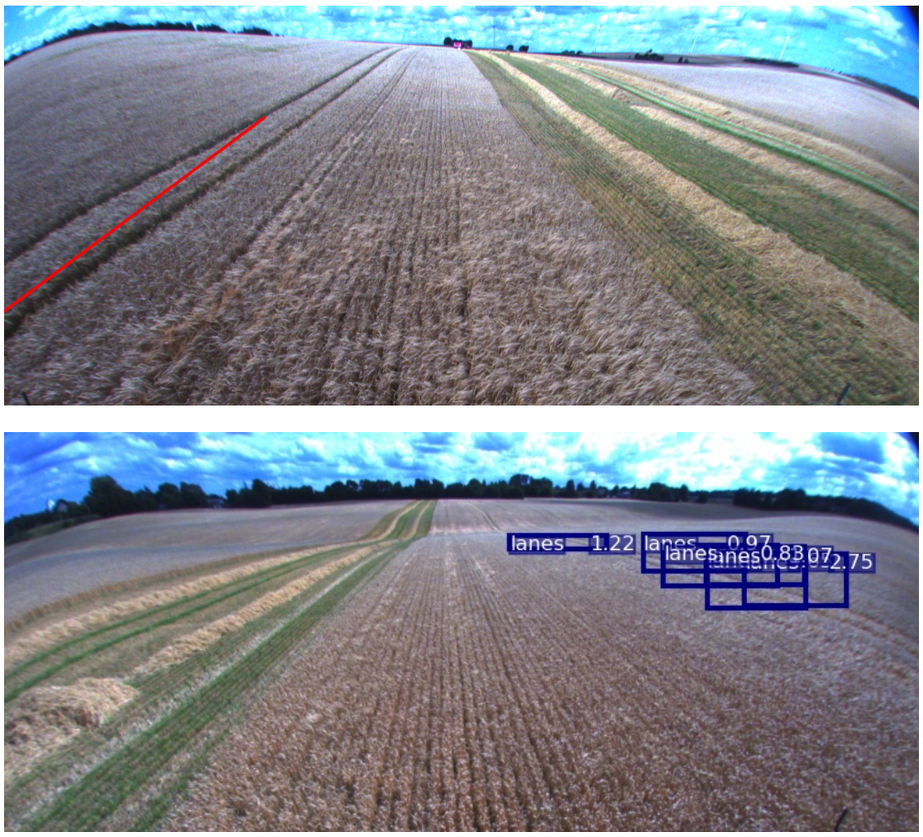
Explorative Comparison Between Classic Computer Vision Techniques and Deep Learning



Project Report

Group 1067

Aalborg University
Robotics Msc.

This report was written in the ShareLaTeX website using the LaTeXtypesetting format and the AAU report template v. 1.2.0. The citations are written in the IEEE format.

Electronics and IT
Aalborg University
https://www.aau.dk/

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Explorative Comparison Between Classic Computer Vision Techniques and Deep Learning

Theme:
Comparing performance of computer vision approaches in dynamic outdoor sceneries

Project Period:
Spring Semester 2021

Project Group:
1067

Participant(s):
Kristian Hyttel Pedersen

Supervisor(s):
Thomas B. Moeslund

Copies: 1

Page Numbers: 25

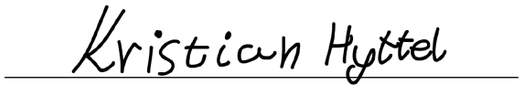
Date of Completion:
2nd June 2021

Abstract:

In order to autonomously control agricultural vehicles, one potential solution could be to follow driving lanes left by tractors, as these are almost always present and permanent inside fields. This is because the simple use of limiting driving to permanent lanes increases yield by an average of $5 - 15\%$. Different sensors can be used to follow these driving lanes, in this project however, a traditional RGB camera was used. A comparison of the performance of a classic computer vision system and a neural network's ability to find driving lanes on agricultural fields, is given. Although this given use-case is fairly niche, the same logic could apply to other areas of expertise. A complete classic computer vision system was developed from scratch, while the neural network is an implementation of YOLOv3. Both of the approaches yield decent results and they are both able to find driving lanes. The classic computer vision system almost always provides a result which could be used to guide an agricultural vehicle. However, this system is more prone to faulty outputs than the neural network. The neural network often does not predict anything inside the inference images, although, it almost never makes false predictions. From this comparison, it was concluded that a classic computer vision system is more appropriate to use given a small time frame or if the implementation serves as a proof of concept. The neural network approach, given more training or a larger dataset, would be better suited for a large scale project because of the more accurate results.

