shows a visualization and not a ready-to-use annotation mask. Annotation masks used by semantic segmentation frameworks in this work use gray scale images. The background class has a gray scale value of 0 whereas the maize class has a value of 1.



Figure 3.1: Example visualization of a semantic segmentation annotation.

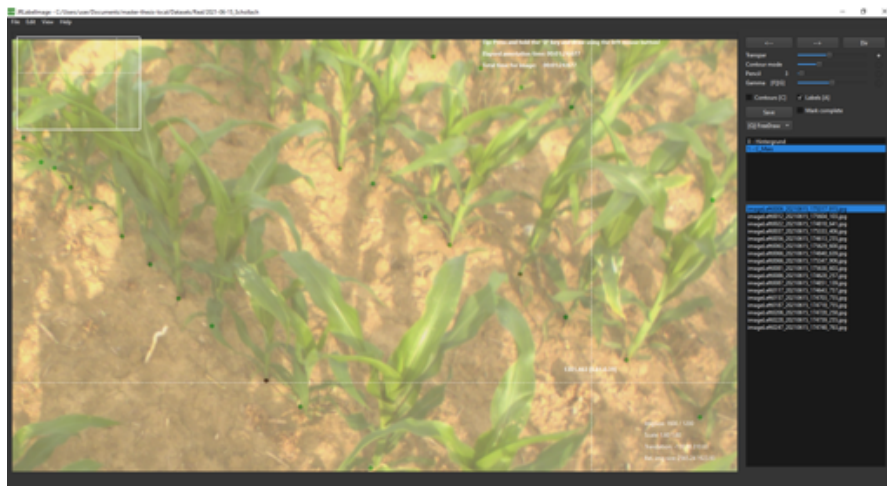


Figure 3.2: Screenshot of the LABELIMAGE user interface.

The research institute *Josephinum Research* provides the tool LABELIMAGE to create semantic segmentation annotation masks. Figure 3.2 shows a screenshot of this program. It provides tools to create circular, line and other types of annotations. For each image annotated with LABELIMAGE, the program automatically generates a gray scale mask with the drawn annotations. In this thesis, this program is used to create annotation masks for a set of 250 images real farmland images, also provided by *Josephinum Research*.

Naturally, the stem base of a maize plant appears bigger on the image, the closer it is to



Figure 3.3: YOLO object detection format. (Source: <https://docs.ultralytics.com/>, accessed on 17.08.2023)

the camera. Due to this fact, it makes sense to annotate stem bases closer to the camera with bigger annotations. However, in order to save time while manually annotating real farmland images with LABELIMAGE, all stem bases are marked with the same pencil size. In order to account for the varying size of stem bases, a python script is used which takes the LABELIMAGE gray scale annotation masks as input and generates edited annotation masks with increased annotation radii. The python script enlarges the annotations to a maximum radius of 18 pixels and a minimum radius of 8 pixels, where the radius is bigger the closer the corresponding stem base is to the lower end of the image. A visualization of the resulting annotation mask is shown in Figure 3.2.

3.1.2 Object Detection Annotations

In order to train a YOLO network with the stem base dataset in this work, rectangular object detection annotations have to be provided for each stem base in an image. For the network training each farmland image has to be provided together with `txt` annotation file, where each line corresponds to a ground truth object, given its object class as well as the corresponding origin, height and width of the rectangular object annotation. An example from the YOLO documentation¹ can be seen in Figure 3.3.

The gray scale semantic segmentation annotation masks with enlarged radii, as described in the previous subsection, are the starting point for the object detection annotation masks. In contrast to the semantic segmentation annotations there is no annotation for the *background* class. This is the reason why there is only one class for object detection in this work, namely the *maize* class. In order to convert the semantic segmentation

¹YOLO annotation documentation: <https://docs.ultralytics.com/datasets/detect/>, accessed on 17.08.2023



Figure 3.4: Example visualization of an object detection annotation.

masks to the required YOLO format, a python script is utilized. This python script uses an OPENCV function called `connectedComponentsWithStats`, which provides information on the circular annotations' origin and radius. The circle origin and radius of each annotation is then used to calculate a corresponding rectangle (or more specifically, a square) origin and side lengths. The calculation results for each circular annotation are then saved to a `txt` file. Figure 3.4 shows a visualization of the YOLO annotations for a real farmland image.

3.1.3 Lane Detection Based Annotations

Like the YOLO annotation format, the LDB methods used in this work, also utilize `txt` annotation files. Lane detection annotations for the real farmland images are generated using the line annotation mode in LABELIMAGE. The program exports a `txt` file where each line corresponds to a line annotation in the image. The line annotations are represented by points, where each point has a x and a y coordinate. This is because one can not only annotate straight lines, but also curves with LABELIMAGE. One line in the `txt` file consists of a sequence of x and y coordinates in interchangeable fashion, where x coordinates are equidistant with 10 pixels. Figure 3.5 shows a visualization of such an annotation including the coordinate frame used by LABELIMAGE.

3.2 Synthetic Dataset

One major advantage of using synthetic datasets from computer games or simulations is the ability to control the environment. With the help of GIANTS EDITOR one can render



Figure 3.5: Example visualization of a LDB annotation.

various plants at any, position, scale and orientation. Additionally - although not utilized in this work - one can also control environmental conditions, e.g. varying the position of the sun to vary shadows in a scene or changing weather conditions. Since rendering the farmland scene, image capturing and annotation generation are automated using scripts, another important advantage is the speed of generating new datasets.

This work focuses on extending the RD (see Section 3.1) with synthetic datasets, where the goal is to make the synthetic datasets look as similar to the real ones as possible. In the course of this thesis two types of synthetic datasets are generated: one dataset *without* and another one *with* texture editing. The former will be referred to as *Synthetic Dataset* (SD), whereas the latter will be referred to as *Synthetic Dataset Texture Editing* (SDTE). The Sections 3.2.1, 3.2.2 and 3.2.3 will focus on generating SD and Section 3.2.4 will cover additional implementation details to generate SDTE.

SD uses all plant models as provided by FARMING SIMULATOR 22 with a few exceptions:

- The red color channel of the *maize* and *grass* model textures are reduced from a value of 255 (Figure 3.6a) to 110 (Figure 3.6b) in order to make the virtual plant colors appear more realistic.
- FARMING SIMULATOR 22 does not provide maize models for plants of age ≤ 25 days. Since a considerable amount of images in the RD shows maize plants in this growth stage, a part of the existing model textures, the so-called *alpha map*, is edited in a way so that it only shows the lower part of the maize model, which is ought to imitate young maize plants. Figure 3.7 shows a comparison of the original and the edited alpha map. After applying the edited alpha map on an existing maize model, the resulting 3D model looks as shown in Figure 3.8b.

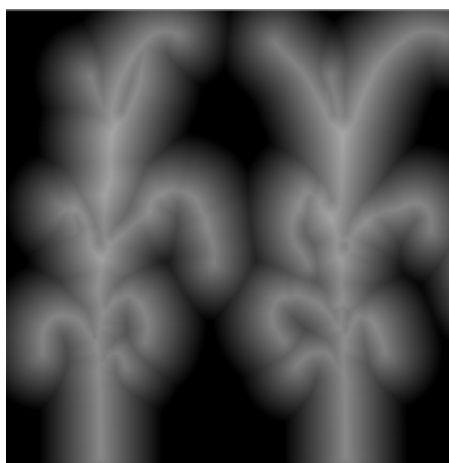


(a) Maize model with original colors.

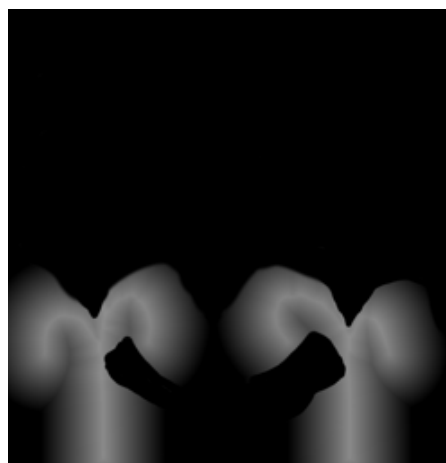


(b) Maize model where the red color channel is reduced from 255 to 110.

Figure 3.6: A FARMING SIMULATOR 22 maize model with different texture colors.



(a) Original alpha map.



(b) Edited alpha map for early maize growth stage.

Figure 3.7: Alpha map editing of a FARMING SIMULATOR 22 maize model.



(a) Model with original alpha map.



(b) Model with edited alpha map (Figure 3.7b) for maize plants of age ≤ 25 days.

Figure 3.8: A maize plant model with two different versions of alpha maps.

3.2.1 Rendering Virtual Environments

The main tool for this and the following subsections is GIANTS EDITOR and the *LUA* scripting interface for the *Giants Engine* (i.e. the game engine). The starting point for the automated rendering procedure in this work is an existing virtual map from FARMING SIMULATOR 22. This map can be opened and edited in GIANTS EDITOR, which is shown in Figure 2.7.

Objects in the virtual scene can be nested in so-called *transform groups* (TGs). These objects can be anything one has a valid *i3D* model for. These models can be loaded via the *LUA* scripting interface and inserted into TGs in form of so-called *scenes*. For example, there is a TG for maize plant scenes and another one for *Rumex* plant scenes (which is a type of weed plant).

In order to get started with rendering the virtual maize field, all TGs and scenes are removed from an existing field on the map. This is done because each maize plant in the field shall be rendered from scratch, so that it is easier to infer the maize plant positions, which are needed to generate the SD annotations. As an additional manual step of preparation, the ground texture is changed to the ground texture type *planted*, which comes with all the other models and textures from FARMING SIMULATOR 22. Figure 3.9 shows a selection of ground texture types.



Figure 3.9: FARMING SIMULATOR 22 ground texture types.

Throughout the whole rendering procedure there are two random distributions used in various steps of the process: the uniform distribution $U(a)$, which generates a random integer in the interval $[1, a]$ and the normal distribution $N(\mu, \sigma)$, which generates a real random number according to a normal distribution with mean μ and standard distribution σ . However, the *LUA* scripting interface only provides a uniform random function. Hence, the *Marsaglia Polar Method* [MB64] is used to model a normal random function using the existing uniform random function.

The *LUA* scripts used in this work are either used to create TGs and render its contents (this section) or to create screenshots together with a xml file to store maize plant coordinates (Section 3.2.2).

The rendering procedure is a multi-step process, where each step has its own *LUA* script. Finally, there is one top-level *LUA* script that runs all the other scripts to render each aspect of the scene. The following subsections document each step of the process.

Maize Rendering

FARMING SIMULATOR 22 makes all 3D models (in form of *i3D* files) and textures available to gamers, so that they can be used for creating custom maps in GIANTS EDITOR. Also the map file itself is a *i3D* file. The game provides various models for maize and weed plants. Usually all maize models are stored in one *i3D* file, where each model instance has its own TG. The same holds for weed plant models.

There are two maize plant models provided by FARMING SIMULATOR 22 that will be used for the rendering step, one of them is shown in Figure 3.8a. In addition to the two original maize plants, two more are generated by applying edited alpha maps (like the one shown in Figure 3.7b) to the original models, which results in models that are used to imitate young maize plants of age ≤ 25 days (see Figure 3.8b). In order to make



(a) Dropout segments on real farmland.

(b) Dropout segments on synthetic farmland.

Figure 3.10: Maize dropout segments due to sub optimal conditions on the farmland or while seeding.

it easier to load maize model via the *LUA* scripting interface they are extracted into separate *i3D* files.

The *LUA* script for actually rendering the maize plants has two modes: *Young Maize* (YM) and *Old Maize* (OM). This mode is chosen manually. In YM mode, the two maize models with edited alpha maps are used and in OM mode, the two original models are used. These maize models will be referred to as YM1/YM2 and OM1/OM2 respectively. In each case, the maize plants are rendered in a grid with fixed spacing. However, there are multiple random rules applied to each maize plant on the field which are described in Tables 3.2, 3.3 and 3.4.

The length and width of the field are defined in the x and z direction of the virtual 3D map. Hence, the height is defined in the y coordinate. In the following, the nominal coordinates of a virtual maize plant in the field are denoted as x_m , y_m and z_m respectively. The maize plants rotation with respect to the y axis is denoted as β_m . Finally, the maize plant's scale in each coordinate is denoted as s_{x_m} , s_{y_m} and s_{z_m} respectively, where s_{y_m} is the plant height. The origin of the field is referred to as x_{F_O} in x direction and as z_{F_O} in z direction. These and other variables are described in Table 3.1.

