CHALMERS
UNIVERSITY OF TECHNOLOGY

Automated Height Adjustment for Truck-Trailer Coupling

Master's thesis in MPENM and MPCAS

Hongrui Chen
Joel Wall

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Automated Height Adjustment for Truck-Trailer Coupling

HONGRUI CHEN
JOEL WALL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Supervisor: Adam Khalif and Mohammad Otoofi, Volvo Trucks
Supervisor: Fredrik Kahl, Department of Electrical Engineering
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Automated Height Adjustment for Truck-Trailer Coupling
HONGRUI CHEN, JOEL WALL
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Coupling a truck to a trailer is a common but non-trivial task in logistics, typically requiring manual height adjustment to align the truck's fifth wheel with the trailer. This thesis explores an automated approach using computer vision to estimate the relative height between the truck's fifth wheel and the trailer, and automatically adjust the height accordingly. The inputs available for estimating this height is the camera feed from a singular backup camera mounted at the rear of the truck, and sensor data from the trucks air suspension.

The proposed system performs keypoint detection, locating corners, utilizing the YOLOv11 pose estimation model. We explore two different methods for calculating distance and then utilize this information to calculate the height difference between the trailer and the fifth wheel of the truck. The first method is based on assuming the width of the trailer and counts the number of pixels between the two bottom corners. The second method assumes both the width and the height of the trailer and solves the perspective-n-point problem to obtain the distance.

The model was tested in real-world scenarios and achieved high keypoint precision and recall. The distance calculation remained within 10% of the ground truth most of the time, which is adequate for this application. Furthermore, the error in height estimation was within 7-15 cm of margin of error which was sufficient to enable automatic height adjustment for the truck-trailer coupling.

The results indicate that the full system is successful on most of the trailers tested. This confirms the practical feasibility of using keypoint detection to understand the geometrical relationships between the camera and the trailer it is observing, which could also be useful for other assisted driving applications. Our results indicate that more training data are needed to obtain robust height and distance estimates when approaching a trailer from the side. The constraints our model must place on the shape of the trailer it is detecting also highlight the need for research into other methods with weaker assumptions about the shape of the trailer.

Keywords: assisted driving, computer vision, keypoint estimation, pose estimation, YOLO, YOLOv11

Acknowledgements

We would like to express our sincere gratitude to Fredrik Kahl for his help and continuous feedback throughout the project. A big thank you to Adam Khalif and Mohammad Otoofi for giving us the opportunity to work on such an interesting and engaging project. We are especially grateful to Mohammad for his great guidance on everything relating to machine learning and to Adam for all his support, practical assistance, and generous offering of his time whenever we needed it. Finally, we want to thank Hannes Winblad for his helpful input, inspiring ideas and putting us in contact with people who could help us with different tools, as well as Nrupathunga Ashok, Lars Johansson, and Mandus Jeber helping us to collect footage.

Hongrui Chen, Joel Wall, Gothenburg, June 2025

Contents

1

Introduction

Machine learning has become a valuable tool in various industries. In the transportation sector, particularly in the trucking industry, automation has played an increasingly critical role in improving efficiency and safety. The technological advancement in computer vision has enabled machines to interpret their environment with excellent accuracy. This paves a path enable automation of complex and repetitive tasks. One such task is the coupling process between trucks and trailers, which is a simple and repetitive task that is performed multiple times a day for a truck driver.

1.1 Background

The coupling mechanism ensures that the trailer is securely attached to the truck, allowing the truck to control the trailer. When coupling a trailer to a truck, the fifth wheel of the truck must match the height of the trailer's kingpin, in order for the kingpin to attach to the fifth wheel, see Figure 1.1. Normally, the truck driver performs the coupling process by exiting the cabin and, with a remote control, adjust the suspension such that the height of the fifth wheel matches the height of the kingpin. This task can be time-consuming and at times, dangerous, since climbing up and down from the cabin takes a toll on the body and carries the risk of falling. To improve efficiency and safety, it is necessary to automate this process, both to assist the driver and to pave the way for fully autonomous driving in the future.

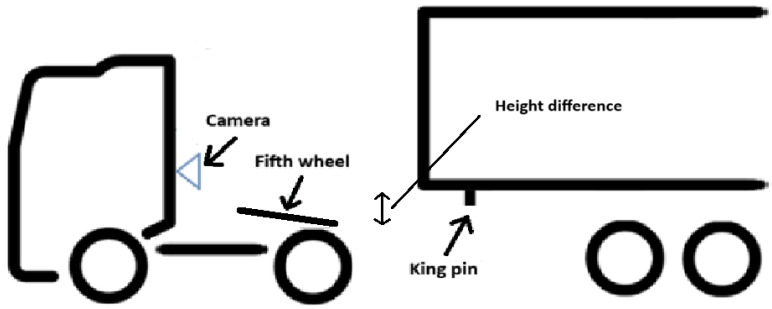

Figure 1.1: Different components of the truck and the trailer. The fifth wheel must be the same height as the kingpin for them to attach to each other.

1.1.1 Truck and Trailer Coupling Process

The coupling process begins with positioning the truck directly in front of the trailer. Then, the driver reverses the truck slowly towards the trailer while maintaining a straight alignment between the truck and the trailer. When the truck is close to the trailer, the driver leaves the cabin to verify that the height of the fifth wheel is the same as the bottom of the trailer. If necessary, the driver will use a remote controller to manually adjust the height of the fifth wheel until the desired height is reached. The driver then continues to reverse the truck until the fifth wheel is securely attached to the kingpin, see Figure 1.2. Finally, to ensure that the trailer can be safely towed on the road, the necessary cables for the braking system and trailer lights must be connected. Additionally, the trailer's landing gear should be fully retracted.

Figure 1.2: The reversing sequence for the truck and trailer coupling.

1.2 Objective

The objective of this project is to automate the height adjustment process during the coupling of a truck and a trailer with the help of the camera mounted on the rear of the truck. The focus is to develop a prototype capable of operating in real-time on a standard laptop with actual trucks and trailers using computer vision.

During this thesis project, these following questions will be answered:

1. How can deep learning be used to estimate the height and distance between the truck and trailer without the need for physical markers?
2. What could an implementation of automated height adjustment during coupling look like?

1.3 Limitations

The model of this project will be a proof of concept prototype that will lay the groundwork for finalization of a product.

- The model will not be deployed on any of the truck's ECUs. Instead, to demonstrate the potential of this project, it is essential for the machine learning

model to function in real-time on a laptop within a controlled environment on a actual truck and trailer.

- Due to the time constraint, the data collected for this project will be limited to the period between February and May in the daytime. As a result, the dataset will not capture the full range of background conditions, lighting variations, or seasonal differences.

- The model will be limited to detecting only rectangular trailers with an approximate dimensions of 244×259 cm. This is because most trailers have standardized dimensions, closely aligning with international standards for shipping containers ISO 668 [16]. However, trailers exist in a wide range of shapes and dimensions, which will unfortunately not be captured in the training data due to the time constraint, as well as the higher difficulty in creating a generalized model.

- Data collection will be performed using only a single truck equipped with a rear mounted camera. This means that the dataset will consist of a single viewing angle.

1.4 Related Works

Volvo Trucks has previously developed an autonomous height adjustment system that utilizes ArUco markers (similar to QR codes) on a fixed position of the trailer and the fifth wheel, see Figure 1.3. In this approach, the size of the ArUco markers are known and the model has a predetermined intrinsic matrix and distortion coefficients, which are parameters obtained from camera calibration. The model detects the keypoints on the ArUco markers by using pose estimation and gets translation and rotation vectors. These vectors describe the position and orientation of the markers relative to the camera. Since both makers are of known size and are placed on a fixed position, it enables the system to calculate the distance and height difference between the truck and the trailer and performing autonomous autonomous height adjustment.

Another similar project produced as a collaboration between General Motors and Michigan State University concerns finding the location of trailer hitches within an image and mapping to a location in 3D space using neural networks [1]. This project did not use truck trailers, but types of trailers usually towed by normal cars, such as caravans. We do not think their approach would be suitable for our application for two reasons.

1. Their geometric model for finding the distance is very sensitive to disturbances when the distance to the trailer is much greater than the height difference between camera and trailer hitch, and does not work at all when the trailer is higher. These scenarios are the norm for the cases our model is built to handle.

Figure 1.3: ArUco markers placed at fixed positions on the trailer and the fifth wheel.

2. Their model contains five different specialized neural networks for different tasks and scenarios. Since our model is developed to eventually run on embedded hardware with limited resources, we deem this approach of many separate networks to be unsuitable.

1.5 Ethical Aspects

The current coupling procedure requires the truck driver to exit the cabin, climb down from the truck, and manually adjust the truck's height to match the trailer using a remote controller. Once the correct height is achieved, the driver climbs back into the cabin and reverses the truck into the trailer. Depending on the driver's skill, this process may need to be repeated several times. This task can be physically demanding and potentially dangerous, especially when there are other trucks nearby or the ground conditions are slippery. By automating this task, the physical strain on drivers will decrease and the risk of accidents will be minimized.

Although the automated system may eventually perform the coupling process with flawless precision, there will likely be cases where the automated system fails. When such failures occur, the determination of responsibility becomes more complex. It is important to establish clear guidelines for accountability to address the ethical and legal concerns. One may suspect that the driver will be considered responsible with trucking companies or insurance paying for damage to property or compensation for resulting injuries, should they occur. We suspect that this approach would be taken as it would follow the pattern of how other assisted driving tools such as adaptive cruise control are currently treated.

When further developing this product beyond a proof of concept, a feature that alerts the driver when the model has low confidence shall be implemented. Such a feature would decrease the rate of incidents until the time when the system reliably outperforms humans under all conditions.

Although a full automation of truck driving and the loading process is still a distant goal, this project still contributes to the development of fully autonomous, driver-

less trucks. Therefore, as automation progresses, it may replace tasks currently performed by humans, leading to potential job replacements. While the system is designed to assist drivers rather than replace them, it is important to consider the broader implications of automation on employment within the industry.

2

Theory

In this chapter, we present the theoretical background for the methods used in this thesis. We introduce the truck setup, camera system, and explain the use of keypoinht estimation for detecting trailer corners, together with two methods for estimating the geometric relationship between camera, fifth wheel and trailer. One method is a geometric model of our own, and the other one is the Perspective-n-Point method for estimating camera pose. Finally, the main model used for this project, YOLOv11 model is introduced, with a detailed overview of its architecture, loss functions, and use of pre-trained weights.

2.1 Truck Specifications

The truck used in this project is provided by Volvo Trucks and the model name is Volvo FH. The truck is equipped with a fisheye camera with a 175-degree diagonal field of view. The perspective from the backup camera can be seen in Figure 2.1. This camera is mounted on the lower left side of the rear of the truck, positioned approximately 8 cm higher than the fifth wheel.