Contents

Author



Kristian Hyttel Pedersen
khpe16@student.aau.dk

Chapter 1

Introduction

In order to autonomously drive around fields without the need for constant realignment from a human operator, many different approaches can be taken. As cameras become cheaper and better, they are also used more often in agriculture. Currently on a system at AGCO in their Research and Development Facility in Randers, they have 5 separate cameras install on a single combine harvester. Four of which are traditional RGB-cameras and a single infrared camera. Alongside this, a plethora of other sensors are used to gather information, some of which could also be used to control a vehicle autonomously. For example, a Differential Global Positioning System (DGPS) could be used to repeatedly find the location of the vehicle and an Inertial Measurement Unit (IMU) could be used to correct for drifting which DGPS is prone to suffer from. The problem with systems like this one is that a lot of prerequisite work must be done in order to figure out how the vehicle should be moving within the bounds of the field. In order for a system using DGPS to work, the location of each turn must be known. By finding a more universal method, the system could work on any field, with only a small amount of alteration, if any. One such method could be to make use of driving lanes which are already present in the field. Driving lanes are a result of agricultural vehicles driving on top of the crop in distinct lanes in order to destroy as little of the planted crops as possible. The lessened destruction of crop directly results in economical benefit, which is one of the reasons why predetermined driving lanes are widely used. As driving lanes are so widely used on my fields, it makes sense to attempt to utilize this fact in order to control agricultural vehicles. A way to do this could be to use an RGB-D camera and the resulting depth map which is generated, to look for "holes" in the crops which would indicate the location of driving lanes these "holes" could then be followed. Another approach which makes use of a traditional RGB camera could be to take an image of the field directly in front of the vehicle use classic computer vision algorithms to identify the location of driving lanes. An interesting aspect of this particular approach is that it could be compared to a neural network which is provided the same RGB input images. This way, both the classic approach and the neural network would be evaluated on their ability to identify driving lanes in a chaotic, ever-changing environment. This comparison could also be of value to companies which are interested in automating systems which currently require a human operator in order to follow any type of line using RGB cameras.

Chapter 2

Problem Analysis

Driving lanes are used on agricultural fields because it reduces the amount of damage which is caused to crops by driving over them. Driving lanes also reduces the number of problems associated with packaging, amount of fuel needed to cultivate the field and reduces the amount of overlapping caused by farmers unknowingly covering the same area of a field multiple times. As a result of all of these factor effectiveness is greatly improved. The overall improvement in yield after harvesting using driving lanes is $5 - 15\%$. The largest increase in yield is seen from crops which require sensitive packaging, such as, potatoes.[1] These are the reasons why driving lanes are so widely used in agriculture and why they are almost always present on fields. While finding lines in an image is not a difficult problem to solve, the challenge in doing this on fields, stem from the "chaotic" nature of the environment. In order to improve the results which are gathered using computer vision techniques, it is common practice to try to optimize lighting conditions and to keep the scenery in the images as consistent as possible. When working on application which are meant to be used outside from many hours per day, the lighting conditions change constantly throughout the day. A similar case could be the problem of finding driving lanes on paved roads. In this case, many of the proposed solutions make use of the fact that the separating lines between lanes are a distinctly different color compared to the rest of the road, most commonly white [2]. [2] uses an input image which can be seen on Figure 2.1. As a reference to this, it can be said that the separating lines on paved roads follow a set of rules, which can be exploited. However, when employing computer vision techniques in an outdoor environment where no clear set of rules are defined, aside from the fact that the driving lanes are still straight, the relatively simple task of finding lines, becomes much more challenging.

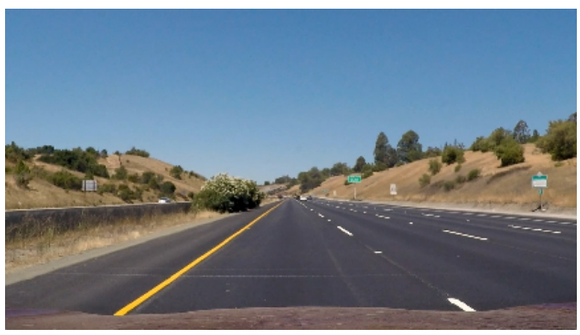


Figure 2.1: Input which is used to identify driving lanes on paved roads.[2]

An example of the images which are available for this project can be seen on Figure 2.2. On this particular image, the driving lanes are clearly visible on the left part of the image. However, it can also be seen that many other lines have a very similar appearance; there are spaces between the individual lines of crop and the right side of the image has been harvested which includes many new lines.

Figure 2.2: Example of field with many straight lines, which look similar to driving lanes.