Figure 3.10a shows a common artefact on a real maize field. Due to sub optimal conditions on the farmland or during the seeding procedure, maize fields shows dropout segments, where plants are either in an earlier stage of development or missing completely.

In order to model these dropout segments, an approach utilizing fourth order polynomials is used. As shown in Figure 3.11, the fourth order polynomial is used to scale the maize plant heights within the dropout segment. In case a maize plant has $y_m < y_{min}$ it is not rendered. The length of a dropout segment DSL is calculated as

$$DSL = \max(DSL_{min}, N(\mu_{DSL}, \sigma_{DSL})), \quad (3.1)$$

3. DATASET GENERATION

Variable	Description	Value in OM	Value in YM
μ_{sym}	Maize plant scaling mean in y direction.	0.2	0.14
σ_{sym}	Maize plant scaling standard deviation in y direction.	0.05	0.035
$drop_m$	Simulating random dropout of individual plants using uniform distribution. One in $drop_m$ maize plants is not rendered.	15	15
y_{min}	Minimum maize plant height.	0.1	-
μ_{DSL}	Mean length of a maize dropout segment, see Figure 3.10b.	1.5	1.5
σ_{DSL}	Standard deviation of maize dropout segment length.	0.5	0.5
DSL_{min}	Minimum dropout segment length.	0.3	0.3
μ_{NDS}	Mean value for number of dropout segments.	18	18
σ_{NDS}	Standard deviation value for number of maize dropout segments.	6	6
N_{DS}	Number of dropout segments on the field.	See Table 3.2	
x_{FO}	Maize field origin in x direction.	-71.87	-71.87
z_{FO}	Maize field origin in z direction.	159.25	194.25
Δx_{max}	Maize field length in x direction.	20	20
Δz_{max}	Maize field width in z direction.	30	30
Δx_F	Field grid spacing in x direction.	0.1	0.1
Δz_F	Field grid spacing in z direction.	0.3	0.3
σ_{xzm}	Standard deviation for individual maize stem base coordinates in x and z .	0.008	0.008
x_m	Maize stem base coordinate in x direction.	See Table 3.2	
y_m	Maize stem base coordinate in y direction.	See Table 3.2	
z_m	Maize stem base coordinate in z direction.	See Table 3.2	
β_m	Maize rotation around y axis.	See Table 3.2	
s_{x_m}	Maize plant scaling in x direction.	See Table 3.3 and Table 3.4	
s_{y_m}	Maize plant scaling in y direction.	See Table 3.2	
s_{z_m}	Maize plant scaling in z direction.	See Table 3.3 and Table 3.4	

Table 3.1: Description of maize rendering variables.

Maize rendering rules for YM and OM

Render YMX/OMX, where $X \sim U(2)$

$$x_m = x_{F_O} + n \cdot \Delta x_F + N(0, \sigma_{xz_m}), \forall n \in \mathbb{N}, 0 \leq n \leq \frac{\Delta x_{max}}{\Delta x_F}$$

$$z_m = z_{F_O} + n \cdot \Delta z_F + N(0, \sigma_{xz_m}), \forall n \in \mathbb{N}, 0 \leq n \leq \frac{\Delta z_{max}}{\Delta z_F}$$

$$y_m = \text{getTerrainHeight}(x_m, z_m)$$

$$\beta_m = U(360)^\circ$$

$$s_{y_m} = N(\mu_{s_{y_m}}, \sigma_{s_{y_m}})$$

$$N_{DS} = N(\mu_{NDS}, \sigma_{NDS})$$

Table 3.2: General rendering rules for generating maize fields in GIANTS EDITOR.

Note: `getTerrainHeight(x, z)` is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.

Maize rendering rules for OM only

Don't render plant if $s_{y_m} < y_{min}$

$$s_{x_m} = s_{z_m} = s_{y_m} + 0.25 \text{ if } s_{y_m} < \mu_{s_{y_m}} \text{ else } s_{y_m} + 0.3$$

Table 3.3: Rendering rules for generating maize fields in GIANTS EDITOR applied specifically in OM mode.

Maize rendering rules for YM only

$$s_{x_m} = s_{z_m} = s_{y_m} + 0.175 \text{ if } s_{y_m} < \mu_{s_{y_m}} \text{ else } s_{y_m} + 0.21$$

Table 3.4: Random rules for rendering maize plants in GIANTS EDITOR applied specifically in YM mode.

based on which the number of maize plants in a segment δ_m is

$$\delta_m = \left\lfloor \frac{DSL}{\Delta x_F} \right\rfloor + 1. \quad (3.2)$$

The parameters for the fourth order polynomial are then calculated based on the general polynomial equation

$$y = ax^4 + bx^3 + cx^2 + dx + e \quad (3.3)$$

with $b = c = d = 0$. Finally, a can be determined by

$$a = \frac{y - e}{x^4} \quad (3.4)$$

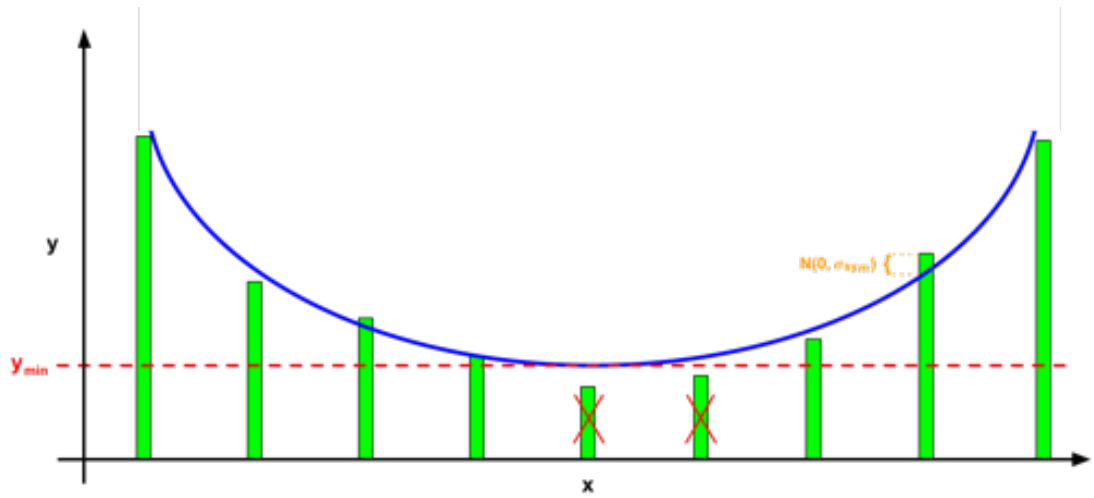


Figure 3.11: A fourth order polynomial (blue) is used to model maize dropout segments. The plant height s_{y_m} is scaled according to the polynomial. In addition to polynomial height scaling, random scaling is added according to $N(0, \sigma_{sy_m})$. Maize plants are shown in green. Maize plants with $y_m < y_{min}$ are not rendered.

where $e = y_{min}$, $y = \mu_{sy_m}$ and

$$x = -\frac{\delta \cdot \Delta x_F}{2}. \quad (3.5)$$

In total, N_{DS} dropout segments are rendered on the field. The starting point of an individual dropout segment is identified by a maize plant index, i.e. the grid index of a maize plant, which is the first plant scaled according to the fourth order polynomial. For each segment the initial maize plant index (crop row index and maize plant index) is determined using the uniform distribution function.

The following subsections cover the rendering procedure of various weed plants.

Clover Rendering

The first type of weed plants added to the virtual maize fields is clover. Figure 3.12 shows the *i3D* clover model used in FARMING SIMULATOR 22. This model is also used for rendering clover in this work.

A work by Sommerville et al. [SSMM20] compares methods for spatial modelling of weed plants on a field. One of them utilizes cellular automata (CA), which - in two dimensions - can be thought of as a grid of cells, where each cell can be *inactive* (0) or *active* (1). Each cell on the grid is randomly initialized as 0 or 1. After this, the CA approach applies a dispersal algorithm in each simulation step. The pseudo code for the cellular automaton algorithm used in this work is shown in Algorithm 3.1. Figure 3.13 shows an example of results for *cellular_automaton*(30, 30, 5, 50).

Algorithm 3.1: Cellular automaton algorithm (*cellular_automaton*) used for spatial clover modelling.

Input: An integer *width*, an integer *height*, an integer *min_iterations* and an integer *min_active*

Output: A matrix **C**

```

1 C ← init(width, height, 0)
2 for x ← 1 to width do
3   for z ← 1 to height do
4     val ← 0
5     u ← uniform(0, 1) // random float between 0 and 1
6     if u < 0.45 then
7       | val ← 1
8     end
9     C[x, z] ← val
10  end
11 end
12 i ← 0
13 num_active_cells ← ∞
14 while i < min_iterations or num_active_cells < min_active do
15   Ccopy ← C
16   num_active_cells ← 0
17   for x ← 1 to width do
18     for z ← 1 to height do
19       active_neighbours ← count_active_neighbours(C, x, z)
20       if C[x, z] > 0 then
21         if active_neighbours < 2 or active_neighbours > 3 then
22           | Ccopy[x, z] ← 0
23         else
24           | Ccopy[x, z] ← 1
25           | num_active_cells ++
26         end
27       else
28         if active_neighbours == 2 or active_neighbours == 3 then
29           | Ccopy[x, z] ← 1
30           | num_active_cells ++
31         else
32           | Ccopy[x, z] ← 0
33         end
34       end
35     end
36     i ++
37     C ← Ccopy
38   end
39 end
40 return C

```



Figure 3.12: FARMING SIMULATOR 22’s clover model.

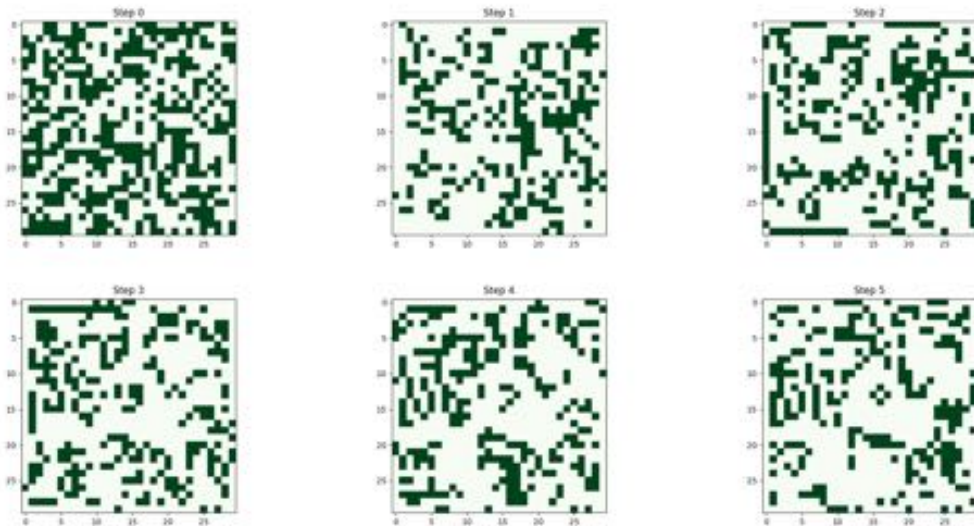


Figure 3.13: Example results for $cellular_automaton(30, 30, 5, 50)$.

In order to model the spatial clover distributions for the synthetic datasets in this work, the virtual maize field is divided into a grid of 500 x 336 cells. The spacial clover distribution is then represented by $clover_cells$, which is generated by

$$clover_cells = cellular_automaton(500, 336). \quad (3.6)$$

A clover plant is then planted at the center of each cell of the field, where the corresponding element in $clover_cells$ equals 1. In addition, random rendering rules are applied according to Tables 3.5 and 3.6. Figure 3.14 shows the rendered clover plants on a virtual field, distributed according to the CA approach.