Figure 2.1: Trailer and fifth wheel viewed from the camera's perspective.

The Volvo FH is equipped with air suspension, which allows it to raise and lower both the front and rear axles. As a result, the camera's angle may vary depending on the suspension configuration. Furthermore, it is possible to access various measurements from the truck via the CAN bus, such as the suspension height at the front and rear axles, or the pressure on the fifth wheel. These values will be useful when calculating

the height difference later on. Moreover, it is also possible to send signals to the truck; this will be used for the main goal of this project by setting a reference height which the trucks built in control system will try to achieve through control of the air suspension.

2.2 Pose Estimation

Pose estimation is a task that involves identifying the position and orientation of a object in an image or video, typically in terms of keypoints. The keypoints can represent various parts of the object such joints, landmarks, or in our case, corners of a trailer. There are 2D pose estimation, which this project mainly focuses on, that estimates the x and y coordinates of keypoints in the image plane. Then, there are also 3D pose estimation which estimates x, y and z coordinates, giving a full 3D representation of the body in space.

2D pose estimation models can localize keypoints. Over time, the model will learn the geometric relationships between keypoints, essentially learning the shape of the object [15]. A good pose estimation model is also capable of inferring the positions of missing or occluded keypoints, which is useful when trailer corners are partially obscured.

Additionally, pose estimation is suitable for this project because our task only requires localizing the bottom two corners of the front face of the trailer to compute the height difference between the trailer and the fifth wheel. Pose estimation is able to localizes all four corners of a trailer, which is more than sufficient.

Alternative methods like semantic segmentation would provide a detailed outline of the entire trailer, and so would not distinguish important components of a trailer such as corners or edges from the rest of the trailer. Extracting important features from a segmentation mask would then require additional post-processing and introduce new sources of potential errors.

2.3 Fisheye Camera

Our camera uses a fisheye lens with an ultra wide-angle field of view, which captures up to a 175 degree diagonal view. For this project, we will assume an equidistant fisheye camera model. In a equidistant fisheye camera model, the further the pixel is from the center (in pixels), the larger the angle θ is, and the relationship is linear [11]:

$$r = f \cdot \theta.$$

Here, r is the pixel distance from the center, θ is the angle between the camera's forward direction and the direction of an incoming light ray, and f is a scaling factor like focal length. This fisheye model assumption will be used in used in Section 3.4.1.

2.3.1 Camera Calibration

To convert between pixel coordinates, which represent how the camera perceives the world, and real-world coordinates, camera calibration is required for high accuracy [8]. Most lenses especially fisheye lenses, introduce distortion that bends straight lines in the image. Calibration corrects for this, producing undistorted views. To model how a camera maps 3D points to 2D images, we use the intrinsic camera matrix:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

where f_x and f_y are the focal lengths in pixels, (c_x, c_y) is the optical center, and s is the skew coefficient. In addition to the intrinsic matrix, distortion coefficients are needed:

$$\text{distortion} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix}, \tag{2.2}$$

where k_1, k_2, k_3 model radial distortion and p_1, p_2 model tangential distortion. With these parameters, it is possible to undistorted the distorted images. The details of how to obtain the intrinsic matrix and distortion coefficients will be further discussed in Section 3.4.2.

2.4 Perspective-n-Point

The Perspective-n-Point (PnP) problem is a task in computer vision that estimates the 3D pose of the camera relative to a object, the object in our case is a trailer [10]. The goal is to find the camera's rotation matrix R and translation matrix t in the 3D space, relative to the object.

Mathematically speaking, given the 3D point $\mathbf{X}_w = [X_w, Y_w, Z_w]^T$ of the object in the real world, and its image projection $\mathbf{x} = [u, v]^T$, the PnP projection equation is

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}.$$

Where s is a scale factor, $\mathbf{K}$ is the intrinsic matrix which is possible to obtain from the camera calibration. The rotation matrix R and translation vector t are the parameters we will optimize.

Since every variable except R and t are known, it is possible to use the function `solvePnP` in the computer vision library, OpenCV, to solve for R and t [9]. `solvePnP` will by default use an iterative method based on Levenberg-Marquardt optimization. This essentially finds the 3D location and rotation of the camera that minimizes the projection error:

$$\text{Total Error} = \sum_{i=1}^{N} \left\| \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)} \right\|^2.$$

The equation is the sum of squared distances between the predicted 2D projection of the 3D point $\hat{\mathbf{x}}^{(i)}$, and the actual 2D image points $\mathbf{x}^{(i)}$. With the obtained translation vector, we can easily obtain the distance from camera to trailer. The practical implementation of this process will be described in Section 3.4.2.

2.5 YOLOv11

The keypoint detection model that this project will use is YOLOv11 (You Only Look Once) developed by Ultralytics [5]. It is a deep learning model that is designed for various computer vision tasks such as object detection and 2D pose estimation. YOLO models are designed to be lightweight and efficient, making them suitable for real-time applications and deployment on embedded systems. Another advantage of using YOLO is its flexibility. It supports training on custom datasets and a streamlined pipeline for training, inference and deployment is available from the developers.

Understanding the loss functions is both essential in understanding the evaluation of the model performance, and key to understanding how YOLO differentiated itself from earlier computer vision models. The internal architecture, while not essential to understand this project, exemplifies many patterns in the architectural designs seen in the computer vision field. Grasping the purpose and mechanism of these components is crucial to understanding the field of computer vision in the age of neural networks.

2.5.1 YOLO Architecture Overview

The YOLO network architecture consists of three parts. The first part is the backbone of the network which extracts features from the image. The second part, which is referred to as the neck, integrates high-level features with more semantic meaning, with less processed high-resolution features which are more basic/primitive. Then, there is the head that maps these learned features to the desired output.

There are many different YOLO variants with different heads for different tasks, including image classification, semantic segmentation, and more. The type of head used for this project will perform keypoint detection.

2.5.2 Output Structure

YOLOs output structure and basic setup of the loss function is inherited from the original YOLO paper [13]. The output is divided into parts, each based on squares in the original image. For each rectangle, the network outputs a probability distribution of the type of object it thinks is most likely to be centered in the square. The network also generates a certain number of bounding boxes centered in each square. Each detection consists of the coordinates of the center of the detection

(which is in the associated rectangle), the width and height of the detected object, and a predicted probability that the detection is real.

For keypoint detection, there are additional outputs for each detection. These are predictions of the location of each keypoint, and the predicted probability that it is visible/in frame.

2.5.3 Loss Functions

The loss function is divided into several parts reflecting the meanings of the different parts of the network output. The **confidence loss** represents the deviation between the predicted probability that the detection is real, and the intersection over union (IOU) between objects in the ground truth and the predicted bounding box. Intersection over union between sets A and B is defined as

$$IOU_B^A = \frac{|A \cap B|}{|A \cup B|}.$$

The loss function we use on the predicted probability and the IOU is the binary cross entropy defined as

$$IOU \cdot \log(p) + (1 - IOU) \cdot \log(1 - p).$$

In cases where there is no labeled object centered in the square, the loss is $(p - 0)^2$.

Bounding box loss takes for each labeled object the most overlapping prediction (highest IOU) centered in the same square, and computes the difference in coordinates between the label and prediction center and difference in width & height between prediction and labeled object. The loss is the sum of the squares of these differences.

Class probability loss is computed on a per square basis, rather than a per prediction basis. The model tries to predict what class of object could be centered in that square. If the labeled image has any object centered in that square, the cross-entropy loss is used. Otherwise, the class probability loss for that square is zero.

The **object keypoint similarity** is the measure of performance which is optimized to learn the keypoints. It is calculated using the *keypoint similarity*, KS_i (for keypoint i), and the visibility flag v_i. KS_i is computed as

$$KS_i = \exp\left(\frac{-d_i^2}{2s^2k_i^2}\right),$$

where d_i is the difference between prediction and label for keypoint i, s^2 is the area of the segmented object, and k_i^2: a scaling factor for the importance of respective keypoints. k_i^2 is by default chosen s.t. $k_i = 2\sigma_i$ where σ_i is the standard deviation of the location of keypoint i within the box; this is so that a keypoint that varies greatly in location is not neglected due to it having a small impact on the derivative of the loss function, even when it is inaccurate.

Then the object keypoint similarity is computed as

$$OKS = \frac{\sum_i KS_i \cdot \delta(v_i > 0)}{\sum_i \delta(v_i > 0)}.$$

This means that OKS is the sum of the keypoint similarity for all visible points, divided by the number of visible points.

Once the individual parts of the loss have been computed, the loss for the whole network is computed as a weighted sum of the losses for the individual predictors. These can be chosen to compensate for different loss functions having different sensitivity, or to modify the relative importance of the different parts of the output.

2.5.4 The key Contribution of YOLO

The way which confidence loss is used is key to understanding the uniqueness of the original YOLO model. When running the model, rather than just training it, bounding boxes are selected by the following method. First a filter is applied removing bounding boxes where the predicted IOU with the label is below a certain threshold. Then, for the remaining bounding boxes, if two bounding boxes overlap by more than a certain amount (IOU > some threshold), the bounding box with smaller *predicted* IOU is discarded.

In this way, YOLO has a network with a constant number of predicted objects in the output, but still can handle the case that the number of objects in this patch of the image is smaller than this number of outputs. This was the key innovation of the original YOLO model, which simplified both training and inference compared to earlier models.

2.5.5 Division of Network Parts & the Role of Skip Connections

The architectural layout of YOLOv11 can be seen in figure 2.2. In the **backbone**, features are extracted from the image. The deeper you get in the backbone, the more advanced features you get at the expense of a much lower resolution. As the figure indicates, the features at three different levels are passed forward to the neck.

In the **neck**, advanced features are combined with lower-level, high-resolution features, to obtain a feature map of both high resolution and high information. The reason why the advanced features have lower resolution for a while before being upscaled is to enable a higher number of layers without a huge performance penalty. It also allows information to be propagated over a larger area of the image for a given number of layers.

The various shortcuts you can observe in the diagram also have the benefit of reducing the vanishing gradient problem present with deep neural networks.