To decide whether a classic computer vision system or a neural network achieves the best results when finding driving lanes on fields, it makes sense to compare the developed classic computer vision approach to an already established neural network to reduce the potential of contaminating the results with a poorly constructed neural network. Two different types object detection neural network which could potentially be used to identify driving lanes on fields are pixel-wise detection and bounding box detection. Neural networks which could be used are SegNet[3] and YOLOv3[4]. These two neural networks are both well-known and yield great results within their respective field. The choice between these two types of object detection methods comes down to which of these types best suit the given application they will be used for. However, a significant distinction between them is the dataset which is being used for training. If pixel-wise detection is desired, pixel-wise annotation of the entire dataset much also be available. In the case for YOLOv3, only bounding boxes with the correct label is necessary. This means that a lot less work goes into the preparation of a YOLOv3 dataset. This is the most significant reason as to why YOLOv3 will be used for the sake of this project. Another bounding box neural network which could have been utilized is RetinaNet[5]. However, when comparing the training time of these two neural networks, YOLOv3 trains faster with only slightly worse accuracy[6]. This lack of accuracy can also be improved by using a large dataset[7].

The final problem formulation for this project is as follows:

Which approach to find driving lanes in fields is most desirable; classic computer vision or neural networks?

Chapter 3

Implementation

This chapter will focus on the development of the two approaches to finding driving lanes on agricultural fields. The first part of this chapter will cover the development of the classic computer vision approach, while the second part will cover the neural network approach.

3.1 Development of Classic Computer Vision Approach

This approach focuses on the use of more traditional computer vision techniques in order to find the driving lanes on fields. Many of the algorithms used throughout the development of this approach come from the library OpenCV[8]. On Figure 3.1, the process of finding driving lanes is split into individual steps to simplify the understanding. Subsequent to each of the elaborated sections, are two examples which demonstrate the result at the specific point in the flowchart.

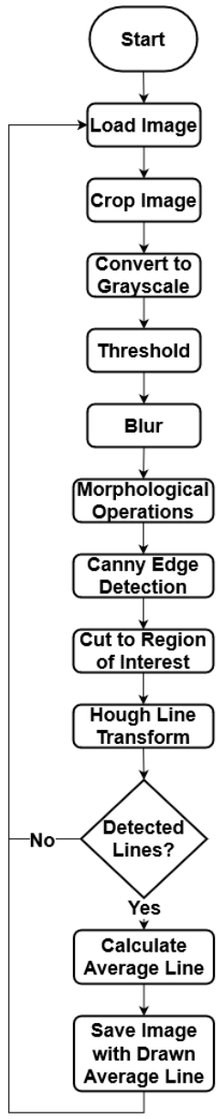


Figure 3.1: Flowchart depicting the flow of events throughout the classic computer vision approach.

Crop Image:
Each image is cropped in order to remove the lower portion of the image and some of the sky. This is done since some of the gathered images have been cropped previously for a different project. This previous crop removed the boom of the combine harvester. For the sake of consistency the input images are cropped to ensure that they all have the same size before going further through the script. Figure 3.2 shows two different images which were some of the images used during the development of this approach. Figure 3.2a is from the data which were previously cropped to remove the boom, and thus, only the top 150 pixels are cropped to remove the sky. Figure 3.2b is from the data which was not previously cropped. This image would then be cropped to remove 150 pixels from the top of the image and all pixels below 740. These two images were specifically chosen as examples because they come from different environments and they have very different average intensities, thus making generalization more difficult.

(a) Example of an image where the boom was previously cropped away.

(b) Example of an image where the image had not been cropped previously.

Figure 3.2: Examples of input images which were used for development of the classic approach.

Convert to Grayscale:
The image is transformed into grayscale using the function *'cv2.cvtColor'* with the color space conversion code *'cv2.COLOR_BGR2GRAY'*.

(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

Figure 3.3: Result of classic approach after grayscale conversion.

Threshold:
The image is turned into a black and white image using the function *'cv2.threshold'*. The threshold value was chosen to be 60 as a result of trial and error. The maxVal is set to 255 and the thresholding technique is *'cv2.THRESH_BINARY'*.

(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

Figure 3.4: Result of classic approach after black and white conversion.

Blur:
The image is blurred in order to remove most of the "salt and pepper"-noise. This is done using the function called *'cv2.medianBlur'* using a kernel size of 3.