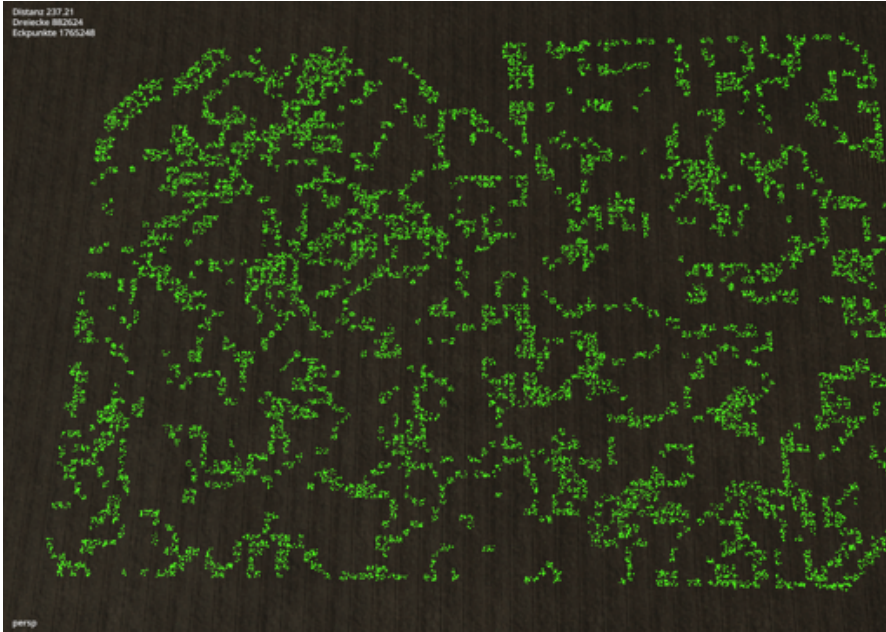


Figure 3.14: Clover distribution on the virtual field. Rendered clover plants are highlighted in green.

Variable	Description	Value in OM	Value in YM
μ_{s_c}	Clover plant scaling mean in x , y and z direction.	0.12	0.06
σ_{s_c}	Maize plant scaling standard deviation in y direction.	0.03	0.015
x_c	Clover base coordinate in x direction.	See Table	3.6
y_c	Clover base coordinate in y direction.	See Table	3.6
z_c	Clover base coordinate in z direction.	See Table	3.6
β_c	Clover rotation around y axis.	See Table	3.6
s_{x_c}	Clover plant scaling in x direction.	See Table	3.6
s_{y_c}	Clover plant scaling in y direction.	See Table	3.6
s_{z_c}	Clover plant scaling in z direction.	See Table	3.6

Table 3.5: Description of clover rendering variables.

 Clover rendering rules for YM and OM

 x_c according to *clover_cells*
 z_c according to *clover_cells*
 $y_c = \text{getTerrainHeight}(x_c, z_c)$
 $\beta_c = U(360)^\circ$
 $s_{x_c} = s_{y_c} = s_{z_c} = N(\mu_{s_c}, \sigma_{s_c})$

Table 3.6: Rendering rules for rendering clover plants in GIANTS EDITOR.

Note: $\text{getTerrainHeight}(x, z)$ is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.



Figure 3.15: FARMING SIMULATOR 22's rumex model.

Rumex Rendering

In this step of the rendering procedure, rumex plants are rendered as another type of weed. FARMING SIMULATOR 22's rumex *i3D* model is shown in Figure 3.15. As in the case of clover rendering, there is an algorithm for modelling the spatial distribution of rumex plants (see Algorithm 3.2). Variables for the rumex rendering sub-procedure are defined in Table 3.7.

In order to determine the rumex plant coordinates, Algorithm 3.2 divides the rendered maize field into sub-cells. The number of cells is based on a random number $\text{num_rumex_plants} = \max(1, N(\mu_{n_r}, \sigma_{n_r}))$, which is also related to the number of rumex plants planted on the field. The helper function *plant_rumex* in Algorithm 3.2 follows the rumex rendering rules defined in Table 3.8.

Algorithm 3.2: Algorithm for spatial rumex modelling.

```

1  $num\_rumex\_plants \leftarrow \max(1, N(\mu_{n_r}, \sigma_{n_r}))$ 
2  $sub\_cell\_side\_length \leftarrow \frac{4}{num\_rumex\_plants}$ 
3  $x\_plant \leftarrow 0$ 
4  $z\_plant \leftarrow 0$ 
5 while  $z\_plant < \Delta z_{max}$  do
6   while  $x\_plant < \Delta x_{max}$  do
7      $plant\_rumex(x\_plant, z\_plant)$ 
8      $x\_plant \leftarrow x\_plant + sub\_cell\_side\_length$ 
9   end
10   $z\_plant \leftarrow z\_plant + sub\_cell\_side\_length$ 
11 end

```

Variable	Description	Value in OM	Value in YM
μ_{s_r}	Rumex plant scaling mean in x , y and z direction.	0.1	0.05
σ_{s_r}	Rumex plant scaling standard deviation in y direction.	0.05	0.025
μ_{n_r}	Clover plant scaling mean in x , y and z direction.	8	8
σ_{n_r}	Maize plant scaling standard deviation in y direction.	3.2	3.2
x_r	Rumex base coordinate in x direction.	See Table	3.8
y_r	Rumex base coordinate in y direction.	See Table	3.8
z_r	Rumex base coordinate in z direction.	See Table	3.8
β_r	Rumex rotation around y axis.	See Table	3.8
s_{x_r}	Rumex plant scaling in x direction.	See Table	3.8
s_{y_r}	Rumex plant scaling in y direction.	See Table	3.8
s_{z_r}	Rumex plant scaling in z direction.	See Table	3.8

Table 3.7: Description of rumex rendering variables.

Rumex rendering rules for YM and OM
x_r according to <code>x_plant</code> in Algorithm 3.2
z_r according to <code>z_plant</code> in Algorithm 3.2
$y_r = \text{getTerrainHeight}(x_r, z_r)$
$\beta_r = U(360)^\circ$
$s_{x_r} = s_{y_r} = s_{z_r} = N(\mu_{s_r}, \sigma_{s_r})$

Table 3.8: Rendering rules for rendering rumex plants in GIANTS EDITOR.

Note: `getTerrainHeight(x, z)` is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.

Grass Rendering

From a certain plant growth stage forward (age > 25 days), some maize fields show segments of weed plants within crop rows. This effect usually shows up due to hoeing in previous growth stages of the maize plants. An example for these segments on real farmland is shown in Figure 3.18a. As in the case of previous subsections, this effect is modelled on virtual farmland using random functions. However, since this phenomenon is usually not observed for maize plants of age ≤ 25 days, these segments are only rendered in OM.

FARMING SIMULATOR 22 provides various grass models. Two of them are selected to render the weed segments in this work. The first one shows pure grass (Figure 3.16a) and the second one shows a mix of grass and clover (Figure 3.16b). To model the weed segments, these grass models are rendered in random sections within the virtual crop rows with a certain probability. Since clover plants are already modelled using the CA approach (see previous subsection), the grass model containing clover plants is rendered with a probability of 33.3% whereas the pure grass model is rendered with a probability of 66.6%.

Variables used for the grass rendering process are defined in Table 3.9. Rendering rules for these segments are defined (for OM only) in Table 3.10 and the text below.

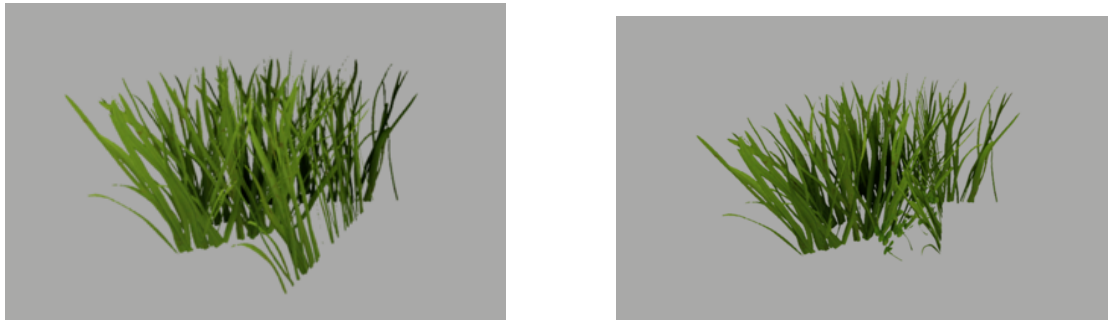
The length of an individual grass segment GSL is calculated in the same way as the dropout segment length. It is calculated in the form

$$GSL = \max(GSL_{min}, N(\mu_{GSL}, \sigma_{GSL})), \quad (3.7)$$

based on which the number of grass models in one segment δ_g is

$$\delta_g = \left\lceil \frac{GSL}{\Delta_g} \right\rceil + 1. \quad (3.8)$$

On real farmland, there is either a high (local) density of grass segments or there are no grass segments at all. Hence, the row index r_{GS} of an individual virtual grass segments



(a) Pure grass model.

(b) Model with a mix of grass and clover.

Figure 3.16: Grass models provided by FARMING SIMULATOR 22.

Variable	Description	Value in OM
μ_{s_g}	Grass plant scaling mean in x , y and z direction.	0.2
σ_{s_g}	Grass plant scaling standard deviation in y direction.	0.03
μ_{r_g}	Mean grass model rotation relative to y axis.	90°
σ_{r_g}	Grass model rotation standard deviation relative to y axis.	5°
μ_{GSL}	Mean length of a grass segment, see Figure 3.18b.	3.5
σ_{GSL}	Standard deviation of a grass segment length.	0.5
GSL_{min}	Minimum grass segment length.	0.3
μ_{NGS}	Mean value for number of grass segments.	48
σ_{NGS}	Standard deviation value for number of grass segments.	6
Δ_g	Grass model spacing in x direction within a grass segment.	0.1
σ_{xz_g}	Standard deviation for individual grass model coordinates in x and z .	0.02
N_{GS}	Number of grass segments on the field.	See Table 3.10
x_g	Grass model coordinate in x direction.	See Table 3.10
y_g	Grass model coordinate in y direction.	See Table 3.10
z_g	Grass model coordinate in z direction.	See Table 3.10
β_g	Grass model rotation around y axis.	See Table 3.10
s_{x_g}	Grass model scaling in x direction.	See Table 3.10
s_{y_g}	Grass model scaling in y direction.	See Table 3.10
s_{z_g}	Grass model scaling in z direction.	See Table 3.10

Table 3.9: Description of grass rendering variables.

Grass rendering rules for OM

x_g according to (3.11)

$z_g = z_{FO} + N(0, \sigma_{xz_g}) + r_{GS} \cdot \Delta z_F$, where r_{GS} is calculated as (3.9)

$y_g = \text{getTerrainHeight}(x_g, z_g)$

$\beta_g = N(\mu_{r_g}, \sigma_{r_g})^\circ$

$s_{x_g} = s_{y_g} = s_{z_g} = N(\mu_{s_g}, \sigma_{s_g})$

Table 3.10: Rendering rules for rendering grass plants in segments in GIANTS EDITOR. *Note:* `getTerrainHeight(x, z)` is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.

is calculated according to

$$r_{GS} = 29 + U(45), \quad (3.9)$$

which means that r_{GS} is drawn uniformly from the interval [30-74]. In this work, a virtual field always has the same total number of crop rows, namely $\frac{\Delta z_{max}}{\Delta z_F} = \frac{30}{0.3} = 100$.

After drawing r_{GS} , the center of the respective grass segment c_{GS} is calculated in the form

$$c_{GS} = x_{FO} + \frac{U(\lfloor \Delta z_{max} \rfloor \cdot 10)}{10}. \quad (3.10)$$

The x coordinate of an individual grass model x_g within that segment is determined by

$$x_g = x_{FO} + N(0, \sigma_{xz_g}) - \frac{c_g}{2} + n \cdot \Delta_g, \forall n \in \mathbb{N}, 0 \leq n \leq \delta_g. \quad (3.11)$$

However, a grass model is not rendered if $x_g < x_{FO}$ or $x_g > x_{FO} + \Delta x_{max}$, i.e. if x_g is outside the field's boundaries in x direction.

Finally, there is one more rendering step for weed plants, which is not automated, i.e. these plants are not re-rendered but instead stay in the same place, even when virtual fields are re-generated.

This layer of weed plants consists of various weed plant models as provided by FARMING SIMULATOR 22 and is used to further increase the variety of weed types and weed density. It is rendered in the left half of the virtual field using GIANTS EDITOR's *Terrain Editing* tool (see Figure 3.17). This layer is only rendered in OM, since the rendered weed types usually do not exist on maize fields with plant age < 25 days.

3.2.2 Screenshot Automation

When generating a new synthetic dataset, all rendering steps described so far are executed sequentially in order to render a new virtual maize field either in OM or YM by one



Figure 3.17: Statically rendered weed plants in order to increase weed plant diversity and density in the left half of the virtual field. Used in OM only.



(a) Dropout segments on real farmland.



(b) Dropout segments on synthetic farmland.

Figure 3.18: Concentrated grass segments within the crop row due to hoeing in previous growth stages.

top-level *LUA* script. In addition to rendering maize plants, the respective *LUA* script also stores the stem base coordinates of all rendered maize plants in two different variants. There is one two-dimensional array - or *table*, as it is called in *LUA* context - for storing the exact rendering coordinates x_m, y_m, z_m (see Table 3.2) and another two-dimensional array to store the maize plant coordinates without the noise influence as drawn from $N(0, \sigma_{xz_m})$. The noise-free version of the coordinates is referred to as x'_m, y'_m and z'_m .