Lastly, the **head** uses all the general image features extracted and processed by the earlier layers and is especially adapted to a particular task. The backbone and

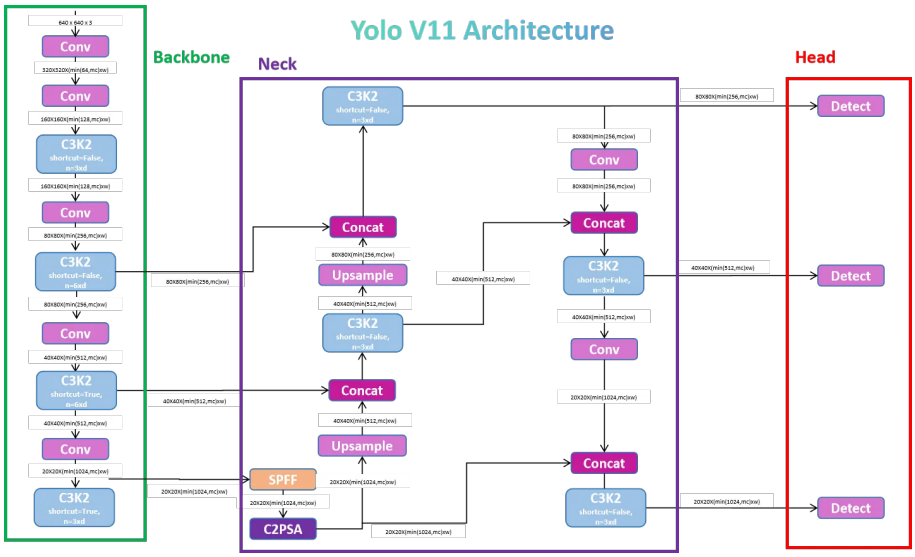

Figure 2.2: High Level Layout of YOLOv11 network [12].

neck are built to perform tasks that are common to all computer vision tasks, while the head will be architecturally adapted to some particular task, such as object detection, image segmentation, classification, or in our case, pose estimation.

2.5.6 Explanation of Common Modules

YOLO11 is quite architecturally different from previous version of YOLO. Below is an explanation of the key core modules for the YOLO11 network, which were absent in early versions [5].

C3K2 Block YOLOv11 uses C3K2 blocks to perform feature extraction. The C3K2 blocks divide the image features into two parts. One part goes through one or multiple bottleneck layers and the other part is skipped. This method results in faster computational time, since half of the features are skipped. Moreover, the skipped features help to keep original details while the processed ones add new information. This skipping of the bottleneck for half the channels also contributes to addressing the problem of vanishing gradients.

A small kernel size is used, which results in a faster and computationally cheaper layer. In the end, the processed and unprocessed features are merged by concatenation and a convolutional layer. This block is illustrated in figure 2.3.

The **Bottleneck** module consists of two convolutional layers where the number of dimensions shrink by (normally) 50% from the first convolution, and the second convolution then expands the number of channels to match the input dimension.

SPPF Layer Spatial Pyramid Pooling - Fast is designed such that strong information about features propagates to surrounding pixels. This module consists in

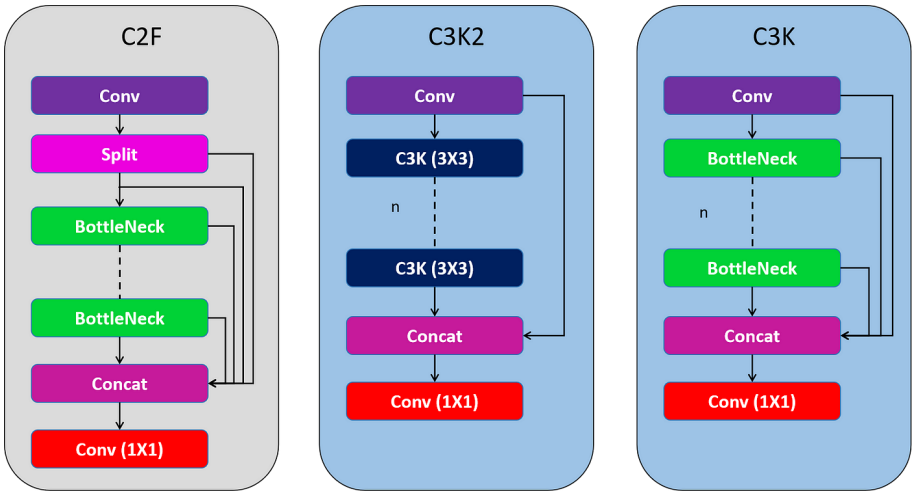

Figure 2.3: Schematic representation of C3K2 block, along with it's constituents (right) and predecessor (left) [12]. For Yolo, typically $n = 1$ for both C3K2 and C3K blocks.

first, a convolutional layer, with activation function (ReLU) and batchnorm. This convolution cuts the number of feature channels in half.

Then there is a MaxPool2d module which has stride, kernel size and padding such that the resolution is unchanged. This module is applied zero, one, two and then three times to the data, and the data at these intermediate stages is appended along the channel dimension.

Then another convolutional layer is applied, changing the number of channels back.

For illustration purposes, the code for this has been specified below where `cv1` and `cv2` are convolutional layers with ReLU activation and batchnorm, and `m` is MaxPool2d with kernel size 5, stride 1, and padding 2.

```
def forward(self, x):
    """Apply sequential pooling operations to input and return
    concatenated feature maps."""
    y = [self.cv1(x)]
    y.extend(self.m(y[-1]) for _ in range(3))
    return self.cv2(torch.cat(y, 1))
```

The code is taken from the Ultralytics repository with code for YOLOv11, but this feature was already introduced in YOLOv5.

C2PSA The C2PSA (Cross Stage Spatial Partial with Spatial Attention) block uses a 2-dimensional transformer block[1] to improve YOLOv11's understanding of the relationship between different regions within the image. Like many of the other

[1]The Ultralytics repository [4] has its own implementation of the attention mechanism for 2d images. To better understand it, please refer to `https://github.com/ultralytics/ultralytics/blob/main/ultralytics/nn/modules/block.py`.

features, it first applies a convolutional layer and picks out half of the features to use for the transformer block, and half that are passed on unchanged. The result of the transformer is then combined with the un-processed input by a convolutional layer.

2.6 Pre-Trained Weights

Because we have limited access to data, we use pre-trained weights from Ultralytics [4]. They also come with utilities for transfer learning to adapt the network to custom datasets with new categories, which greatly simplified the training process for us. Using transfer learning allows us to have an effective network with much fewer samples in the training set. The small size of our training set would mean that training from scratch would be totally infeasible for our problem without a massively expanded data collection.

Since the pre-trained model was used for something different, the head of the network had to be trained from scratch, even if the backbone and neck could be reused from the earlier task on which the pretrained weights were trained.

3
Method

In this chapter, we present the methods used to develop our height estimation model. We begin by describing the hardware setup used for video stream collection, and access to CAN signals. Next, we explain the data collection process, how the images were labeled, and how the collected data was used to train the computer vision model. Finally, we explain how we obtained the distance from the camera to the trailer, from the keypoint detection, and from that, how the height estimation is calculated.

3.1 Setup for Data Collection

The truck used for this project did not have any built-in feature to stream the camera feed to an external device. This meant recording or streaming the backup sequence on a computer was an engineering task in itself.

To solve this, we intercepted the video stream sent from the camera to the truck's internal display. We did this by disconnecting the cable between the camera and the display, and placing a network switch, a device that forwards data between devices, in between. The switch route was configured to mirror all the network traffic to an additional port, which we connected to our computer. Using *Wireshark* [17], we captured all the data packets being sent from the camera.

To extract the video from this data, we filtered the network traffic by destination IP and port to isolate the video stream. The video was encoded in H.264, a video compression standard, but since the video frames were split across multiple packets, we had to reassemble them manually. This required understanding of RTP (Real-Time Protocol), a network protocol for streaming data live. Specifically, identifying which packets contain full frames, and which packets mark the start of a fragmented frame. Then a start sequence, `0x00 0x00 0x00 0x01`, had to be inserted in the stream to mark the start of each frame.

To handle this, we wrote a custom program that reconstructed the video stream and passed it to *FFmpeg* [3], a software for recording, converting and streaming audio and video, to generate a proper .mp4 file. After implementing this program in Python and noting the poor performance, we re-implemented the program in C. This forced us to work directly with the raw memory layout of the network packets instead of relying on higher-level abstractions like Python dictionaries which, while

tedious, did give a great increase in speed. This was essential for later being able to process a video feed in real time.

We use the CAN interface to read sensor data such as suspension height and weight on rear axles, and to set a target height which the suspension control system tries to achieve. The height data in the front and back is useful to estimate the way the truck may tilt as the front and rear suspension is adjusted at different speeds. The suspension height in the back is also later used to provide a true label for the correct height of the trailer. The rear axle load is used as an indicator to detect when the truck is raised enough to make contact with the trailer.

3.1.1 System Setup for Real-Time Video Processing

Collecting and decoding the video stream, processing it, and sending height adjustment commands to the truck requires passing data between multiple programs in real time.

To do this, we replaced Wireshark with *tshark*, its command line counterpart, as Wireshark requires saving network traffic to a file before it can be processed. In contrast, tshark allows streaming the captured data directly to standard output. With the use of unnamed pipes, we were able to pass this data directly to our extraction program. The output of the video extraction program was then passed to a named pipe, allowing the machine learning model to read the video as if it were a regular MP4 file using *OpenCV*[1].

3.1.2 Software Environment

Setting up our software environment was challenging due to the need to meet conflicting requirements.

- Since model training was to be performed on a server running a Linux environment, we wanted use a similar setup on our Volvo-supplied laptops. This meant using WSL2, a virtual machine with some tweaks to be light-weight, and somewhat simplify interaction with the host system[2].
- In order to read network packages to extract the video feed in the truck and access CAN-interfaces to read current suspension height and send commands, access to the hosts network interface and USB devices was needed, neither of which are available by default in a virtual machine.

Since we did not fully anticipate the difficulty of the second problem, and regardless, only had previous programming experience with interprocess communication

[1]This necessitated using FFmpeg options for excluding metadata that usually appears last in mp4 files. This was achieved using the flag `empty_moov` [3].

[2]This includes having the hosts file-system mounted in the file system of the virtual machine, and the ability to launch Windows programs from within the virtual machine (Ran on the host, but with output passed to the virtual machine.)

on Unix systems, we designed our video extraction problem to be run in WSL. Our solution for the network interfaces was to launch tshark on the host system from within WSL, as this enables both access to the network interfaces, but output through `stdout` is still redirected into WSL. To enable sending and receiving CAN-messages, we needed to set up USB pass through from windows to the virtual machine. This could be done with the *usbipd* program[3].

We were lucky that all traffic could be directed to and from WSL as the details of how we would communicate with external devices were an afterthought and our program for decoding the video feed was already written before we realized the complexities of communicating with the external devices.

3.1.3 Hardware Setup

A photo of the actual hardware setup used for video extraction is shown in Figure 3.1. The camera traffic uses the IPv4 protocol, but the FAKRA HSD connectors (2) used between the camera and display are not compatible with the Ethernet switch (3). To bridge this, BroadR-Reach media converters (1) were used to translate the signal for Ethernet cables. The Ethernet switch placed in between was configured to mirror all traffic to the port connected to our computer, allowing us to capture the video stream. At last, an Ethernet to USB-C adapter (4) was used for our laptop.