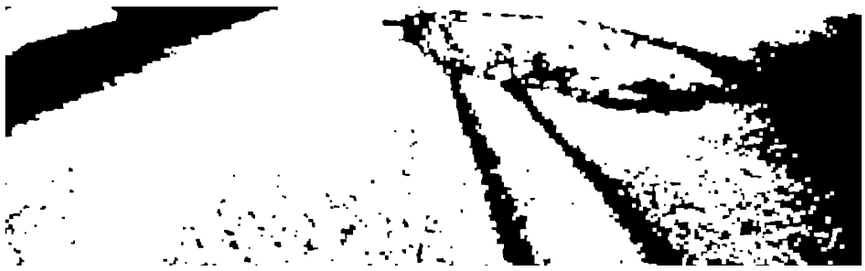
(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

Figure 3.5: Result of classic approach after blurring.

Morphological Operations:
In order to remove much more of the noise which is present in the image, a series of morphological operations are carried out. First off the images are dilated using *'cv2.dilate'* with an $8x8$ kernel and afterwards dilated again with $4x4$ kernel. The image is then eroded using *'cv2.erode'* with another $4x4$ kernel. This dilation and erosion with a $4x4$ kernel effectively the same as performing 'closing' on the image with a $4x4$ kernel.

(a) Result of Image 1 at the current point in the flowchart.



(b) Result of Image 2 at the current point in the flowchart.

Figure 3.6: Result of classic approach after the morphological operations.

Canny Edge Detection:
Canny edge detection is used to find edges in the image. This is done because these edges can be used to find straight lines later. The function which is used find edges is called '*cv2.canny*'. This function requires that you pass at least three arguments to it; image, threshold1 and threshold2. For the purposed of this script threshold1 and threshold2 can be any arbitrary integer values as a threshold has already been applied to the image.

(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

Figure 3.7: Result of classic approach after Canny Edge Detection.

Cut to Region of Interest:
As the driving lanes on fields typically move from the bottom of the image towards the horizon in the center of the image, a region of interest is applied to the image to remove unwanted areas. The mask which is applied to the image can be seen on Figure 3.8

Figure 3.8: Region of interest which is applied as a mask over the image.

After this region of interest is applied to the image using *'cv2.bitwise_and'*, the image looks as follows:

(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

Figure 3.9: Result of classic approach after the region of interest mask is applied.

Hough Line Transform:
To find straight lines in the image, a method called Hough Line Transform is used. In a polar coordinate system, lines can be represented as (ρ, θ); where ρ is the perpendicular distance from the line to the origin and θ is angle between this perpendicular line and the horizontal line measured counter-clockwise. Thus a line's equation can be written as:

$$y = \left(-\frac{cos\theta}{sin\theta} \right) x + \left(\frac{\rho}{sin\theta} \right) \tag{3.1}$$

Isolate ρ:

$$\rho = x\, cos\theta + y\, sin\theta \tag{3.2}$$

In the input image, for each white pixel the algorithm calculates and stores the result of Equation (3.2) where θ is incremented from 0 to 179 degrees. If all of the results are plotted in a (ρ, θ) coordinate system, they form different sinusoidal curves. An intersection between these sinusoids means that there is a straight line between the two pixels. In order to identify straight lines which contain more that just two pixels, the number of intersections are summarized and a threshold for a lowest number of intersections is applied. This way only lines with many white pixels on them will be accepted as actual straight lines in the image. The output of the function *'cv2.HoughLinesP'* is an array of the extremes of the accepted lines (x_0, y_0, x_1, y_1).[9] Due to the chaotic nature of agricultural fields, many lines which are faulty detections are still accepted by the algorithm; as can be seen on Figure 3.10

(a) Result of Image 1 at the current point in the flowchart.

(b) Result of Image 2 at the current point in the flowchart.

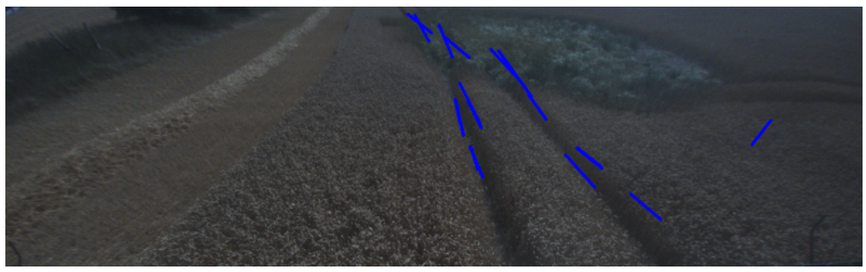
Figure 3.10: Result of classic approach after the Hough Line Transform.