The automated screenshot generation - as described in this section - is initiated by executing a separate *LUA* script. This *LUA* script first sets the camera (also called *view port node*) position and orientation to its initial values and then registers a so-called *update listener*. An update listener is a function that is called periodically, in a giants engine internal time interval dt . In GIANTS EDITOR the periodic execution of registered

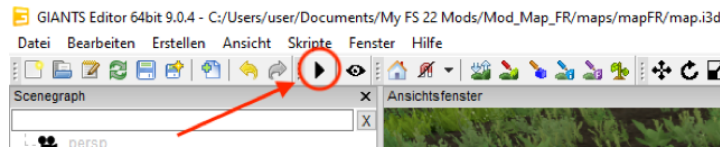


Figure 3.19: *Play* button in GIANTS EDITOR to start periodic execution of registered update listener functions.

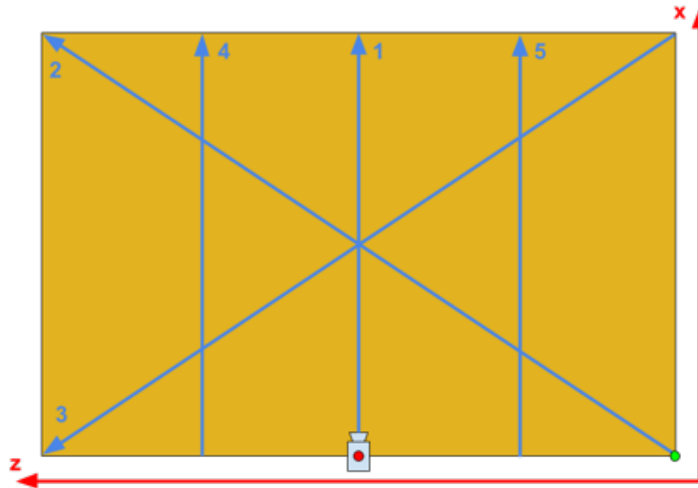


Figure 3.20: Camera movement paths (blue arrows) when generating a new dataset in GIANTS EDITOR. The numbers next to the camera paths represent the order in which the paths are followed. The green dot indicates the origin coordinates of a virtual maize field x_{FO} and z_{FO} , whereas the red dot indicates the initial position of the camera node.

update listeners can be started by clicking the *play* button in the top left of the editor window (see Figure 3.19).

This functionality is usually used for physical simulations. However, in this work it is used to move the camera along certain paths. After moving the camera, the update listener function creates a screenshot, projects all maize stem bases onto the camera's current 2D view plane and saves the projected screen coordinates of all currently visible stem bases to a xml file, i.e. there is one xml file per screenshot. Figure 3.20 shows the field's origin (green dot), the initial camera position (red dot) and the camera movement paths (blue arrows).

The coordinate projection from three to two dimensions is performed using the giants engine's built-in `project` function, which takes 3D world coordinates as an input and outputs three projected coordinates x_m^p , y_m^p and z_m^p . In the case of noise-free coordinates, these projections are referred to as x_m^p , y_m^p and z_m^p . The projected coordinates x_m^p and y_m^p correspond to the screen coordinates in x and y direction of the image, whereas the coordinate z_m^p provides depth information of a projected point, since it equals the

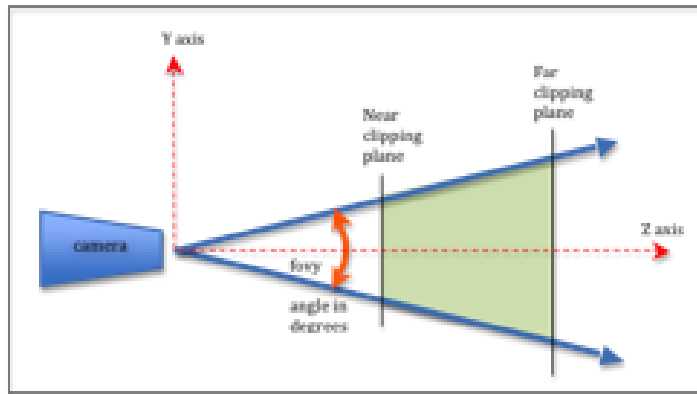


Figure 3.21: Side view of the perspective projection used to transform 3D stem base coordinates to 2D image plane coordinates. In addition to the screen coordinates x_m^p and y_m^p , the projection also outputs a depth coordinate z_m^p , which equals the distance of the 3D point to the image plane. The projected coordinates are normalized to the interval $[0, 1]$. The normalization in z_m^p is performed with respect to the *near clipping plane* and the *far clipping plane*. (Source: <http://learnwebgl.brown37.net>, accessed on 29.08.2023)

distance of the 3D point to the image plane.

Each projected coordinate is normalized to the interval $[0, 1]$. x_m^p and y_m^p are normalized using the width and height of the screenshot and z_m^p is normalized using the *near* and *far clipping plane* (see Figure 3.21). Both clipping planes can be configured in GIANTS EDITOR. In order to figure out, which stem bases are currently visible on the screen, the following check is performed: $0 \leq s_x, s_y, s_z \leq 1$, i.e. a stem base is currently visible if each projected coordinate is within the interval $[0, 1]$.

3.2.3 Annotation Generation

As mentioned in previous subsections, annotations for SD and SDTE are generated from FARMING SIMULATOR 22 screenshots and xml files containing projected screen coordinates of visible maize plant stem bases. The xml stem base coordinates are stored in an *array-like* manner, in order to group stem bases by crop rows. More specifically there are two arrays, one storing the exact rendering coordinates x_m, y_m, z_m and another one storing the noise-free coordinates x'_m, y'_m, z'_m .

In a post-processing step a python script is used in order to generate semantic segmentation and LDB annotations from the screenshots and xml files. The semantic segmentation annotations are generated using x_m^p, y_m^p, z_m^p , whereas the LDB annotations are generated using $x_m^{p'}, y_m^{p'}, z_m^{p'}$. In the former case, the projected depth coordinate z_m^p is used to scale the circular stem base annotations between radius values of 18 and 8 pixels. The smaller z_m^p , the bigger the annotation radius. In the latter case, a linear system of equations

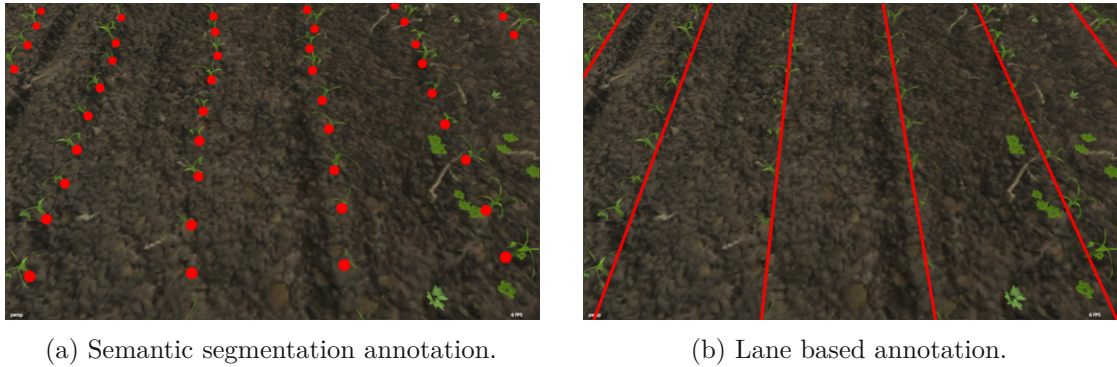


Figure 3.22: Visualization of semantic segmentation and lane based annotations for synthetic datasets.

is defined and numpy’s linear algebra toolbox is utilized to calculate the least squares solution (using `numpy.linalg.lstsq`²). The linear system of equations is constructed in the form

$$\begin{bmatrix} x_{m_0}^{p'} & 1 \\ x_{m_1}^{p'} & 1 \\ x_{m_2}^{p'} & 1 \\ \vdots & \vdots \\ x_{m_n}^{p'} & 1 \end{bmatrix} \begin{bmatrix} k \\ d \end{bmatrix} = \begin{bmatrix} y_{m_0}^{p'} \\ y_{m_1}^{p'} \\ y_{m_2}^{p'} \\ \vdots \\ y_{m_n}^{p'} \end{bmatrix}$$

where n is the number of coordinates belonging to the currently considered line. Performing linear regression using `numpy.linalg.lstsq` and solving for the values k and d gives the least squares solution for the slope k and the intercept d of the line, which is used to generate the LDB annotations. Figure 3.22 shows a visualization of both annotation types, semantic segmentation 3.22a and lane based 3.22b.

Finally, another python script is used to generate object detection annotations from semantic segmentation annotations. As in the case of RD this is done using an OPENCV function called `connectedComponentsWithStats`. Figure 3.23 shows an example visualization for an object detection annotation for a synthetic image.

3.2.4 Texture Editing

Subsections 3.2.1, 3.2.2 and 3.2.3 covered the generation of SD. The SDTE dataset is generated in the same way with the only difference that ground and maize plant textures have been edited.

²`numpy.linalg.lstsq`: <https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html>, accessed on 29.08.23



Figure 3.23: Visualization of an object detection annotation for synthetic datasets.



Figure 3.24: Different textures are manually applied to the field ground in a stacked fashion as shown in this Figure. The used textures can be seen in Figure [3.9](#).

Ground Texture

For ground texture editing, six different textures are applied to the virtual field ground in a stacked fashion as shown in Figure [3.24](#). The ground textures used in this case are provided by FARMING SIMULATOR 22 and shown in Figure [3.9](#). Ground textures are edited for both modes, OM and YM.

Maize Plant Texture

In order to increase the variety of maize plants on the virtual field, the three following approaches are used.



Figure 3.25: Cloud rendering filter in PHOTOSHOP used to generate color variations of existing maize plant textures.

1. 2D maize textures, more specifically the *diffuse* maps projected onto the 3D model, are edited using *rendering filters* in PHOTOSHOP³
2. The 3D models of maize plants are deformed using *deformers* in MAYA⁴.
3. The *UV map* (which is the mapping of a 2D texture onto a 3D model) is edited in MAYA using the *UV Editor*.

PHOTOSHOP offers a feature called *cloud rendering filters*, which allows to generate a random 2D pattern that can be used to apply different colors with varying and randomized intensity on an existing image. In this work, this feature is used to apply different shades of green onto existing maize plant textures in a seemingly natural way. Figure 3.25 shows an example of a cloud rendering filter used to create color variations based on existing diffuse maps from virtual maize plants.

Figure 3.26 shows a comparison of the original diffuse map of both original maize models from FARMING SIMULATOR 22 and an edited version of it using cloud rendering filters, where Figure 3.26a shows the original version and Figure 3.26b shows the edited version with increased intensity for better comparison in this Figure. The texture variation shown in Figure 3.26b uses an especially dark shade of green to imitate dirt on the stem base. Nine different variations are generated this way, which - including the original diffuse map - gives a total number of ten diffuse maps, where each of them contains textures of two maize plants.

A variation of 3D models is created by editing the two original models in MAYA. Figure 3.27 demonstrates how twist (Figure 3.27a) and wave deformers (Figure 3.27b) can be used to vary the shape of the 3D models.

³PHOTOSHOP: <https://www.adobe.com/products/photoshop.html>, accessed on 30.8.23

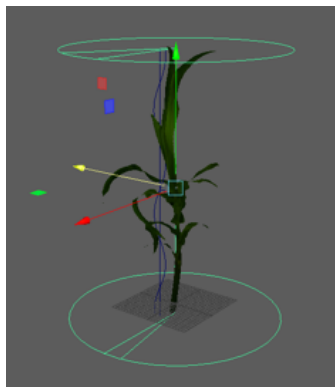
⁴<https://www.autodesk.de/products/maya>, accessed on 30.8.23



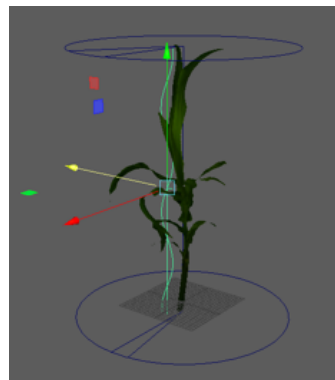
(a) Original diffuse map from FARMING SIMULATOR 22.

(b) Diffuse map edited with cloud rendering filters.