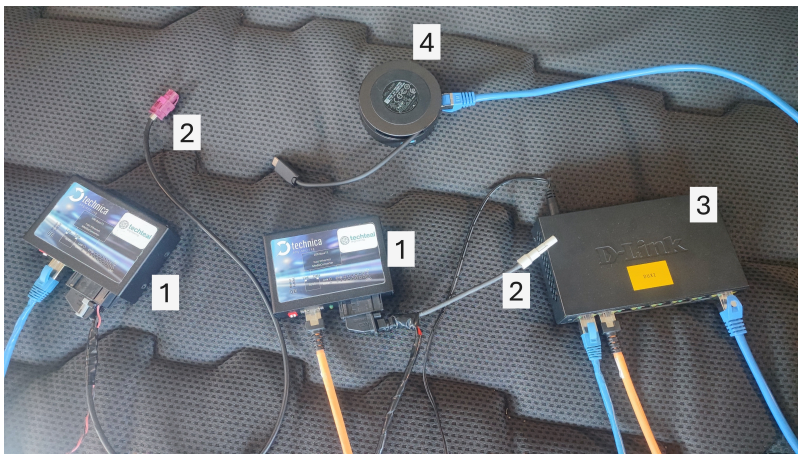

Figure 3.1: A picture of the hardware used to extract the camera stream.

Figure 3.2 shows a schematic of how the hardware was connected to intercept data and send/receive messages to and from the truck.

CAN is an interface for internal communication within vehicles and other devices where reliability is essential. Unlike IP, messages are not sent to a certain destination, but broadcast to all devices on the network. To be able to read this traffic from our computers, we use the Kvaser Memorator Pro 2xHS v2, which enables us to read and write messages over USB. A detail of how we use this device is that

[3]This program forwards USB traffic to WSL over a network interface, and the firewall rules are setup to not allow for this on public networks. To make this work within the truck, we had to manually configure windows to view the truck's internal network as a private network.

the drivers should be installed within WSL for our software setup, as this is where we communicate with the device. Windows is just set up to pass through traffic without the need or any use for interpreting the traffic.

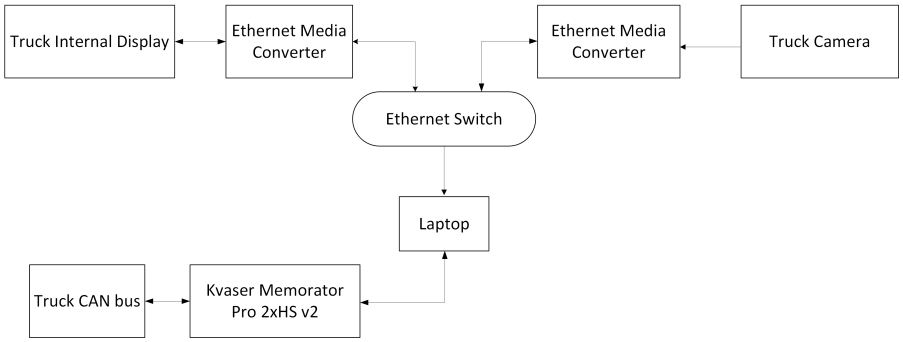

Figure 3.2: Schematic representation of the hardware used to communicate with the truck.

3.2 Collecting Reversing Sequence

The recorded videos of reversing sequences mainly involved trucks backing up straight and slowly into trailers, with adjustments in height during the sequence to align the fifth wheel with the bottom part of the trailer. The footage were gathered from various trailer parking lots around Gothenburg. We usually asked companies for permission to use their trailer parking for recording.

Since many companies used similar containers for their trailers, we only managed to collect data from approximately 30 to 40 different trailers. In the end, a total of 100 reversing sequence videos were obtained from the rear cameras perspective. Each reversing video was around one minute long. Given that the truck was reversing slowly straight into the trailers, many of the frames were similar. We extracted roughly ten representative but varied images from each video, resulting in a final dataset of 1000 images.

In addition to recording video of the reversing sequences, we also collected data for validating the model by logging the suspension height at both the front and the rear. Consequently, for reversing sequences where the fifth wheel aligns with the height of the trailer in the end, we can calculate the actual height difference between the fifth wheel and the trailer by comparing the recorded suspension height measurements. These will be used as ground truth for evaluating the model's height estimation.

3.2.1 Labeling for Pose Estimation

Labeling for pose estimation involves creating bounding boxes for the objects in question, in our case the front end of the trailer, and marking the visible keypoints which are the corners of the trailer. The location of the keypoints can be represented

as a set of 3D `[x, y, visible]` coordinates. Here, the visibility flag is a value between 0 and 2 defined as:

- **0**: Not labeled, meaning the keypoint is outside of the image or was not annotated by the annotator.
- **1**: Not visible, which means that the keypoint exists in the image, but is obstructed.
- **2**: Labeled and visible, which means that the keypoint clearly exists in the image and is annotated.

Furthermore, each corner of the front end of a trailer is uniquely labeled `[bottomleft, bottomright, topleft, topright]`. The annotations were performed with a free tool called *Roboflow* [2]. An example of this annotation is displayed in figure 3.3.

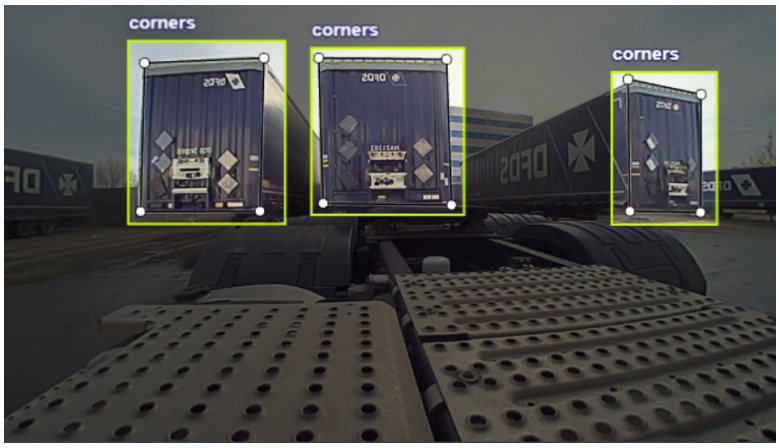

Figure 3.3: Corners as keypoints for the trailers.

3.3 Training the Model

The YOLO model uses its own `.txt` file annotation format. Roboflow is able to convert various annotation format into YOLOs format. In addition to the annotation format, YOLO demands the folder structure illustrated in figure 3.4:

Here, `data.yaml` is a configuration file used to define the dataset structure for training the YOLO model. It provides metadata such as the paths to the images and labels, and the object classes. After this, we can apply transfer learning with the YOLO model by training it with our labeled custom dataset and develop a trailer corner detection system. However, there are still a few problems that need to be considered.

3.3.1 Preventing Data Leakage

Data leakage in the context of machine learning means that verification is done with data that are also present in the training or test set. If it happens, then the model

Figure 3.4: Folder structure which YOLO expects for the dataset.

performance can increase due to overfitting not only on the training data, but also on the verification data, thus making the reported results unrealistically optimistic. Thus, avoiding data leakage is an essential consideration in any machine learning project.

When we record a backup sequence, we convert the MP4 file containing our target trailer into multiple images. Since there can be multiple images of a single trailer, we split images in such a way as to minimize contamination between the different dataset splits. When recording video footage of one trailer, it often captures neighboring trailers beside the target trailer. This makes avoiding data contamination between different dataset splits difficult since each video usually captures multiple trailers.

One solution is to sort trailer footage by location and not mix footage from one spot between dataset splits. This would eliminate the worry of data leakage. However, it would also introduce a new problem, which is the lack of variety in the background. We did not manage to collect trailer images from enough places with varied backgrounds. Monotonous backgrounds may cause the model to learn to rely on the background to make predictions. Without diverse backgrounds, the model may not generalize well to different lighting conditions or in different landscapes.

3.3.2 Data Augmentation

One way to compensate for the relative lack of data is to utilize data augmentation. Both Roboflow and YOLO offers options to augment the dataset. For YOLO, it uses on-the-fly data augmentation. Each time an image is loaded during training, it is randomly augmented before being fed into the model. This means that the dataset is not modified or expanded on disk since the augmentations happen in memory at runtime. Therefore, applying data augmentation during training is easy with YOLO.

YOLO can activate several augmentation techniques that are built into the training script simply by launching it with the appropriate arguments. These methods include changing saturation, introducing noise, rotating the image, randomly erasing part of the image, and more. Since the augmentations are random, the model may

never see the completely unaltered raw image. However, since the same original image will appear in different augmented forms over multiple epochs, the model will hopefully learn to recognize the essence of the features in the original image.

3.4 Geometric Post Processing

After finding the corners of the trailer within the image, we need to use these corner positions to calculate the height difference h between the fifth wheel and the bottom edge of the trailer. For this, conventional geometric formulas will be used. This task can be subdivided into the following parts:

- Find the distance d from the camera to the trailer, which will be explained in Section 3.4.1 and 3.4.2.

- Find the angle θ_{5th} and $\theta_{trailer}$ between the ground plane to the fifth wheel and bottom of the trailer from the camera, and using these to find the height difference h, which we will talk about in Section 3.4.3.

Several parameters will be assumed as constants in our system. For example, the height h_{5th} and distance d_{5th} of the fifth wheel relative to the camera. The length of the wheelbase s_{wb}. Additionally, we treat the width W and height H of the trailer as constants, based on the ISO 668 standard for shipping containers [16].

Besides the assumed constants, we can also read measurements from the internal computer in the truck, such as the current height of the air suspension of the front and rear wheels, s_f and s_r. At last, there are the localized corner keypoints that we get from running the model on the backup camera stream. A sketch of many of the relevant measurements can be found in Figure 3.5.

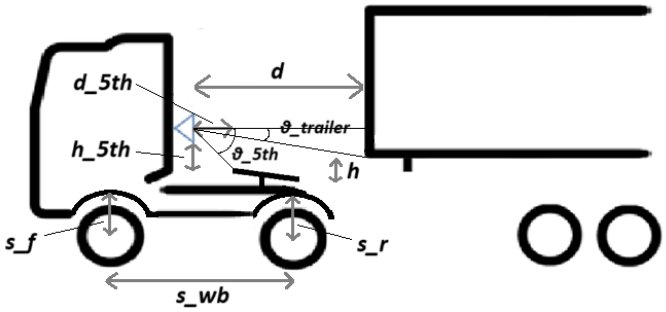

Figure 3.5: Measurements that will be used for the height estimation.