The vast majority of faulty line detections are caused by close-to-horizontal lines as can be seen on Figure 3.10b. To remove these errors another threshold was applied to all of the lines based on their angle. The threshold is constructed as following:

$$|(x_0 - x_1)| \cdot 0.7 < |(y_0 - y_1)| \tag{3.3}$$

If Equation (3.3) is true, the line is kept. Otherwise, it is discarded. This removes all of the lines which have angles that are less than 35 degrees compared to a horizontal line. A result of this threshold is also that the script is no longer capable of finding driving lanes which are intentionally close to horizontal. For example, in Figure 3.10b a horizontal driving lane can be seen in the right side of the image, which might have gotten detection correctly as a driving lane later on in an image series. However, now those potential lines would be discarded. This side effect was accepted due to the overall improvement of the scripts ability to correctly detect close-to-vertical lines which are much more prominent in the dataset. As can be seen on Figure 3.11, most of the important lines are kept and a lot of faulty detections have been removed.

(a) Result of Image 1 at the current point in the flowchart.



(b) Result of Image 2 at the current point in the flowchart.

Figure 3.11: Result of classic approach after removing close to horizontal lines.

Calculate Average Line:
The slope of all of the individual lines are calculated and an average slope for all of the lines is found. Alongside this, the average x- and y-value of the lines is used as an intersection point that the average of all the lines will go though. The average line is then extended using the previously calculated slope by using the standard equation:

$$y = ax + b \tag{3.4}$$

Which is isolated for x:

$$x = \frac{(y - b)}{a} \tag{3.5}$$

Using this equation, x is calculated given predefined desired y-values; $highest_y$ and $lowest_y$. These values can be any desired values between 0 and 740, depending on how far into the horizon the line should be projected. In this implementation, $highest_y$ and $lowest_y$ were chosen to be 100 and 740. The final result of the script can be seen on Figure 4.1.

3.2 Development of Neural Network Approach

The developed neural network was based on the work of [10]. Many changes were made to this code as it was made to be run on a Linux operating system and altered to run on a Windows operating system. Most of the alterations were made to file handling. The model which is used for YOLOv3 is called *'Darknet-53'*, where *'53'* represents the number of convolutional layers. The structure of the model implemented for the YOLOv3 network can be seen on Figure 3.12.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.12: Structure of the model used for YOLOv3.[4]

Dataset:
The dataset used for training and inference consists of 568 images, which were hand-picked from a larger dataset of images from agricultural fields to ensure that all of the included images had driving lanes in them. The dataset was split so that 454 images were used for training and 114 were used for inference. The annotation used for training and inference were all done manually using a tool by [11]. This tool loads all images of a desired format and allows the user to draw on each image to indicate the position of certain objects within the image. In the case of this project, only a single class is used (*'lanes'*), however, if more classes are used, a specific class is chosen before drawing each object's bounding box. All of the classes which are present in the dataset are added to a file called *'classes.names'*, where each row represents the name of a new class. An example of an image which was annotated using this method can be seen on Figure 3.13.

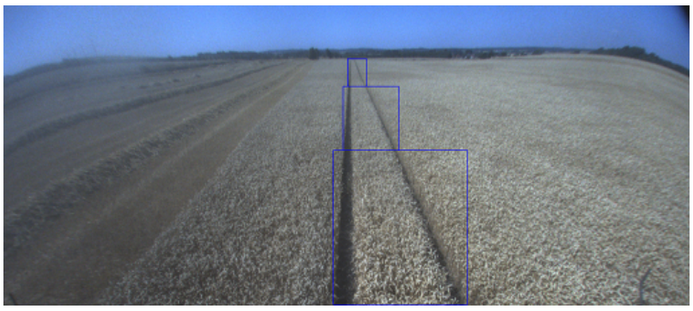


Figure 3.13: Example of annotation done using annotation tool by [11].

After choosing a class and drawing the bounding boxes for each of the objects, the tool reformats the bounding boxes into the following syntax:

$$class_{id} \quad x_{center} \quad y_{center} \quad width \quad height$$

Where $class_{id}$ corresponds to that class' row in the '*classes.names*' file (zero-indexed). x_{center} and y_{center} is the center of the bounding box. *width* and *height* is the width and height of the bounding box. x_{center}, y_{center}, *width* and *height* are all normalized from 0 to 1, such that 1 is the total width and height of the input image. An example of the annotation which is generated using the tool following the correct syntax, can be seen below. Here, each new row corresponds to a new object in the image.

0	0.579833984375	0.7416294642857142	0.19482421875	0.5100446428571428
0	0.536865234375	0.3805803571428571	0.08056640625	0.2120535714285714
0	0.516845703125	0.2293526785714285	0.02783203125	0.0904017857142857

Training:
The training data was loaded using the dataloader from the library '*torch*'. The data was also shuffled such that no particular order is kept while training. The network was trained using an Intel i9-10900K Central Processing Unit (CPU) @ 3.70GHZ, while running on 16 threads. The network was trained on a CPU, and not Graphics Processing Unit (GPU), because no CUDA compatible GPU was available, which increased training time a lot, as training on GPU is much faster. The learning rate of the network was 0.001 and the batch size was 8. The training process used no previously trained weights and was trained for 150 epochs. During the training process, binary cross-entropy loss is used to make class predictions. Training for this amount of epochs took approximately 25 hours, utilizing 80% of the CPU's total processing power, around 15.8 GB of Random Access Memory (RAM) and around 45-50 GB of Virtual RAM. After each increment of 5 epochs, the algorithm saves the best set of weights based on the Intersection over Union (IoU) value gathered from the evaluation.