Figure 3.26: Comparison of the original diffuse map from FARMING SIMULATOR 22 and an edited diffuse map using cloud rendering filters in PHOTOSHOP. (The color variation shown in the right Figure is not used in this work. It's variation has increased intensity for better comparison in this Figure.)



(a) The twist deformer in this sub-figure adds increasing rotation to the 3D model around the y axis from bottom to top.



(b) The wave deformer in this sub-figure deforms the 3D model according to a vertical sine wave with adjustable amplitude and frequency.

Figure 3.27: The two MAYA deformers used in this work to generate variations of the original 3D models from FARMING SIMULATOR 22.

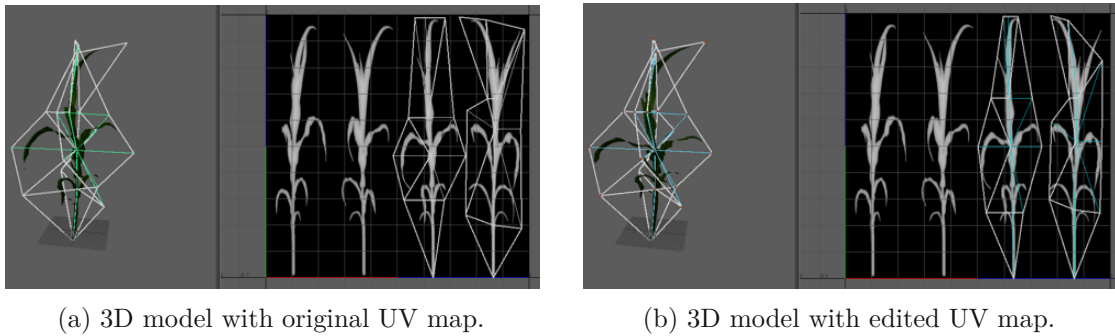


Figure 3.28: MAYA's UV editor used to create variations of the mapping from 2D textures onto 3D models.

Another way of creating variations in the plant models is editing the so-called UV map (see Figure 3.28). UV maps are a technique used to map 2D texture images onto the surfaces of 3D models. By moving the vertices of a UV map one can change the appearance of plant model, which is shown in Figure 3.28b.

Using the approaches described in Figures 3.27 and 3.28, four variations of the original 3D models are generated. Including the two original 3D models from FARMING SIMULATOR 22 this gives six different models in total. Together with the ten different diffuse maps 36 different combinations of 3D models and plant textures are generated, which are used for rendering maize fields in the SDTE dataset for the OM mode. In order to render SDTE maize fields in YM mode, the edited alpha map shown in Figure 3.7b is applied to the 36 texture-edited models.

Evaluation and Results

As mentioned in Section 3.2 three different datasets are generated in this work: RD, SD and SDTE. These three datasets are used to train three different deep learning models:

- PADL (LDB method [FGT+22]),
- PointRend [KWHG19] (semantic segmentation) and
- YOLOv8 [RDGF15] (object detection).

4.1 Dataset Definitions

The RD dataset is generated as described in Section 3.1. It consists of 250 real farmland images that show maize plants in different growth stages. Both synthetic datasets SD and SDTE are generated as described in Section 3.2, where the former uses original textures (as provided by FARMING SIMULATOR 22) and the latter uses textures edited using PHOTOSHOP and MAYA. Table 4.1 shows the numbers of the datasplit in train and test data for RD, SD and SDTE. Figures 4.1, 4.2 and 4.3 shows example images for each of the three datasets respectively.

Dataset	Train Images	Test Images
RD	200	50
SD	1706	426
SDTE	1706	426

Table 4.1: Datasplit in train and test data for RD, SD and SDTE.

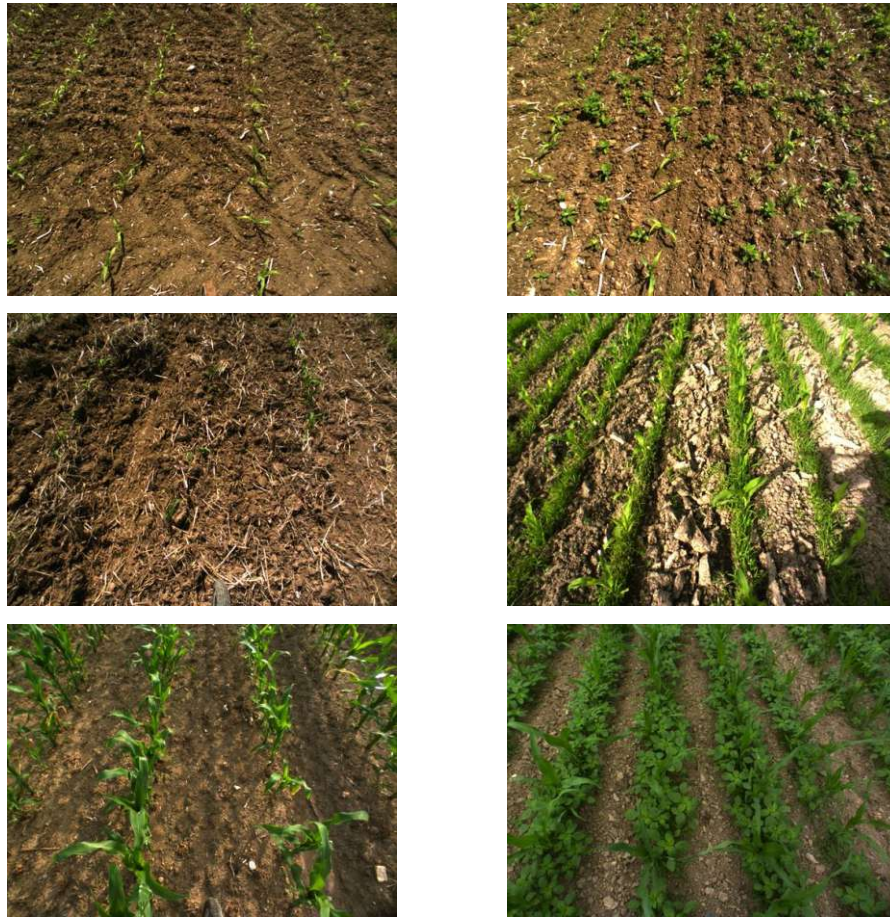


Figure 4.1: Example images from the RD dataset.

The datasets RD, SD and SDTE either contain real farmland images only or synthetic farmland images only. In addition to those three datasets, six more datasets are generated by mixing real and synthetic images:

- 20% real images from RD and 80% synthetic images from SD (called MD20r80s)
- 50% real images from RD and 50% synthetic images from SD (called MD50r50s)
- 80% real images from RD and 20% synthetic images from SD (called MD80r20s)
- 20% real images from RD and 80% synthetic images from SDTE (called MD20r80sTE)
- 50% real images from RD and 50% synthetic images from SDTE (called MD50r50sTE)
- 80% real images from RD and 20% synthetic images from SDTE (called MD80r20sTE)

Table [4.2](#) shows the split of the six mixed datasets into train and test images and also separates these numbers by its original datasets RD, SD and SDTE.

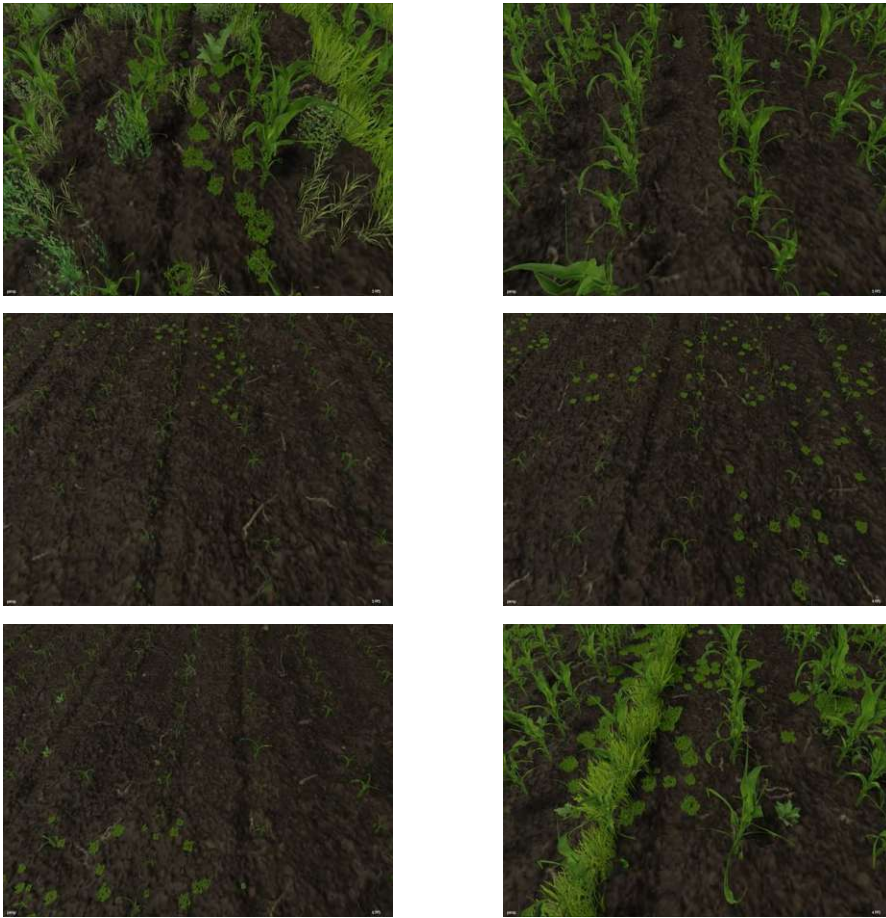


Figure 4.2: Example images from the SD dataset.

4.2 Study of Deep Learning Methods

This section summarizes the research that lead to choosing the CNN techniques used for DeepRow and DeepStem in this work:

- a lane detection based model as implemented in the PYTORCHAUTODRIVE framework [FGT⁺22] (referred to as *PADL*) for DeepRow,
- a semantic segmentation model called *PointRend* [KWHG19] for DeepStem and
- an object detection model called *YOLOv8* [RDGF15] for DeepStem.

4.2.1 Lane Detection Based Method

Feng et al. [FGT⁺22] presented a novel approach for vision based lane detection in the context of autonomous driving and created the open-source framework PYTORCHAUTODRIVE

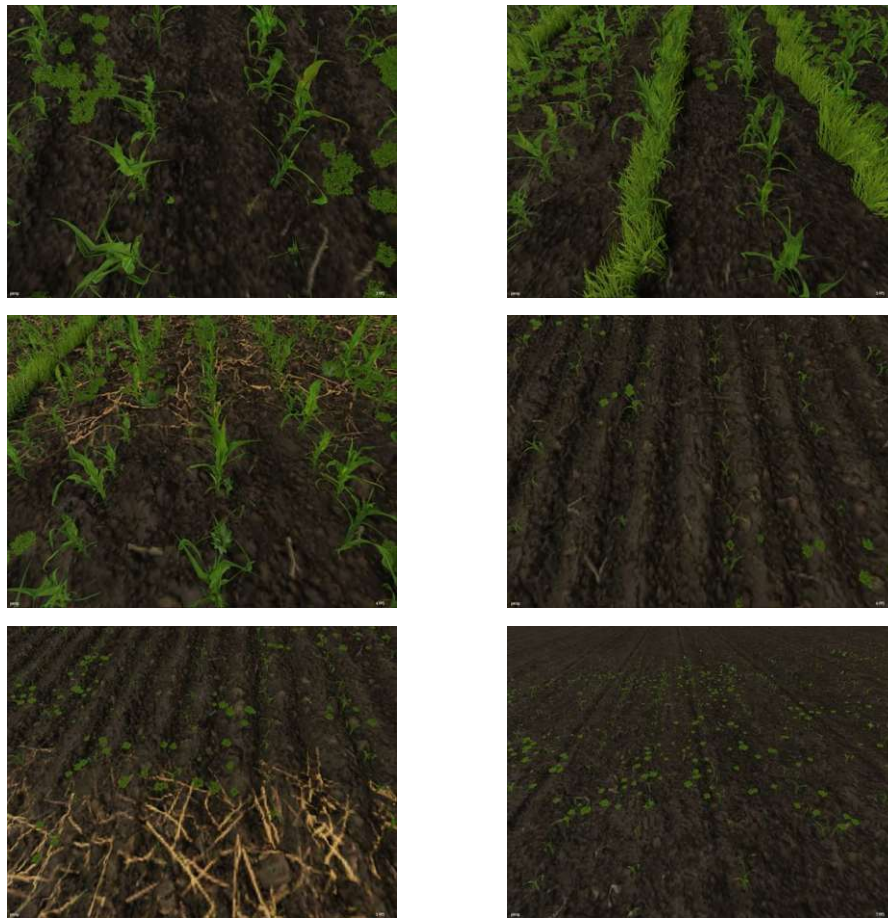


Figure 4.3: Example images from the SDTE dataset.

for lane detection based on their work. Riegler-Nurscher and Rupp [RNR22] examined the capability of various semantic segmentation CNN architectures for crop row detection utilizing PYTORCHAUTODRIVE. The baseline architectures used by Riegler-Nurscher and Rupp use ResNet [HZRS16] as backbone and DeepLab for up-sampling. Their results show that lane detection methods based on PYTORCHAUTODRIVE can successfully be utilized for the task of crop row detection and that especially *Spatial CNN* [PSL⁺17] and *Recurrent Feature Shift Aggregator* [ZFZ⁺20] modules on top of the baseline architecture can improve detection accuracy. The PYTORCHAUTODRIVE architecture with ResNet18 backbone and DeepLabv2 as top-module is used as crop row detection approach in this work and results shall be compared to [RNR22].