We have developed two methods for distance calculation, each with its own pros and cons, which we will now present.

3.4.1 Distance to Trailer Using Pixel Width

To calculate the distance d to the trailer, we use the pixel difference in the x-coordinate of the two bottom corners. To do this, we assume that the number of pixels in between the corners is proportional to the angle between the line segments starting in the camera and ending in the respective corners. For this relationship to be the same for all parts of the image, we must assume the fisheye camera model.

Once we have made this assumption, we need to find the relationship between number of pixels and angle. Our camera has a diagonal field of view of FoV $= 175°$. The camera resolution is 1280×720. Thus, the angle per pixel is

$$\Omega = \frac{2\pi \cdot \text{FoV}}{360°} / \sqrt{1280^2 + 720^2} \quad [\text{rad/px}] \,.$$

We find the width w of the trailer in the image in pixels from the vision model. Under the assumption that the trailer face is perpendicular to the camera direction, the following relationship should hold for the observed number of pixels:

$$\frac{w \cdot \Omega}{2} = \arctan\left(\frac{W/2}{d}\right),$$

where d is the length from the camera to the trailer. An illustration of what measurements these quantities measure in the physical setup is contained in Figure 3.6.

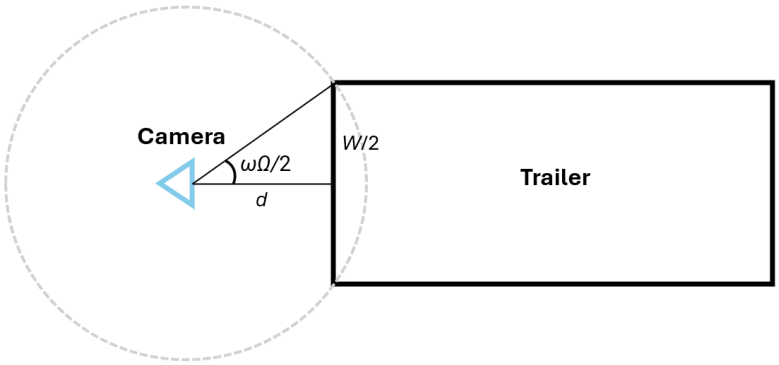

Figure 3.6: The measurements and angles involved in computing the distance to the trailer d.

Rearranging the terms to solve for the distance to the trailer, we find that

$$d = \frac{W}{2} / \tan\left(\frac{w \cdot \Omega}{2}\right).$$

Since the distance estimated using this method may deviate from the true distance d, we will now refer to the distance obtained from the pixel width method as d_{pixel}.

3.4.2 Distance to Trailer Using Perspective-n-Point

The camera is a fisheye type with an ultra wide-angle lens. To calibrate it, we used a printed checkerboard pattern and captured 10-20 images from different positions and angles with our camera. Then, the OpenCV calibration functions will automatically detect corners of the checkerboard and estimate the camera's intrinsic matrix and distortion coefficients, as shown in Figure 3.7. Once calibrated, we use OpenCV's

Figure 3.7: Checkerboard image before and after corner detection.

`solvePnP` function to estimate the position of the trailer relative to the camera. To do that, we define the 3D positions of the trailer corners in a local object coordinate system, assuming standardized dimensions. For a trailer of width w and height h, the corners are set at

$$(-W/2, 0, 0),\ (W/2, 0, 0),\ (-W/2, H, 0) \text{ and } (W/2, H, 0).$$

These 3D coordinates, combines with the 2D keypoints from the trailer and the camera calibration parameters, are used as input to `solvePNP`. This gives us the position vector t and orientation vector R of the camera. We can use this to calculate the distance d_{pnp} and angle of the camera.

3.4.3 Angle Between Horizon and Trailer Bottom

The angle between the horizon and the trailer bottom is computed by first finding the angle to the fifth wheel from the camera and then the difference in angle between the fifth wheel and the trailer bottom.

The angle between the ground plane and the line segment from camera to the fifth wheel can be expressed as the sum of two parts:

$$\theta_{5th} = \arctan\left(\frac{h_{5th}}{d_{5th}}\right) + \arcsin\left(\frac{s_f - s_r}{s_{wb}}\right).$$

The first term is the angle when the truck sits flat on the ground, and the second term is the angle induced when there is a difference in suspension height between front, s_f, and back, s_r.

To find $\theta_{trailer}$, we define $y_{trailer}$ and y_{5th} as the vertical pixel position (in image coordinates) of the fifth wheel and the bottom of the trailer respectively. Using these,

we compute the angle from the camera to the trailer bottom by first calculating the vertical pixel difference between these two points in the image. This pixel difference is then multiplied by the angle-per-pixel ratio Ω, and added to the previously calculated θ_{5th} :

$$\theta_{trailer} = \theta_{5th} + (y_{trailer} - y_{5th}) \cdot \Omega.$$

Then the height difference h between the camera and the fifth wheel or the trailer can be found with the respective distances and the sine of their respective angles:

$$h = d_{5th} \cdot \sin(\theta_{5th}) - d \cdot \sin(\theta_{trailer}).$$

4

Results

This section presents the results of our work, including the training performance of the YOLO pose estimation model and the practical evaluation of distance and height estimation. We begin by showing the model's training metrics, followed by an analysis of how accurately the system estimates the distance between the camera and the trailer, and finally, an evaluation of the height estimation between the fifth wheel and the trailer.

4.1 Vision Performance

We trained a YOLO pose estimation model, by fine-tuning the pretrained model `yolo11l-pose`, on 1000 images using a input image size of 640 $\times$640 pixels and a batch size of 16 for 200 epochs with data augmentation. The dataset split sizes were 0.7, 0.2 and 0.1 for train, validation, and test.

Figure 4.1 shows the training and validation losses for the bounding box of the trailer and pose estimation of the corners. The train box loss, which is how well the model predicts the bounding box around each object, consistently decreases throughout the training and ended around 0.4. While the validation loss plateaus after approximately 50 epochs at 0.8. The pose loss, which is the model's ability to accurately predict keypoint locations, shows a rapid drop during the early stages of training. In the end, both the training and validation pose losses converge to values around 0.15.

Figure 4.2 shows the precision and the recall score for pose estimation across 200 training epochs. The calculation of recall and precision is based on OKS, which where discussed in Section 2.5.3. Essentially, high precision indicates many of the detected keypoints where close to ground truth (OKS $\geq$ threshold), and high recall means that majority of the ground truth were correctly detected. After approximately 10 epochs, both metrics stabilize at high values, with precision consistently above 0.9 and recall around 0.85.

Table 4.1 illustrates the result of our model on the test dataset. The model achieved a pose precision of 0.918 and a pose recall of 0.887.

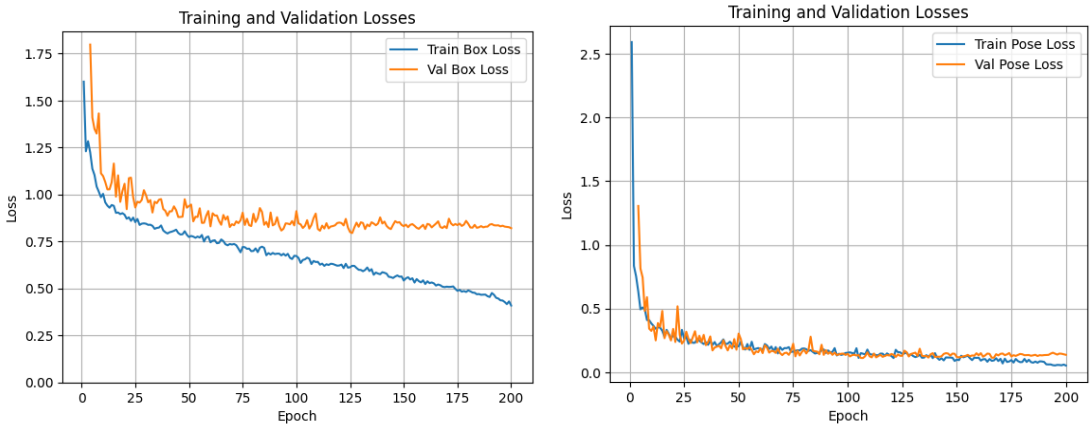

Figure 4.1: Training and validation losses for the bounding box (left) and the pose estimation (right).

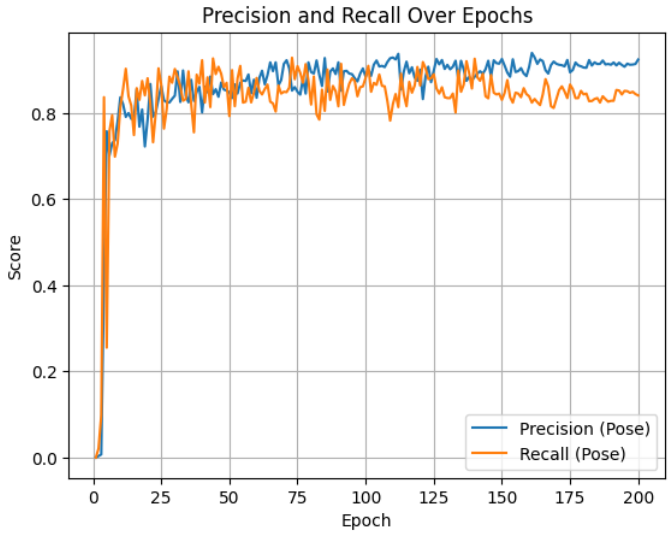

Figure 4.2: Precision and recall score of the pose estimation model.

Images	Trailers	Pose Precision	Pose Recall
101	239	0.918	0.887

Table 4.1: Model evaluation using test dataset.

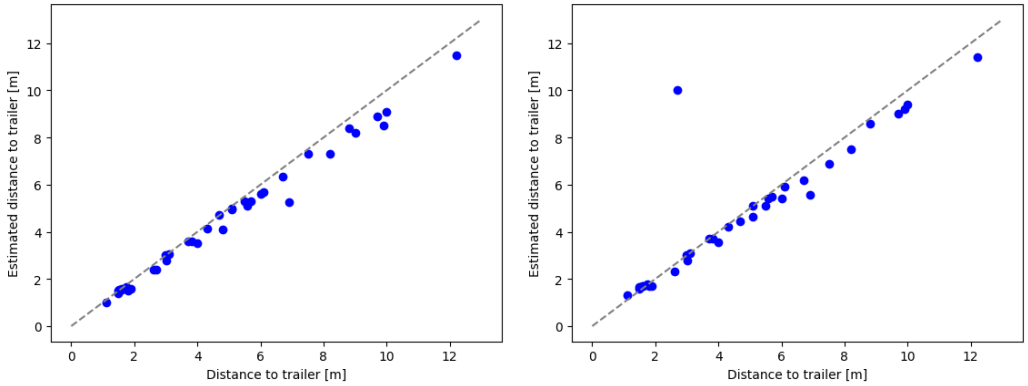

Figure 4.3: Relationship between the real distance from the trailer and the estimated distance from the trailer by pixel width method (left) and with solvePNP (right).