Chapter 4

Results

4.1 Classic Approach

The end results of the classic computer vision approach can be seen on Figure 4.1. These two images are examples of good results. From these results, it is clear that this method can identify and correctly indicate the position and slope of driving lanes in different environments. It is able to find driving lanes even though these two environments are very distinctly different, in terms of planted crops and lighting conditions.

(a) Result of Image 1.

(b) Result of Image 2.

Figure 4.1: Result of classic approach after all steps have been completed.

However, the approach does not always yield good results. Examples of a few of the bad results which were also generated by this approach can be seen on Figure 4.2.

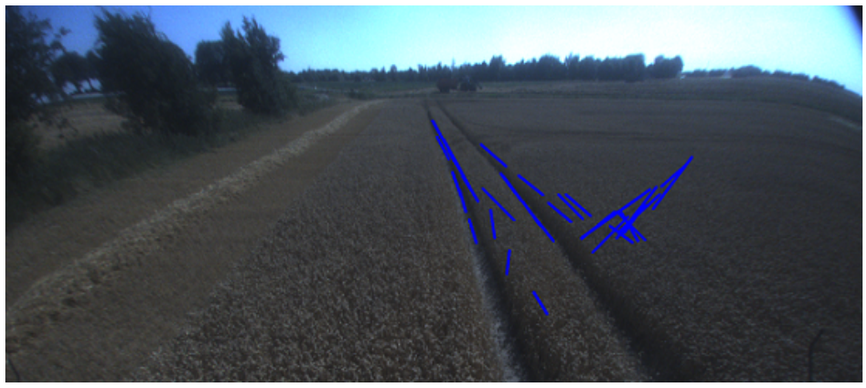
(a) Bad result of image from same environment as Image 1.

(b) Bad result of image from same environment as Image 2.

Figure 4.2: Bad result of classic approach after all steps have been completed.

These bad results come from the "chaotic" nature of the scene. As can be seen after the Canny Edge Detection step on Figure 3.7b, many noisy edges are still present. To clarify the issue which causes the bad results, Figure 4.3 depicts the Hough lines which were averaged to generate Figure 4.2.

(a) Hough lines which were average to generate averaged line.



(b) Hough lines which were average to generate averaged line.

Figure 4.3: Hough lines with a large amount faulty lines which results in a bad average.

On Figure 4.3a, the low amount of detected Hough lines means that the single faulty detection

has a much larger impact on the average, compared to Figure 4.3b where the larger number of detections means that the average more accurately follows the truth. From this , it can be concluded that the total number of detected Hough lines has a large impact on the resulting average line. In scenes where only a few Hough lines are detected, there is a high risk that the average line is inaccurate.

The following link will forward you to a video which demonstrates the result of the classic computer vision approach on a series of images:

`https://www.youtube.com/watch?v=o6r6qxy1Z7U`

4.2 Neural Network Approach

A few of the results which were generated using the YOLOv3 neural network can be seen on Figure 4.4.