4.2.2 PointRend

PointRend was introduced by Kirillov et al. [KWHG19] in 2019. The PointRend CNN module views semantic segmentation as a *rendering problem* and draws analogies to the

Dataset	Train Images			Test Images		
	RD	SD	SDTE	RD	SD	SDTE
MD20r80s	200	800	0	50	200	0
MD50r50s	200	200	0	50	50	0
MD80r20s	200	50	0	50	13	0
MD20r80sTE	200	0	800	50	0	200
MD50r50sTE	200	0	200	50	0	50
MD80r20sTE	200	0	50	50	0	13

Table 4.2: Datasplit for the mixed datasets. In addition to the split in train and test images, the image count is also split by it's original dataset.

problem of over- and under-sampling in computer graphics. Instead of making predictions for every pixel in the image, PointRend does so by selecting specific points in the image and making predictions based on the information at those selected points. These selected points are chosen in a way that adapts to the characteristics of the image. As shown by the authors, the presented approach performs especially well when it comes to accurately detecting object boundaries and masking fine-grained details in images (e.g. human fingers). Since multiple stem bases of maize plants appear as small objects in the dataset images of this work, the capability of PointRend to detect individual stem bases shall be examined in this work.

4.2.3 YOLOv8

Since it's initial publication by Redmon et al. [RDGF15] in 2015, YOLO has established as a state-of-the-art object detection framework with multiple versions since it's inception. The first version YOLOv1 introduced the concept of single-pass object detection. It divides the image into a grid and predicted bounding boxes and class probabilities. However, it has issues with small object detection and precise localization. Since YOLOv1 various improvements like anchor boxes [RF17], batch normalization, several architectural changes (especially for the backbone part of the network), focal loss instead of cross-entropy loss functions (which promises better detection for small object classes) and spatial pyramid pooling [NB09] have been added to the framework, resulting in it's latest version YOLOv8, released by *Ultralytics*¹.

YOLO is a single-shot object detection framework, which means that the processed image is passed through the network only once. In contrast, two-shot architectures like R-CNN [GDDM13] and it's variants pass images through the network twice to refine predictions. While single-shot techniques tend to be faster (in terms inference time),

¹<https://docs.ultralytics.com/>, accessed on 05.09.2023

CNN Model	Number of Epochs
PADL	120
PointRend	120
YOLOv8	100

Table 4.3: Number of epoch for training PADL, PointRend and YOLOv8 models in this work.

two-shot techniques tend to be more accurate. Given the context that this work focuses on real-time application and due to the ease of use of YOLOv8, YOLO is chosen as the object detection method for this thesis. Further, FE-YOLO (a variant of YOLOv3) has successfully been used in a similar application to count maize seedlings on UAV imagery as presented by Zhang et al. [ZFH⁺21].

4.3 Evaluation on Deep Learning Models

All nine datasets are used to train the three deep learning models mentioned above. The metrics used to evaluate each approach are described in Section 2.2.3. The number of epochs for training each CNN model type is chosen individually based on validation loss convergence. These numbers are shown in Table 4.3.

4.3.1 Lane Detection Based Model

The PADL model is evaluated based on the metrics *Accuracy* and *IoU*. In case of this model the ground truth segmentation mask consists of lines with a width of 16 pixels annotating crop rows in the farmland scenes. Table 4.4 shows the results for PADL evaluated on the generated datasets. Each line in the Table lists result for a model that was trained on a specific dataset, where the column *Primary Testset* shows results evaluated on the testset of the respective dataset and the column *RD Testset* shows results of the same model but evaluated on the test images of the RD dataset.

The results in Table 4.4 show that mixed datasets perform best for the task of crop row detection when evaluated on the RD testset. The model trained on the RD dataset shows comparable performance on the RD testset, although the metrics are slightly lower compared to the average mixed dataset. The metrics for the purely synthetic datasets SD and SDTE are comparatively poor when compared to the rest of the generated datasets when evaluated on RD. However, when evaluated on *Primary Testset* the synthetic datasets perform best, which can be explained by the high number of images in the train- and testset for SD and SDTE (see Table 4.1).

Another notable observation is that - among mixed datasets - the dataset MD20r80sTE clearly outperforms the rest of the mixed datasets with an Accuracy of 64.74 and an

Dataset Type	Dataset Name	Primary Testset		RD Testset	
		Accuracy	IoU	Accuracy	IoU
Real	RD	59.36	38.13	59.36	38.13
Synthetic	SD	78.58	54.13	8.66	7.53
	SDTE	77.83	52.28	17.73	14.06
Mixed	MD20r80s	73.25	48.44	61.36	39.57
	MD50r50s	67.47	42.94	61.18	39.01
	MD80r20s	61.79	39.35	62.43	39.54
	MD20r80sTE	71.82	46.12	64.74	40.57
	MD50r50sTE	62.01	38.51	61.03	38.35
	MD80r20sTE	59.56	37.40	61.70	38.69

Table 4.4: Results of the PADL model evaluated on the generated datasets. The column *Primary Testset* contains results of the model evaluated on the respective test images of the dataset and the column *RD Testset* shows results for the same model but evaluated on the RD testset. **Note:** Values in this table are multiplied by 100.

IoU of 40.57. MD20r80sTE is a dataset containing 20% real images and 80% synthetic images from SDTE. Additionally, when comparing the metrics of datasets without texture editing, MD20r80s performs slightly better than MD50r50s and MD80r20s. Based on this, it can be concluded that both, a high percentage in synthetic images and texture editing contribute positively to the performance of the investigated LDB model, when it comes to detecting crop rows on real farmland.

In this context, it should also be noted that there is an inconsistency when comparing the metrics for MD50r50s, MD80r20s, MD50r50sTE and MD80r20sTE. MD80r20s and MD80r20sTE (the mixed datasets with lowest percentage of synthetic images) show a marginal increase in performance compared to MD50r50s and MD50r50sTE respectively, when evaluated on RD. This means that - in that specific case - a higher percentage of synthetic images slightly lowers the metrics, although the difference is marginal. However, one might explain this inconsistency by the fact that the number of synthetic train images in MD50r50s and MD50r50sTE (200) is only four times higher as compared to MD80r20s and MD80r20sTE (50), whereas the number of synthetic train images is sixteen times higher when comparing MD20r80s and MD20r80sTE (800) to MD80r20s and MD80r20sTE (see Table 4.2). The synthetic images added to the mixed datasets are picked randomly from SD and SDTE. Hence, the performance of models trained on mixed datasets also depends on which specific synthetic images have been selected. It should also be noted that approximately 1% of variation in metrics is due to the random selection of images for each batch when training the network. Repeating these

experiments with a higher number of synthetic images is necessary to investigate this inconsistency in more detail.

The results shown in Table 4.4 can be compared to the results of the baseline implementation in the work by Riegler-Nurscher and Rupp [RNR22]. As in this work, the authors use a CNN architecture with ResNet18 backend and a DeepLab network for up-sampling as provided by the PYTORCHAUTODRIVE framework. The most performant variant on real farmland images in this work achieves an Accuracy of 64.74 and IoU of 40.57. In [RNR22] the respective reference model achieves an Accuracy of 66.83 and IoU of 44.07, meaning the reference model outperforms the most performant PADL model in this work. However, the discrepancy in performance can partially be explained by the larger dataset used in [RNR22]. The dataset used by the authors consists of 3475 real farmland images for training and 395 for testing.

Figure 4.4 shows example evaluations of PADL models trained on RD, SD and MD20r80sTE. The intensity of the annotated lines indicates the certainty of the respective model. All models shown in Figure 4.4 show passable results for the first example image (first row). The image shown in the second row is seemingly the hardest example in terms of accurate crop row detection. The model trained on RD does not detect crop rows in the right half of that image. Interestingly, the SDTE model shows high confidence values and seemingly accurate results for those crop rows. In contrast, the SDTE model clearly shows the poorest performance for the example image in the third row.

4.3.2 PointRend

Table 4.5 shows results for the PointRend model. The metrics used to evaluate this model are the F1 score and IoU. Like PADL, the PointRend model is trained on each of the nine generated datasets and metrics are calculated on the testset of the respective dataset and on the RD testset.

As in the case of the PADL model, the PointRend models trained on purely synthetic datasets perform best on the testset of their respective datasets, but show the poorest performance on the RD testset. Again, the former circumstance can be explained by the high total image numbers of SD and SDTE, whereas the latter circumstance is attributed to the fact that SD and SDTE do not contain real images, which means that the respective CNNs are confronted with real farmland images for the first time at inference on RD.

The best performance is shown by PointRend models trained on RD. Models trained on mixed datasets show a decrease in performance as the percentage of synthetic images increases. With regard to texture editing it can be said that the performance tends to decrease further when images from SDTE instead of SD are used. The only exception in that context can be seen when comparing results for MD80r20s and MD80r20sTE, where metrics only differ in the second decimal place.

Example evaluations of the PointRend model trained on RD, SDTE and MD20r80sTE are shown in Figure 4.5. With regard to the results shown in the first row it can be

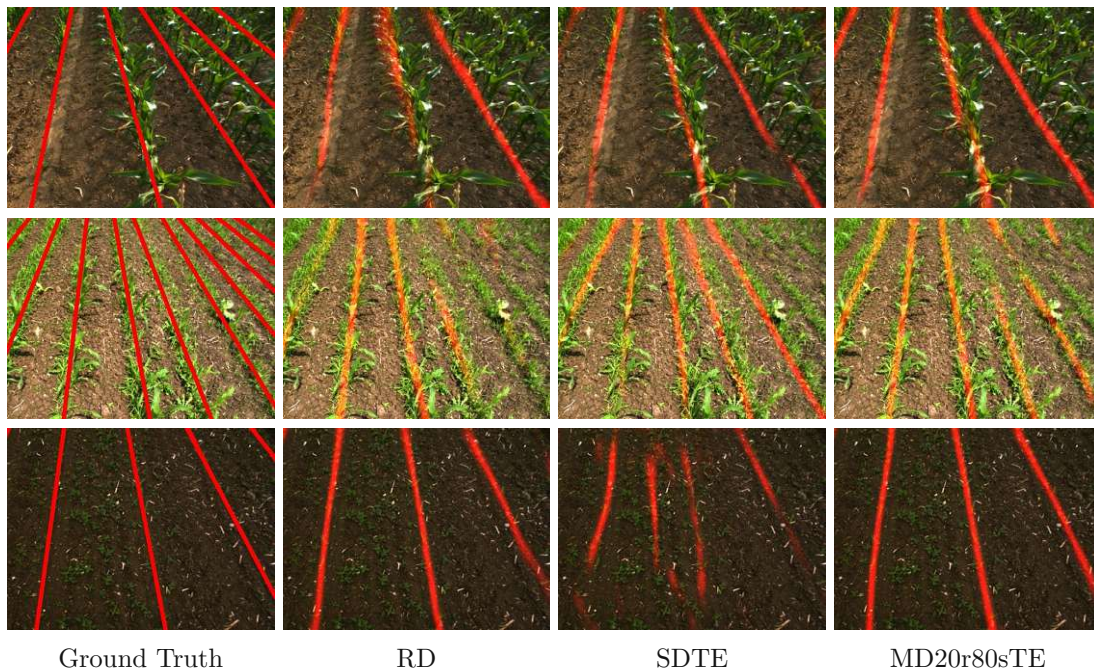


Figure 4.4: Evaluation examples of the PADL model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE. The intensity of the annotated lines indicates the certainty of the model.

said that both, the RD and the MD20r80sTE model accurately detect some of the stem bases, especially to the left and right of the central crop row. The models clearly have difficulties detecting stem bases in the central row, since stems in that row are partially covering each other. It should also be noted that the RD and MD20r80sTE results in the last row of the Figure indicate that the models are capable of distinguishing young maize plants from weeds. In general, the SDTE model seems over-confident and produces too many cluttered annotations.