4.2 Geometric Performance

We evaluated our keypoint detection model on approximately 20 different trailers. In all but one case, the model successfully detected trailer corners. The single failure occurred with a specific trailer in one location, but when a trailer of the same model and color was tested in different location, the model performed as expected.

4.2.1 Distance Calculation Performance

The ground truth for the distance measurement was collected from different points during straight reversal sequences. The relationship between the estimated distance and the real distance to the trailer can be seen in Figure 4.3. The one outlier occurred when we used a trailer that did not have standard dimension. The graph for distances found by solvePNP underestimates the error rate, as there are a few outliers so large they lay above the region depicted in the plot.

To better visualize the relationship between the real and estimated distances, we also plot their logarithms, see Figure 4.4. This transformation is motivated by two key factors:

1. It is more important for our application to be accurate when near the trailer, as when the truck is close, it does not have time to make large adjustments.

2. The geometric formula used to estimate the distance becomes more sensitive to keypoint detection errors when the trailer is far away, i.e., when trailer is small in the image. Plotting the logarithm helps compensate for the disproportionally large impact of trailers that are far away.

The result of applying linear regression to the logarithm of the real and predicted values for both distance models can be seen in Figure 4.4. A few outliers prevent the regression model of solvePNP to be explanatory. After removing them, and

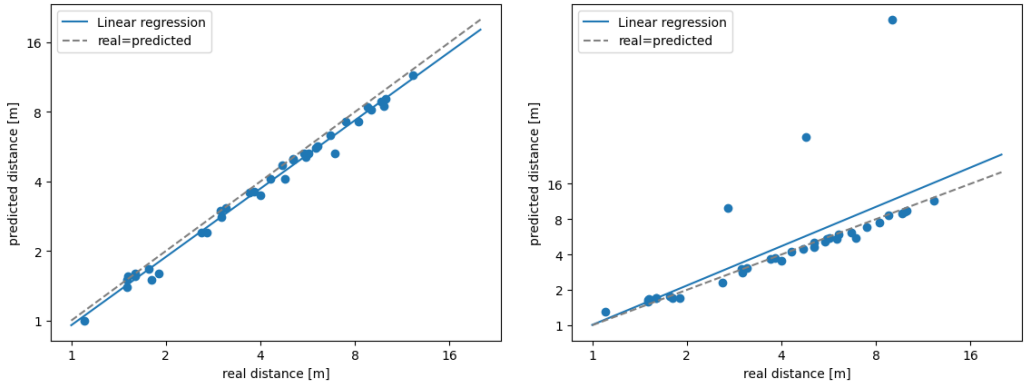

Figure 4.4: Linear regression on the logarithm of the measured and predicted distance values by our own method (left) and solvePNP (right).

recomputing the regression parameters for our model, d_{pixel}, and for solvePNP, d_{pnp}, we find the following relationships:

$$d_{pixel} \approx 0.96 d^{0.98} \quad \text{and} \quad d_{pnp} \approx 1.07 d^{0.92},$$

where d is the measured distance from the trailer.

Comparison of the linear regression coefficients found to the identity function, which if equal would indicate a unbiased estimate of the distance, we conclude with greater than 95% probability that the deviation in parameter estimates is not just due to random noise, but reflect a true bias in the model, or perhaps in the measurement of the ground truth.

4.2.2 Distance Calculation Performance when Viewed from the side

We also tested reversing the truck towards the trailer from the side to evaluate the performance of both distance estimation methods. However, due to time constraints and the fact that reversing sequences are typically performed in a straight line, we only collected a limited number of measurement points. The comparison between the pixel width method and the PnP method is shown in Figure 4.5.

The angles at which the truck reversed into the trailer were chosen randomly. Additionally, since our model was trained on a dataset consisting only of images where the truck reverses in a straight line, it was unable to detect the trailer's keypoints approximately 20% of the time when the trailer was viewed from an angle, therefore those results were not included.

4.2.3 Height Estimation Performance

To evaluate the accuracy of the height difference estimation, we first needed a reference value to compare against. This ground truth was obtained by backing the truck straight underneath a trailer and gradually raising the suspension until the

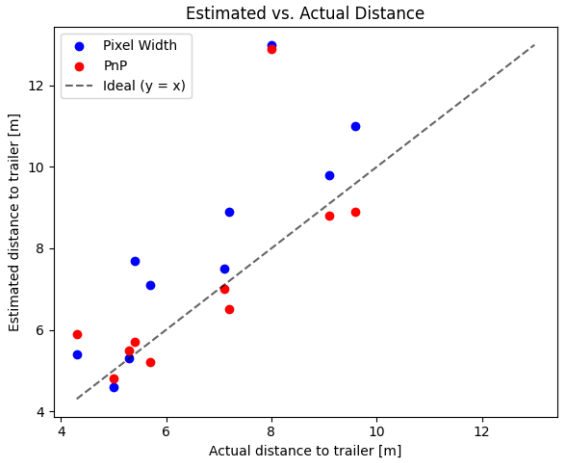

Figure 4.5: Distance estimation comparison when the truck is reversing into the trailer from the side.

fifth wheel made contact with the trailer. During this process, we logged the suspension height and rear wheel pressure for each frame on which the model was run. The logs where the truck reversed from the side were excluded because a coupling sequence is typically performed only when the truck reverses straight into the trailer.

The ground truth height difference for each frame was defined as the difference in suspension height between that frame and the final frame, where the fifth wheel and trailer had fully connected. This approach is relatively simple to automate, but has a limitation: if the trailer is light, it may be slightly lifted when the truck overshoots the target height, introducing a small error in the reference value. An example of the logged data from a single run is shown in Figure 4.6.

The sequence of events represented in the figure is as follows:

1. The truck begins around 6 meters from the trailer standing still. The automatic height adjustment is activated, and attempts to achieve a 15 cm margin of error between the height of the fifth wheel and the trailer.

2. At or around 31 seconds, the distance estimate crosses a threshold. At this point the control system switches from trying to maintain the 15 cm margin of error, to assuming the fifth wheel is under the trailer, and tries to go up.

3. After 40 seconds, the model detects that a threshold in the weight on the rear wheels is exceeded, and the program is set to maintain the current height.

It is not typical in real world scenarios that the truck stands still in place for extended periods of time, Our model does not rely on the truck standing still, it does take some time for the truck to descend enough to achieve the targeted margin of error.

As is reflected in the figure, the back of the truck is much faster to ascend, but slower to descend compared to the front. This is because there is much more weight on the front axle of the truck than at the rear axles. This is the main source of the forward tilt of the truck that the geometric model has to account for.

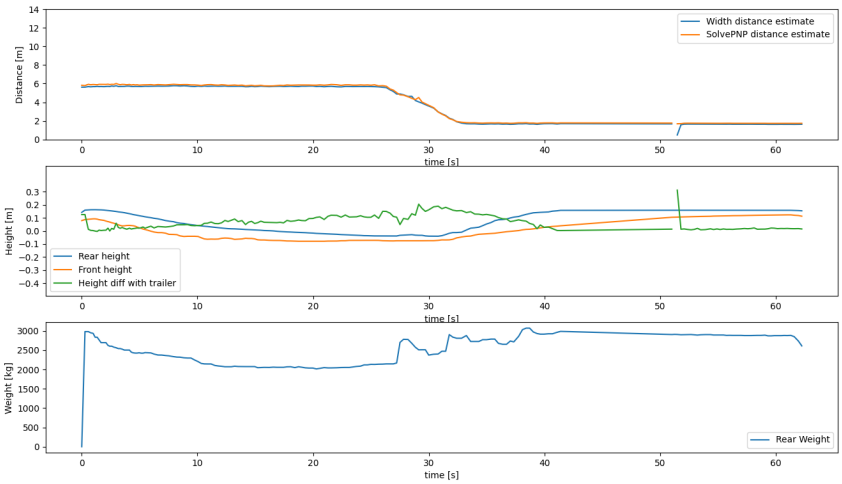

Figure 4.6: An example of the log from one backup sequence. It includes the distance estimates produced (top), the suspension height and height difference estimates (middle), and the weight on the rear wheels (bottom) for detecting when truck raised enough for fifth wheel to put pressure against trailer.

If the model is working properly, then the sum of the suspension height at the back and the estimated height difference should be constant, as any decrease in the actual height difference comes from the rear suspension being raised that same amount. The estimated height difference should also be zero at the end as the fifth wheel and trailer meet. Thus, the sum of the rear height and estimated height difference, minus the final value for the rear height should be zero if the model is working properly. We call this value the estimation error hereafter.

If we take this estimation error at different distances from the trailer for all the verification backup sequences we have collected, we can find the average error and the variation in the error for a given distance for all different trailers. This estimation bias and variance can be seen in Figure 4.7.

When interpreting this figure, it is important to note that all the recorded backup sequences did not start and end at the same distance. Observing the portion of backup sequences containing different distances, we must conclude that we can have confidence in our estimation bias measurement within approximately the range 1.4-7.0 meters.

A problem we must account for here is that if we stand still in one place when collecting data, the model will run many times at that distance for that particular trailer, making the bias estimate biased by this. The way we counter this is by taking the mean height error for every 20 cm interval in distance from the trailer, and then taking the mean of these averages across all backup sequences. In this way, every video has the same impact on the height-bias estimate for that given distance.

When doing this, it is essential to not accidentally lose the estimation error variance between different video frames from the same trailer. We assume that the total

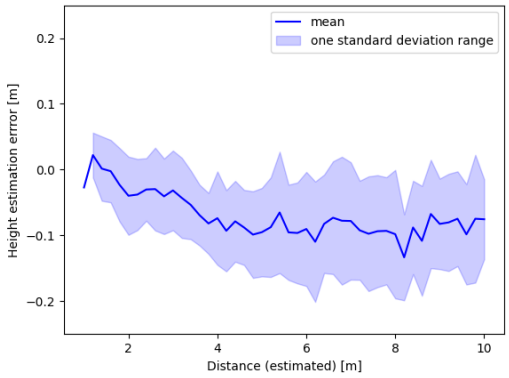

Figure 4.7: Average height estimation bias as a function of the estimated distance to the trailer.

estimation error is the sum of some random error specific to that trailer & distance, and one part due to errors from random variations between images of the same trailer & distance. Under the assumption that these errors are independent, the variance of the error is the sum of the variances of the mean error for different trailers plus the variance for different pictures of the same trailer. Another way to state this is that we assume that the covariance between the two is zero.