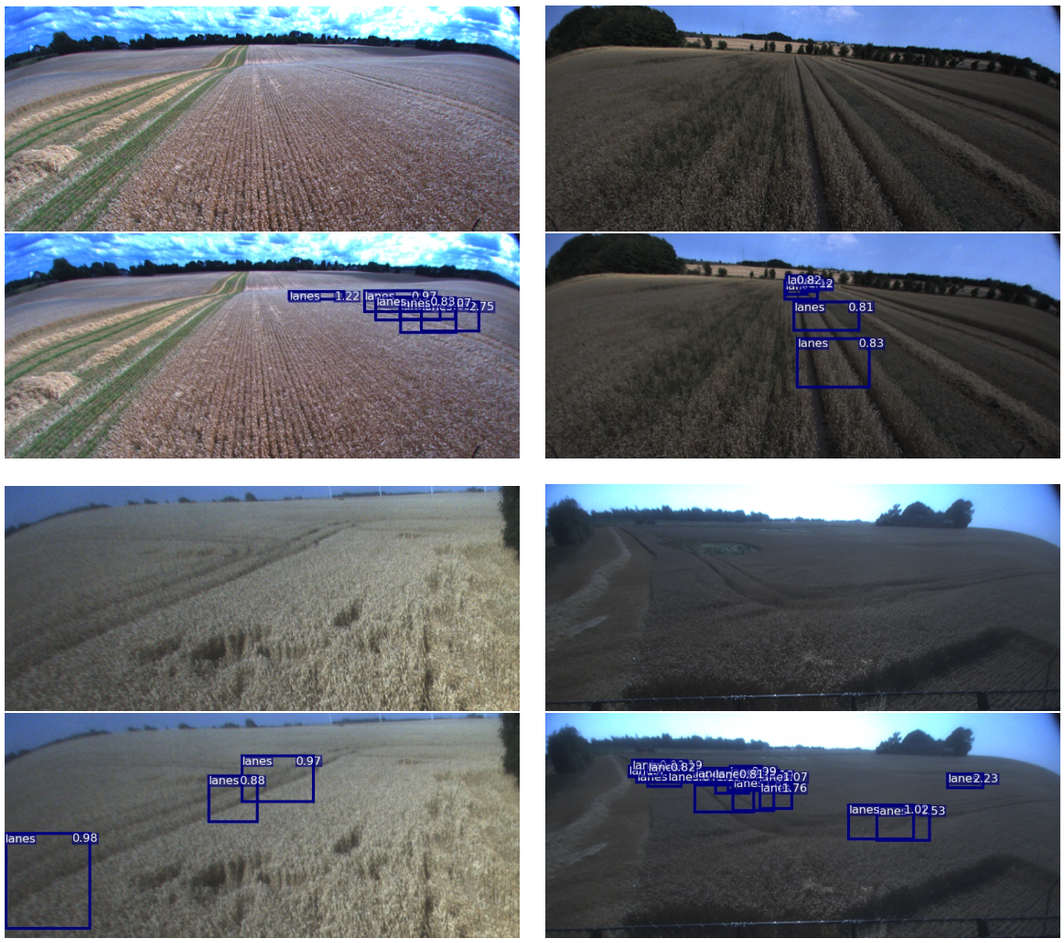


Figure 4.4: Result of neural network approach.

Here it can be seen that the neural network is able to successfully detect driving lanes in fields. It should be noted that these are some of the best results which were generated. Often the neural network failed to make any detections at all. However, the neural network very rarely makes false-positives; meaning it very rarely predicts something other than driving lanes as driving lanes.

4.3 Comparison

Both of these two approaches have their own positive and negative features. The classic approach excels at achieving an accurate approximation of the driving lane, if enough Hough lines are detected. Even if a low number of Hough lines are detected, as long as there is at least one Hough line, the script will give a result. On the contrary, the neural network approach will quite

often return no predictions at all. Overall the neural network's predictions are more accurate. Out of the 114 images which were used for inference, only a single image had a misclassification. In this case, the neural network detected a part of the harvested area in an image similar to the images in the top-left corner of Figure 4.4 as lanes. This is obviously an incorrect predictions, however to a certain degree, understandable. This area contains straight lines which closely resembles the driving lanes on the right side of the image. Although both approaches are able to correctly identify driving lanes to a decently accurate degree, an advantage of the neural network approach is its ability to detect driving lanes which are relatively horizontal. The neural network is also able to predict what would represent a turn in the driving lane. This behaviour can be seen on the bottom-right images on Figure 4.4. In the middle of these images, there is a turn. Here the classic approach would simply output a single line which approximates the two lanes on either side of the turn. The results of the neural network could be used to generate two individual lines, each representing their own respective side of the turn.

In order to summarize the positives and negatives of each system, Table 4.1 contains the key features which differentiate the two approaches.

Table 4.1: Positives and negative of the two approaches.

	Classic Approach	**Neural Network Approach**
Positives	Consistent approximations Can quickly be adapted to new environments	Very consistently good results Can detect horizontal driving lanes
Negatives	Faulty detections Can only detect driving lanes up to 35°from vertical	Does not always provide a prediction Requires new training if an environment is not recognizable

The classic approach's primary strength lies in its ability to almost always give a result which can at least help guide a vehicle towards the correct location. The primary positive feature of the neural network approach is its vastly superior accuracy.

Chapter 5

Conclusion

Two different approaches were designed in order to evaluate the best approach to take when attempting to find driving lanes in fields using a camera. The two approaches are able to find said driving lanes through the use of classic computer vision techniques and neural networks. Here the two approaches were both able to correctly find driving lanes left by tractors, although to different degrees of success. The classic computer vision approach consistently gives an output, which typically is a little worse than the ones produced by the neural network. The neural network produces good results which could easily be used as a reference to control a vehicle into the correct path. However, this approach does not yield consistent results. This could be a problem if consistent results are needed for the given application. Agricultural vehicles typically move quite slowly, which lessens the severity of this deficiency.