4.3.3 YOLOv8

As described in Section 2.2.3, the Average Precision (AP) metric is used for the evaluation of YOLOv8 models. Table 4.6 shows AP values for the resulting models trained on each of the generated datasets. The listed AP values are given for a confidence-threshold of 0.5.

Table 4.6 shows that the YOLOv8 model trained on RD shows the best performance on the RD testset, followed by MD80r20sTE with a marginal difference in AP. As in the case of PADL and PointRend results, the models trained on SD and SDTE clearly show the highest metrics evaluated on their respective testset (*Primary Testset*), but the poorest performance evaluated on the RD testset. Like the PointRend metrics (Table

Dataset Type	Dataset Name	Primary Testset		RD Testset	
		F1	IoU	F1	IoU
Real	RD	50.30	33.60	50.30	33.60
Synthetic	SD	82.06	69.58	13.66	7.33
	SDTE	79.55	66.03	17.40	9.53
Mixed	MD20r80s	75.52	60.66	39.49	24.61
	MD50r50s	64.13	47.20	46.44	30.24
	MD80r20s	56.71	39.58	48.38	31.91
	MD20r80sTE	71.91	56.14	38.72	24.01
	MD50r50sTE	62.98	45.97	45.04	29.07
	MD80r20sTE	56.36	39.24	48.40	31.93

Table 4.5: PointRend results: The column *Primary Testset* contains results of the model evaluated on the respective test images of the dataset and the column *RD Testset* shows results for the same model but evaluated on the RD testset. **Note:** Values in this table are multiplied by 100.

Dataset Type	Dataset Name	AP (Primary Testset)	AP (RD Testset)
Real	RD	31.9	31.9
Synthetic	SD	90.6	1.3
	SDTE	87.5	1.1
Mixed	MD20r80s	80.1	28.7
	MD50r50s	59.2	29.4
	MD80r20s	44.8	29.8
	MD20r80sTE	76.9	28.5
	MD50r50sTE	58.7	26.9
	MD80r20sTE	46.5	31.8

Table 4.6: YOLOv8 results: AP values at a confidence-threshold of 0.5. The column *AP (Primary Testset)* contains results of the model evaluated on the respective test images of the dataset and the column *AP (RD Testset)* shows results for the same model but evaluated on the RD testset. **Note:** Values in this table are multiplied by 100.

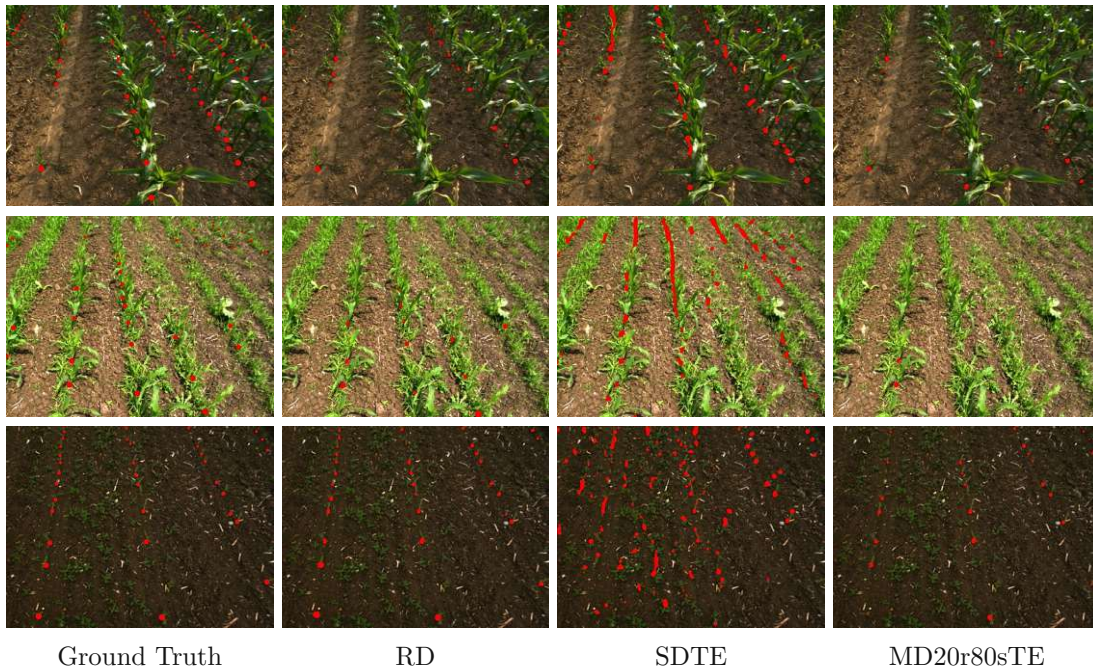


Figure 4.5: Evaluation examples of the PointRend model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE.

4.5), the AP metric tends to decrease with increasing percentage of synthetic image. Again, performance tends to decrease even more when synthetic images from SDTE instead of SD are used. One exception in this context can be seen for the MD80r20sTE model in the *AP (RD Testset)* column, where AP indicates comparable performance to the RD model. This inconsistency might be explained by the low percentage (20%) of synthetic images in MD80r20sTE. Hence, the results of this model highly depend on the random selection of images from SDTE. However, in order to give a definite answer in this context, further investigation is necessary.

Figure 4.6 shows example evaluations of YOLOv8 trained on RD, SDTE and MD20r80sTE. As in the case of PointRend evaluations (Figure 4.5), results in the first row indicate that the RD and MD20r80sTE models have difficulties accurately detecting stem bases in the central row. Results in the second row show that all models have difficulties detecting stem bases when they are covered by in-row weeds. The evaluations on the last example image (last row) show that - like the PointRend models - YOLOv8 models are capable of distinguishing young maize plants from weeds. At least for the third example image, none of the given models confuses maize plants with weeds. This is especially worth noting for the SDTE case, since PADL and PointRend models trained on SDTE usually seem to be over-confident and produce false positives.

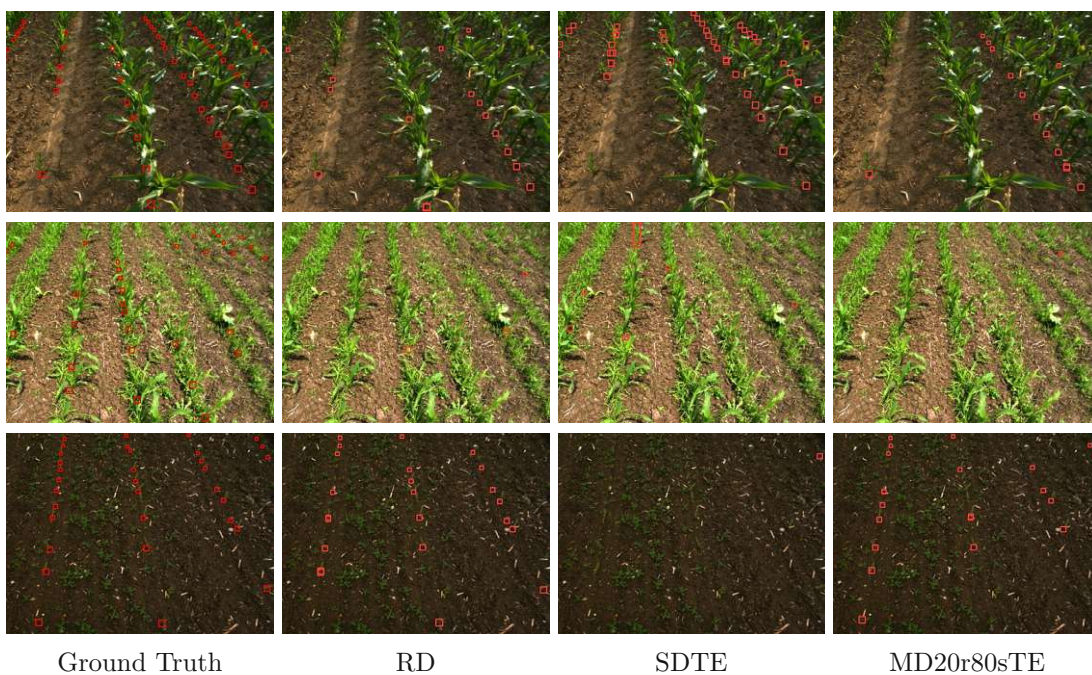


Figure 4.6: Evaluation examples of the YOLOv8 model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE.

Conclusion

The presented methods DeepRow and DeepStem examined the capability of synthetic farmland images generated from the computer game FARMING SIMULATOR 22 to increase detection accuracy of CNN-based approaches for crop row detection and the detection of individual stem bases on farmland. Depending on the type of dataset the models are trained on, the CNN methods are referred to as *Real-*, *Synthetic-* and *Mixed-*DeepRow or DeepStem respectively.

5.1 Results

Comparing the results of all nine datasets for crop row detection showed that the average Mixed-DeepRow model outperformed Synthetic-DeepRow and Real-DeepRow, where the model trained on MD20r80sTE (20% real and 80% texture-edited synthetic images) performed best when evaluated on the RD testset. More specifically, it was shown that the PADL performance metrics *Accuracy* and *IoU* increased by 9.06% and 6.4% respectively, compared to the model trained on RD. An inconsistency showed up when comparing the PADL metrics for MD50r50s, MD80r20s, MD50r50sTE and MD80r20sTE. MD80r20s and MD80r20sTE showed a marginal increase in performance compared to MD50r50s and MD50r50sTE respectively, when evaluated on RD, which indicates that - in this case - a lower percentage of synthetic images increases performance.

When evaluating individual stem base detection models, it was observed that Real-DeepStem outperformed Synthetic-DeepStem and Mixed-DeepStem when assessed using RD test images. For both, PointRend and YOLOv8, performance metrics for the RD testset decreased with increasing percentage of synthetic images. With a few exceptions, the metrics tended to decrease further when using synthetic images from SDTE instead of SD. In the case of PointRend the best Mixed-DeepStem approach (trained on MD80r20sTE) yielded a decrease of 3.78% and 4.97% for F1 score and IoU respectively. Evidently, PointRend models trained on purely synthetic datasets showed

overconfidence in detecting stem bases, which is apparent from the cluttered detection results in Figure 4.5 (SDTE column). For YOLOv8, one exception to this trend was the model trained on MD80r20sTE, which showed similar performance to the model trained on RD.

One possible reason for the variability in results observed among models trained on mixed datasets is the random selection of images from SD and SDTE. In the case of models trained on MD80r20s and MD80r20sTE, which only contain 50 synthetic images, the performance outcome highly depends on the specific selection of synthetic images. Additionally, it's noteworthy that a minor variation of approximately 1% in inference metrics is attributed to the random selection of images for each batch while training the network. However, to properly address these inconsistencies, further investigations are necessary, including experiments conducted with more extensive datasets.

Challenges in the implementation of the DeepStem approach became evident during the visualization of inference examples for PointRend and YOLOv8. Examination of the first two rows of images in Figures 4.5 and 4.6 reveals that both DeepStem approaches encounter difficulties in accurately detecting stem bases within the central crop row of each image. This can be attributed to the challenge presented by partial occlusion of stem bases, particularly evident in images of plants aged approximately 25 days and older. This occlusion makes annotating stem bases in RD images more demanding and consequently hinders accurate detection on real farmland images.

Based on these findings, it can be concluded that generating synthetic datasets that enhance the performance of DeepRow models is a relatively simpler task compared to DeepStem models. This difference can be attributed to the fact that stem base detection is a task which relies more heavily on capturing texture details in an image, compared to the task of crop row detection.

5.2 Future Work

Virtual 3D models of young maize plants (see Figure 3.8b) posed a specific challenge, which primarily arose from the thinness of their leaves when viewed from above. The thinness of these leaves became an issue due to their important role in the accurate detection of crop rows and the identification of stem bases. This issue could be addressed by refining the 3D models of maize plants in future research.

As previously mentioned, one of the challenges encountered when annotating stem bases in real farmland images is the partial occlusion of these bases. An alternative approach could opt to annotate only non-overlapping plant stems. However, for synthetic images, a notable challenge would arise from determining which plant stems are overlapping and which are not.

In order to generate more realistic synthetic images, the *SimGAN* approach proposed by Shrivastava et al. [SPT⁺16] could be utilized. In this approach, a Discriminator and

a Refiner network collaboratively generate improved versions of the original synthetic images by leveraging both real and synthetic images as references.