We use this assumption of independence because we do not see any reason why they should be dependent, and even if there is some correlation, we have no idea how to estimate it.

5

Discussion

5.1 Interpretation of Training Result

Figure 4.1 shows a promising decline in both the box loss and the keypoint detection loss. For the box loss, the training loss continues to decrease, while the validation loss plateaus after approximately 75 epochs. As training progresses, the model increasingly fits the training data and may begin to overfit, particularly given the relatively small dataset. One possible explanation for the validation loss plateau is the lack of a strict standard for how bounding boxes were labeled; boxes were only required to enclose the trailer, without specific requirement on how the bounding boxes should look like. This inconsistency makes it difficult for the model to achieve a low IoU loss on the validation set. Nonetheless, the model still manages to reach a reasonably good IoU score.

Training and validation loss for the pose decreases throughout the training and stays close to each other. This means the model is learning without overfitting and thus generalizes well to unseen data. This is also shown in Figure 4.2 and Table 4.1, as precision and recall stays relatively stable throughout both training and in the test dataset. The model consistently detects trailer keypoints with few false positives and few false negatives (missed detections). One possible reason for the slightly lower recall is that many images contain multiple trailers. As the truck approaches the target trailer, other trailers in the background will often be partially cropped out of the frame. In such cases, only a subset of their corners remain visible. These incomplete views make it more difficult for the model to detect the keypoints at all, which may lead to a higher number of missed detections and thus lower recall. However, this is not a problem since only the keypoints for the target trailers are relevant during the reversing sequence.

5.2 Distance Estimation

From Figure 4.3, we see that both distance estimation methods perform well. Often times, the estimated distances falls within a 10% error margin. In general, both methods gives similar results, meaning that the choice of which distance estimate to use has small impact on the final height estimation.

However, each method has its own strengths and limitations. Both methods need

to assume a standardized dimension of the trailer. However, to compute d_{pixel}, we only need to assume the width of the trailer, whereas the PnP method requires assumptions about both width and height. Based on our observations, trailer widths vary less than heights, making the PnP method more prone to outliers, as illustrated in Figure 4.3.

On the other hand, the pixel width method assumes that the truck is reversing directly toward the trailer, which may not always be the case. In contrast, the PnP method estimates the spatial relationship between the camera and the trailer, which can handle more varied reversing angles and positions.

Finally, both methods require camera-specific parameters, pixel width method requires camera's FoV, while the PnP method requires a calibration. The latter may involve more work but gives more information, including FoV.

Interpretation of Regression Parameters for Logarithm of Distance The fact that the multiplicative constant is not one for the estimated width regression, $d_{pixel} \approx 0.96x^{0.98}$, is likely due to imprecise constants in the geometric model, and could likely be corrected with relative ease.

The result for solvePNP, $d_{pnp} = 1.07d^{0.92}$, is a little more difficult to explain. In the width estimation with solvePNP we have to solve the problem of the top two corners going out of frame when we are closer than ≈ 3.7 m, Our solution to this was to project where we think they would be based on the height / width ratio of the trailer. This is imperfect, especially considering the distortion caused by the camera, and could explain a shift in the behavior of the distance estimator, and thus also the deviation of the exponent from one. This is of course very speculative, but we do not see any other phenomena which we think would give rise to such a behavior.

5.2.1 Distance Estimation when Approach from Side

From Figure 4.5, we observe that distance estimation using the PnP method yielded better results than the pixel width method. This outcome is expected, as PnP is theoretically more robust when the trailer is viewed from an angle. However, many of the measurements were less accurate compared to those obtained during straight reversing. We believe that this is due to the model's difficulty in detecting the trailer's face from the side, a scenario underrepresented in the training data. As a result, in many cases, the trailer was undetectable or the detected keypoints were inaccurate.

If keypoint detection were more accurate, which can be achieved by training on a larger and more diverse dataset, the PnP method would likely provide reliable distance estimates from all angles. However, within the constraints of this project and the current dataset, this is not the case. Given the extra assumption about the height, and the fact that distance estimation from an angle remains somewhat unreliable even for PnP, we conclude that the pixel width-based method is currently the more dependable approach.

5.3 Height Estimation

Our results indicate that, when working, the height estimate is consistent within a range of 8-15 cm depending on the distance to the trailer. Although not perfect, the automatic height adjustment performs as intended in most cases, with the exception of situations where the trailer was positioned too high or too low for the suspension to compensate.

Bias in the Target Value for the Height Difference Right now the height of the trailer is taken to be the rear suspension height at the end of the backup sequence. If the fifth wheel just about touches the bottom of the trailer, this is very accurate. If, on the other hand, the fifth wheel pushes up enough to lift the trailer the height of the trailer will increase at the end creating a bias when compared to the height difference earlier.

Such lifting on the scale of maybe five centimeters was observed several times in our recordings, and this likely accounts for some part of the negative bias observed in the height estimate result in Figure 4.7. Consequently, the real accuracy of the height estimate is likely slightly better than what the results seem to suggest.

There is a downward trend in the average estimation error that needs to be explained. We think that this most likely arises from some error in the constants of the geometric model, such as the pixel coordinates of the top of the fifth wheel, or similar.

Variance of the Height Difference Estimate Since the geometric model is deterministic, the variance of the height difference estimate can only arise from inconsistency in keypoint detection. The height difference is computed as the distance to the trailer multiplied by the sine of the angle between bottom of the trailer and the horizon. Thus, we may expect that the standard deviation would grow linearly compared to the actual distance from the trailer if the error is consistent in terms of pixels in the image, but this does not appear to be the case.

The standard deviation increases with distance, but not at a rate that would affirm a linear relationship between it and the distance. This would indicate that the accuracy of the keypoints counted in pixels seems to get better for trailers that are further away. We know of two ways such a trend could arise.

- When we are further away from the trailers, more of them fit into the picture, meaning there are more samples at this distance in the training set.

- YOLO's loss function punishes deviations from the labeled keypoint locations in inverse proportion to the area of the size of the bounding box around the detected trailer. This means that small bounding boxes from trailers far away demand more precision for the same loss.

Summary The height difference estimate works well enough for the automatic height adjustment algorithm to work in a clear majority of the tested cases. The average height difference error estimate is biased, which could arise from small errors in parameters from the geometric model and a disturbance in the label for the target height caused by the trailer being ever so slightly lifted at the end; fortunately, this bias appears to be small enough not to prevent the functioning of the automatic height adjustment. The variance in the height difference estimate increases as you get further away, but not nearly as fast as one may have had reason to fear.

5.4 Comparison with Previous Technologies and Implication

The method developed in this project is similar to Volvo's previous solution, which relied on ArUco markers for determining spatial positioning. To enable automatic height adjustment, it is essential to know the positions of both the trailer and the fifth wheel. In Volvo's earlier implementation, ArUco markers were placed on both components, enabling accurate 3D pose estimation. Since ArUco markers are designed to be easy to detect and come with known dimensions, the resulting pose estimations were highly accurate.

However, a major limitation of their approach is the assumption that every trailer in operation would be equipped with ArUco markers, since trucks couple with trailers from different companies, it would be impossible to have that assumption. In contrast, the method developed in this project avoids that dependency by detecting the corners of the trailer and use either our pixel width method or our PnP method to estimate the spatial relationship between the camera and the trailer. Then, the position of the fifth wheel relative to the camera is assumed to be known. However, this is not a perfect solution, as the camera's mounting position can vary depending on the truck model, and the fifth wheel itself may also be adjustable or mounted in different positions.

While requiring trailers to carry ArUco markers is unrealistic, equipping the truck itself with such markers is far more feasible. Since Volvo manufactures the truck, it would be relatively simple to place an ArUco marker near the fifth wheel, allowing the camera system to accurately detect its position. This approach of using computer vision for trailer detection and a marker for the truck could provide a robust and practical solution, especially considering that Volvo is also the intended user of this technology.

5.5 Limitations of the Study

Data Access We would have wanted more data both for training and verifying our model. More pictures of trailers seen at an angle, rather than straight from the back would be beneficial for expanding the capability of our model to deal with this

case. Due to time constraints, we only got verification data from one location with trailers mostly from the same company and with similar visual features. Expanding the verification set in particular would be needed to give us greater confidence in our results.

The ideal case for data collection for us, would have been collecting data from trucks operating as they usually do, rather than us collecting a few logs, all from the same few locations. This would give us a larger and more varied dataset for training, and if testing our automatic height adjustment feature, give us an indication if the suspension is adjusted at a sufficient pace to keep up with the speed at which a typical truck driver attempts to couple with the trailer. This data would be difficult to collect for a variety of reasons.

Firstly, GDPR rules mean that we would have to get permission from each company whose premises we would collect data from. This means that we could probably only collect data from trucks where we know beforehand which companies they are transporting to / from. Another challenge would be creating the hardware setup to perform this data collection automatically and especially performing all the safety verification required for features shipped to end users driving on public roads.

Due to the infeasibility of solving these problems within the project, we have been limited with regards to variations in how the trailers look.

Data Sample Bias As has already been mentioned, all data was collected during the same time of year with quite small variation in weather. We have not attempted to test the model at night or in low light conditions. While we may safely assume that this has a detrimental impact on our models performance in these other conditions, we currently have no way of estimating how large the impact is.

Model The pose model we used was pretrained on features of the human body. This means that the features it was trained to look for was quite different, and the training pictures was often from an environment quite different from a road environment or industrial parking lots. Had we found a model pretrained on a dataset for self-driving, we may suspect that performance would have been better.

In the beginning, we also planned to incorporate detection of the fifth wheel from the image, instead of hard-coding a location within the image. This would simplify the adaptation to a different position for the camera or for the fifth wheel, and lead to a more flexible model overall. Had we attempted this, we would have to deal with the fifth wheel being sometimes hidden by the trailer, and we would need to perform more labeling.

5.6 Methodological Considerations

Choice of Analyzing Logarithm of Distance Choosing to analyze the logarithm of the distance instead of the distance itself is not a design choice without downsides. For example, using a laser distance meter from the back of the trailer

pointing towards the camera, we can get a measurement error of a decimeter or two from hitting the back wall of the truck beside where the camera is mounted. The variance of this measurement error would not grow as we go further from the trailer but would remain constant. Therefore, there is a risk that there is a disproportionally large error in the logarithm of the distance when we are close to the trailer, which may have a significant impact on the logarithmic model.

Observing the results in Figure 4.4, we do not see a larger variance when close to the trailer, which would indicate that this potential source of bias would not have a significant impact on the results.

Software Setup Although our software setup ended up working, if replicating a project like this, it would be preferable to either

1. use a Linux computer instead of working with a virtual machine, or
2. design the software with windows in mind from the start.