As mentioned in Section 4.3, both approaches have positives and negatives. There is no approach which is clearly best to take. Due to the classic computer vision approach's quick adaptability, it would serve as a good testbed from which a good standpoint could be achieved. This approach can relatively easily and quickly be altered to fit into a completely new environment, by simply changing the parameters of the used algorithms. As this approach requires much less work before progress can be begun, it would be smart to use this approach if a small time frame is precedent or it is to be used as a proof of concept.

If more time is available, the neural network has proven to yield better results and if a large enough training set is available, it could also improve the probability of predictions being made. The neural network approach would be better for full-scale systems due to its better performance, however, only in the cases where the number of predictions does not affect the usefulness of the system or can be increased; either through more training or a larger dataset.

Chapter 6

Discussion

Although decent results were acquired using the two approaches taken in this project, a few considerations have to be taken into account. The dataset which was used to train the neural network was fairly small. The dataset consisted of 568 images which is not very large. This likely caused the neural network to perform worse than what could have been achieved if a bigger dataset was available. The dataset was provided by AGCO and initially consisted of 1000 images. Sadly, it was not possible to gather more images as these are stored in a Robot Operating System (ROS) .bag format. ROS is intended to run on a Linux system, which was not available for this project. There is also no tool available which could open these bags and extract information from them. Thus, this tool would also have to be created, which would most likely have taken a lot of time away from the focus of the project. A way to adjust the neural network slightly to better accommodate a small dataset could be to change the loss function from being binary cross-entropy loss to something else. This loss function is used by the developers of YOLOv3, because the dataset which they trained on had many overlapping labels. An example of this could be '*Person*' and '*Woman*'. However, since the dataset which was used for this project only contains a single class, a softmax might have been better. The only limitation which is put on the dataset while using a softmax is that each bounding box must always only contain exactly one class.

As mentioned in Chapter 1, multiple other sensors could be used for detecting driving lanes, which could also yield interesting results if they were investigated. The use of an RGB-D sensor in particular could provide some good results. Important things to take into consideration if this sensor was utilized is potentials for crop lodging and areas with weed. These areas typically lower the overall height, compared to areas with crop. It would be essential to take such cases into consideration and ensure that they do not contaminate the results.

Bibliography

[1] Lars J. Munkholm, *Hvorfor faste kørespor – hvilke problemer løser de, og kan man tjene penge på dem?* [Online]. Available: https://sp.landbrugsinfo.dk/maskiner-markteknik/jordbearbejdning/jordpakning-og-loesning/sider/plk06_03_1_1_l_j_munkholm.pdf?download=true (visited on 31/05/2021).

[2] Moataz Elmasry, *Computer vision for lane finding*. [Online]. Available: https://towardsdatascience.com/computer-vision-for-lane-finding-24ea77f25209 (visited on 31/05/2021).

[3] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, *Segnet: A deep convolutional encoder-decoder architecture for image segmentation*. [Online]. Available: https://arxiv.org/abs/1511.00561v3 (visited on 31/05/2021).

[4] Joseph Redmon, Ali Farhadi, *Yolov3: An incremental improvement*. [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf (visited on 25/05/2021).

[5] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, *Focal loss for dense object detection*. [Online]. Available: https://arxiv.org/abs/1708.02002v2 (visited on 31/05/2021).

[6] Jonathan Hui, *Object detection: Speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3)*. [Online]. Available: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 (visited on 31/05/2021).

[7] Vidushi Meel, *Yolov3: Real-time object detection algorithm (what's new?)* [Online]. Available: https://viso.ai/deep-learning/yolov3-overview/ (visited on 31/05/2021).

[8] OpenCV, *Opencv*. [Online]. Available: https://opencv.org/ (visited on 20/05/2021).

[9] OpenCV (Bradski and Kaehler), *Hough line transform*. [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html (visited on 20/05/2021).

[10] Borda, *Pytorch-yolov3*. [Online]. Available: https://github.com/Borda/PyTorch-YOLOv3 (visited on 27/05/2021).

[11] miki998, *Yolov3-annotation-tool*. [Online]. Available: https://github.com/miki998/YoloV3_Annotation_Tool (visited on 27/05/2021).

Acronyms

IMU Inertial Measurement Unit 1
CPU Central Processing Unit 12
GPU Graphics Processing Unit 12
RAM Random Access Memory 12
IoU Intersection over Union 12
DGPS Differential Global Positioning System 1
ROS Robot Operating System 19

List of Figures