As a general direction for future research, it is advisable to repeat the experiments outlined in this work using larger datasets. This approach aims to address discrepancies in performance metrics, as described in Section [5.1](#).

List of Figures

1.1	Maize hoeing with camera control. (Source: https://www.einboeck.at , accessed on 12.10.2022)	2
1.2	Maize plants on a field at different growth stages and different levels of weed coverage.	3
1.3	Ground truth visualization of object detection for maize plant stem bases. Bounding boxes are shown in red.	4
2.1	Subfields of Artificial Intelligence.	8
2.2	Example for Object Detection. (Source: https://towardsdatascience.com , accessed on 04.08.2023)	9
2.3	Example for Semantic Segmentation, (Source: https://www.superannotate.com , accessed on 04.08.2023)	10
2.4	Calculation of Intersection over Union, (Source: https://www.superannotate.com , accessed on 04.08.2023)	12
2.5	Example of a Precision-Recall-Curve for a single object class. (Source: https://github.com/ultralytics , accessed on 18.08.2023)	13
2.6	Synthetic image of a maize field from FARMING SIMULATOR 22	14
2.7	Screenshot GIANTS EDITOR	15
3.1	Example visualization of a semantic segmentation annotation.	20
3.2	Screenshot of the LABELIMAGE user interface.	20
3.3	YOLO object detection format. (Source: https://docs.ultralytics.com/ , accessed on 17.08.2023)	21
3.4	Example visualization of an object detection annotation.	22
3.5	Example visualization of a LDB annotation.	23
3.6	A FARMING SIMULATOR 22 maize model with different texture colors.	24
3.7	Alpha map editing of a FARMING SIMULATOR 22 maize model.	24
3.8	A maize plant model with two different versions of alpha maps.	25
3.9	FARMING SIMULATOR 22 ground texture types.	26
3.10	Maize dropout segments due to sub optimal conditions on the farmland or while seeding.	27

3.11 A fourth order polynomial (blue) is used to model maize dropout segments. The plant height s_{y_m} is scaled according to the polynomial. In addition to polynomial height scaling, random scaling is added according to $N(0, \sigma_{s_{y_m}})$. Maize plants are shown in green. Maize plants with $y_m < y_{min}$ are not rendered.	30
3.12 FARMING SIMULATOR 22's clover model.	32
3.13 Example results for <i>cellular_automaton(30, 30, 5, 50)</i> .	32
3.14 Clover distribution on the virtual field. Rendered clover plants are highlighted in green.	33
3.15 FARMING SIMULATOR 22's rumex model.	34
3.16 Grass models provided by FARMING SIMULATOR 22.	37
3.17 Statically rendered weed plants in order to increase weed plant diversity and density in the left half of the virtual field. Used in OM only.	39
3.18 Concentrated grass segments within the crop row due to hoeing in previous growth stages.	39
3.19 <i>Play</i> button in GIANTS EDITOR to start periodic execution of registered update listener functions.	40
3.20 Camera movement paths (blue arrows) when generating a new dataset in GIANTS EDITOR. The numbers next to the camera paths represent the order in which the paths are followed. The green dot indicates the origin coordinates of a virtual maize field x_{FO} and z_{FO} , whereas the red dot indicates the initial position of the camera node.	40
3.21 Side view of the perspective projection used to transform 3D stem base coordinates to 2D image plane coordinates. In addition to the screen coordinates x_m^p and y_m^p , the projection also outputs a depth coordinate z_m^p , which equals the distance of the 3D point to the image plane. The projected coordinates are normalized to the interval $[0, 1]$. The normalization in z_m^p is performed with respect to the <i>near clipping plane</i> and the <i>far clipping plane</i> . (Source: http://learnwebgl.brown37.net , accessed on 29.08.2023)	41
3.22 Visualization of semantic segmentation and lane based annotations for synthetic datasets.	42
3.23 Visualization of an object detection annotation for synthetic datasets.	43
3.24 Different textures are manually applied to the field ground in a stacked fashion as shown in this Figure. The used textures can be seen in Figure 3.9.	43
3.25 Cloud rendering filter in PHOTOSHOP used to generate color variations of existing maize plant textures.	44
3.26 Comparison of the original diffuse map from FARMING SIMULATOR 22 and an edited diffuse map using cloud rendering filters in PHOTOSHOP. (The color variation shown in the right Figure is not used in this work. It's variation has increased intensity for better comparison in this Figure.)	45
3.27 The two MAYA deformers used in this work to generate variations of the original 3D models from FARMING SIMULATOR 22.	45

3.28	MAYA's UV editor used to create variations of the mapping from 2D textures onto 3D models.	46
4.1	Example images from the RD dataset.	48
4.2	Example images from the SD dataset.	49
4.3	Example images from the SDTE dataset.	50
4.4	Evaluation examples of the PADL model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE. The intensity of the annotated lines indicates the certainty of the model.	55
4.5	Evaluation examples of the PointRend model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE.	57
4.6	Evaluation examples of the YOLOv8 model trained on different datasets on real farmland images. The first column shows the ground truth annotations, whereas the following columns show results for training on RD, SDTE and MD20r80sTE.	58

List of Tables

3.1 Description of maize rendering variables.	28
3.2 General rendering rules for generating maize fields in GIANTS EDITOR. <i>Note:</i> getTerrainHeight (x, z) is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.	29
3.3 Rendering rules for generating maize fields in GIANTS EDITOR applied specif- ically in OM mode.	29
3.4 Random rules for rendering maize plants in GIANTS EDITOR applied specif- ically in YM mode.	29
3.5 Description of clover rendering variables.	33
3.6 Rendering rules for rendering clover plants in GIANTS EDITOR. <i>Note:</i> getTerrainHeight (x, z) is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.	34
3.7 Description of rumex rendering variables.	35
3.8 Rendering rules for rendering rumex plants in GIANTS EDITOR. <i>Note:</i> getTerrainHeight (x, z) is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.	36
3.9 Description of grass rendering variables.	37
3.10 Rendering rules for rendering grass plants in segments in GIANTS EDITOR. <i>Note:</i> getTerrainHeight (x, z) is a giants engine function, which takes x and z coordinates as input and returns the (virtual) terrain height at the given coordinates.	38
4.1 Datasplit in train and test data for RD, SD and SDTE.	47
4.2 Datasplit for the mixed datasets. In addition to the split in train and test images, the image count is also split by it's original dataset.	51
4.3 Number of epoch for training PADL, PointRend and YOLOv8 models in this work.	52
4.4 Results of the PADL model evaluated on the generated datasets. The column <i>Primary Testset</i> contains results of the model evaluated on the respective test images of the dataset and the column <i>RD Testset</i> shows results for the same model but evaluated on the RD testset. Note: Values in this table are multiplied by 100.	53
	67

4.5	PointRend results: The column <i>Primary Testset</i> contains results of the model evaluated on the respective test images of the dataset and the column <i>RD Testset</i> shows results for the same model but evaluated on the RD testset. Note: Values in this table are multiplied by 100.	56
4.6	YOLOv8 results: AP values at a confidence-threshold of 0.5. The column <i>AP (Primary Testset)</i> contains results of the model evaluated on the respective test images of the dataset and the column <i>AP (RD Testset)</i> shows results for the same model but evaluated on the RD testset. Note: Values in this table are multiplied by 100.	56

Bibliography

- [BHC18] M Dian Bah, Adel Hafiane, and Raphael Canals. Deep learning with unsupervised data labeling for weed detection in line crops in uav images. *Remote Sensing*, 10(11), 2018.
- [BPSG04] R. Bennett, R. Phipps, A. Strange, and P. Grey. Environmental and human health impacts of growing genetically modified herbicide-tolerant sugar beet: A life-cycle assessment. *Plant Biotechnology Journal*, 2(4):273–278, 2004.
- [CPK⁺15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.
- [CPK⁺16] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.
- [CPK⁺18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Rethinking atrous convolution for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.
- [EUD22] A european green deal. https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal_en, Accessed on 19.10.2022.
- [FGT⁺22] Zhengyang Feng, Shaohua Guo, Xin Tan, Ke Xu, Min Wang, and Lizhuang Ma. Rethinking efficient lane detection via curve modeling, 2022.

- [Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.
- [FXLX14] D. Fu, L. Xu, D. Li, and L. Xin. Automatic detection and segmentation of stems of potted tomato plant using kinect. In *Proceedings of SPIE - The International Society for Optical Engineering*, volume 9159, 2014.
- [GDDM13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- [HBMO14] Sebastian Haug, Peter Biber, Andreas Michaels, and Jörn Ostermann. Plant stem detection and position estimation using machine vision. 2014.
- [Hou62] P.V.C Hough. A method and means for recognizing complex patterns, U.S. Patent 3,069,654, December 18, 1962.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [JKDC08] Guoquan Jiang, X. Ke, S. Du, and J. Chen. Detection algorithm of crop rows based on machine vision and randomized method. *Nongye Jixie Xuebao/Transactions of the Chinese Society of Agricultural Machinery*, 39:85–88+93, 2008.
- [KJ12] Sajad Kiani and Abdolabbas Jafari. Crop detection and positioning in the field using discriminant analysis and neural networks based on shape features. *Journal of Agricultural Science and Technology*, 14:755–765, 2012.
- [KWHG19] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross B. Girshick. Pointnet: Image segmentation as rendering. *CoRR*, abs/1912.08193, 2019.
- [LBG⁺16] H. Liu, S. Blagodatsky, M. Giese, F. Liu, J. Xu, and G. Cadisch. Impact of herbicide application on soil erosion and induced carbon loss in a rubber plantation of southwest china. *Catena*, 145:180–192, 2016.
- [LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.
- [MB64] G. Marsaglia and T. A. Bray. A convenient method for generating normal variables. *SIAM Review*, 6(3):260–264, 1964.
- [MHW⁺20] A. R. Main, M. L. Hladik, E. B. Webb, K. W. Goyne, and D. Mengel. Beyond neonicotinoids – wild pollinators are exposed to a range of pesticides while foraging in agroecosystems. *Science of the Total Environment*, 742, 2020.

- [NB09] Heiko Neumann and Joachim M. Buhmann. Efficient svm training using low-rank kernels. In *Proceedings of the 12th International Conference on Computer Vision (ICCV)*. IEEE, 2009.
- [PFLCO⁺21] J. C. Piñar Fuentes, F. Leiva, A. Cano-Ortiz, C. M. Musarella, R. Quinto-Canas, C. J. Pinto-Gomes, and E. Cano. Impact of grass cover management with herbicides on biodiversity, soil cover and humidity in olive groves in the southern iberian. *Agronomy*, 11(3), 2021.
- [Pie10] P. J. Pieterse. Herbicide resistance in weeds—a threat to effective chemical weed control in south africa. *South African Journal of Plant and Soil*, 27(1):66–73, 2010.
- [PL90] P. Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.
- [PSL⁺17] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial CNN for traffic scene understanding. *CoRR*, abs/1712.06080, 2017.
- [RDGF15] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [RF17] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2017.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [RNR22] Peter Riegler-Nurscher and Leopold Rupp. Crop row detection utilizing spatial cnn modules. *OAGM Workshop 2022*, pages 77–79, October 2022.
- [SHE⁺21] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes: Synthetic data takes flight. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 207–217, 2021.
- [SLS16] A. Shafaei, J. J. Little, and M. Schmidt. Play and learn: Using video games to train computer vision models. In *British Machine Vision Conference 2016, BMVC 2016*, volume 2016-September, pages 26.1–26.13, 2016.
- [SPT⁺16] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016.

- [SSMM20] Gayle J. Somerville, Mette Sønderskov, Solvejg Kopp Mathiassen, and Helen Metcalfe. Spatial modelling of within-field weed populations; a review. *Agronomy*, 10(7), 2020.
- [ZFH⁺21] Hongming Zhang, Zhenyu Fu, Wenting Han, Guang Yang, Dangdang Niu, and Xinyu Zhou. Detection method of maize seedlings number based on improved yolo. *Nongye Jixie Xuebao/Transactions of the Chinese Society for Agricultural Machinery*, 52(4):221 – 229, 2021.
- [ZfZ⁺20] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, and Deng Cai. RESA: recurrent feature-shift aggregator for lane detection. *CoRR*, abs/2008.13719, 2020.
- [ZLZ⁺18] Xiya Zhang, Xiaona Li, Baohua Zhang, Jun Zhou, Guangzhao Tian, Yingjun Xiong, and Baoxing Gu. Automated robust crop-row detection in maize fields based on position clustering algorithm and shortest path method. *Computers and Electronics in Agriculture*, 154:165–175, 2018.