While working around the limitations of WSL was possible, working out driver installation, forwarding of USB devises, and firewall issues was a lot of work compared to the benefits; the code for training the neural network and running it in the truck was separated anyway, and thus had no need to target the same operating system.

Choice of Evaluation Metrics The choice of using height estimation bias and variance as metrics to evaluate the model was great for verifying the functionality of our system, as the height is the thing being adjusted. As such, it is all that really matters to the end user.

The other metrics, along with checking the system output at intermediate steps in the process, can serve as helpful hints when debugging or further developing the system, thereby justifying the recording of and attention to these metrics.

5.7 Future Directions

Automatic Breaking

In cases where the truck has not been lowered sufficiently, it may crash into the trailer it is backing in towards, and cause damage to the fifth wheel or the battery. In ports and freight terminals where trailers are frequently attached and detached, often in a hurry, this is a very common cause of damage to the tractors. In such a case, breaking automatically if approaching a trailer fast while being too low would prevent a lot of accidents, and resulting repair costs.

Such a feature would require much more testing and verification because anything controlling the breaks could present much more danger if it malfunctioned.

There is also the possibility of controlling the truck in other ways. For example, using the keypoints in combination with perspective-n-point you could try to steer the truck to correctly align with the trailer automatically when backing in.

Alternative Machine Learning Model

Our original approach At the beginning of our work, we were trying to develop another kind of model which was going to use semantic segmentation and per-pixel depth estimation instead. We shifted away from this approach in favor of the method outlined in this report because it would be simpler, but the model presented makes very strong assumptions about the shape of the trailer.

Although container trailers are very standardized, there are other types of trailers, like tankers for liquids such as gasoline or milk, or empty trailers, which the architecture described above cannot handle. This is not a limitation in the training data we use, but a consequence of the geometric post-processing we have to do on the output of our model.

The model we initially tried may be more difficult to successfully implement and tune, but has no inherent constraints on shape. Thus, we recommend considering it for future work on automatizing trailer coupling. A more detailed explanation and documentation of the progress we have made are contained in appendix A.

End to end model Another so far unexamined approach worth mentioning is a end to end system where you collect many more backup sequences while logging the height, and training a model to predict what height adjustments a human would make in either the next few seconds, or the total change until the end of the backup sequence.

If values such as distance traveled was incorporated and the "end-time" was set to the time when the trailer is connected, then the labeling of training data for this model could be performed automatically. If these data were uploaded to the cloud, fine tuning of the model could be performed automatically. This would greatly improve the model over time as training data from different regions, climates, seasons, etc. could be incorporated automatically.

6

Conclusion

A method for automated height adjustment for truck–trailer coupling has been developed and demonstrated in this thesis. The proposed solution is based on computer vision and uses a rear-mounted backup camera along with a keypoint detection model based on YOLOv11 to estimate the height difference between the trailer and the truck's fifth wheel. The system first estimates the distance to the trailer, and then uses this distance to infer the height difference by assuming a standardized dimension of the trailer and a fixed position of the fifth wheel.

Distance estimation was performed using two methods, both of which showed promising results with an average error within 10%. Based on this, the height estimation achieved an accuracy of approximately 7–15 cm, with performance improving as the truck approached the trailer. The complete system was tested on real truck–trailer coupling sequences in a controlled environment. It was also tested by a highly experienced truck driver, who provided positive feedback on its usability.

Despite being trained on a relatively limited dataset, the model achieved high precision and recall in detecting trailer corners. The results demonstrate that using machine learning for automated coupling assistance is both feasible and effective. This proof-of-concept lays a solid foundation for further development, and with expanded data collection and integration into the truck's electronic control unit, it has potential to serve as the basis for a future production-ready solution.

Bibliography

[1] Yousef Atoum et al. "Monocular Video-Based Trailer Coupler Detection Using Multiplexer Convolutional Neural Network". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5478–5486. DOI: 10.1109/ICCV.2017.584.

[2] B. Dwyer et al. *Roboflow (Version 1.0)*. Computer vision. 2024. URL: https://roboflow.com.

[3] FFmpeg Developers. *FFmpeg Documentation*. Accessed: 2025-04-22. 2025. URL: https://www.ffmpeg.org/ffmpeg.html.

[4] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. *Ultralytics YOLO*. Version 8.0.0. Repository: https://github.com/ultralytics/ultralytics, License: AGPL-3.0, Released: 2023-01-10. 2023. URL: https://ultralytics.com.

[5] Rahima Khanam and Muhammad Hussain. *YOLOv11: An Overview of the Key Architectural Enhancements*. 2024. arXiv: 2410.17725 [cs.CV]. URL: https://arxiv.org/abs/2410.17725.

[6] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[7] Luca Medeiros. *lang-segment-anything*. https://github.com/luca-medeiros/lang-segment-anything. 2024.

[8] OpenCV Developers. *Camera Calibration*. Accessed: 2025-04-01. 2025. URL: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

[9] OpenCV Developers. *Perspective-n-Point*. Accessed: 2025-04-01. 2025. URL: https://docs.opencv.org/3.4/d5/d1f/calib3d_solvePnP.html.

[10] *Perspective-n-Point Wikipedia*. 2024. URL: https://en.wikipedia.org/wiki/Perspective-n-Point.

[11] Elad Plaut. *Fish eye camera model*. 2024. URL: https://plaut.github.io/fisheye_tutorial/.

[12] S Nikhileswara Rao. *YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy*. 2024. URL: https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71 (visited on 03/24/2024).

[13] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[14] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV]. URL: https://arxiv.org/abs/1801.04381.

[15] Ultralytics. *Ultralytics Pose*. 2024. URL: https://docs.ultralytics.com/tasks/pose/.

[16] Wikipedia contributors. *ISO 668 — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-May-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=ISO_668&oldid=1273977711.

[17] Wireshark Contributors. *Wireshark: Network Protocol Analyzer*. Accessed: 2025-04-22. 2025. URL: https://www.wireshark.org.

[18] Lihe Yang et al. *Depth Anything V2*. 2024. arXiv: 2406.09414 [cs.CV]. URL: https://arxiv.org/abs/2406.09414.

[19] Lihe Yang et al. *Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data*. 2024. arXiv: 2401.10891 [cs.CV]. URL: https://arxiv.org/abs/2401.10891.

A

Future Work – Alternative Model

The current model has the flaw that it makes various assumptions about the shape of the trailer. While this makes things simpler, it also means that the model does not generalize to less common, more oddly shaped trailers.

Before trying the keypoint estimation approach, we tried the approach of making a neural network that segmented the image to detect the trailers and fifth wheel, and estimate the distance to each pixel visible in the image, all in one neural network.

We believe that if this feature should be applicable to all kinds of trailers, further exploration of this approach may be necessary. Therefore, we will present our work on this model even if it has never been integrated into our height approximation program, and does not yet have sufficient accuracy. While we got some results from this model, which we will show, no systematic evaluation before this model was shelved in favor of keypoint estimation with YOLOv11.

This model consisted of the encoder part borrowed from MobileNetV2 [14], and two separate custom made heads for pixel-wise depth estimation and semantic segmentation.

A.1 Encoder

The role of the encoder in a neural network is to extract features from the image which can be used later for other tasks such as image classification, object detection, segmentation, and more. While you train the weights of the encoder for a specific task, the features learned will typically be useful for other tasks aswell.

As the encoder part both generalizes and typically contains a large percentage of the total weights in a model, re-using the encoder from another network is a common approach in deep learning when one does not have a lot of data or need to minimize training time.

MobileNet in particular was chosen because it has a relatively low computational cost, and the encoder part of the network is clearly separated from the head making reusing it simple.

A.2 Skip Connections

MobileNet's encoder is built in such a way that early on, each convolutional layer has a low number of channels, and high resolution. As you go deeper in the network, resolution is decreased to reduce computational cost but the number of channels is expanded simultaneously to represent more feature types and their interactions.

We want our output to have high resolution, and the heads also need more high-level features, we want to incorporate information from both the early and later layers of the head.

The way we solve this is by introducing *skip connections*, which means using the output of multiple layers and combining them. This combination is done by resizing them to a common resolution, and concatenating along the channel dimension. In our implementation a few convolutional layers are applied at each level before resizing them to a common resolution, which is a small departure from the working of the typical skip connections, but the role and purpose is the same.

A.3 Segmentation and Depth Heads

The segmentation and depth heads are standard convolutional neural networks with ReLU activation function. The segmentation head has 3 channels in the final layer (equal to the number of classes: background, trailer and fifth wheel), and softmax applied along the channel dimension as segmentation is categorical.

The depth head has one channel for the final layer and the ReLU function as the activation function. The output of this layer represents the (multiplicative) inverse of the distance to an object. The logic behind this is that a difference of (for example) 1 meter gives a much greater impact when comparing objects that are near the camera, whereas a one meter difference for objects far away may be near imperceptible.

After abandoning this approach to the problem, we noticed a small problem with our code, which was never corrected. In the segmentation head we do not skip applying ReLU in the last layer before applying softmax2d. This may be in some part responsible for the relatively poor performance of this model.

A.4 Training Data

Creating depth and pixel-wise segmentation labels for all images was completely unfeasible. Therefore, we tried using a technique called distillation, which means using the output from another network as pseudo-labels that you can train on and try to match. For segmentation labels, we tried using Segment Anything by Meta AI [6], and explored the possibility of using the derived model lang-segment-anything [7]. For depth labels, we tried using Depth Anything v2 [18]. Depth Anything also provides an appropriate loss function for depth estimation [19].

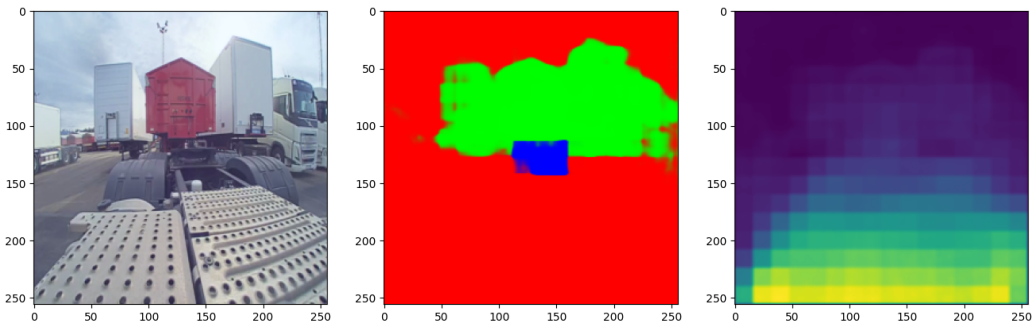

Figure A.1: Input image (left), with estimated segmentation (middle) and depth (right) for an example image.

A.5 First Preliminary Results

The results we obtained from this method without further improvements were certainly not good enough. A representative example of the model input & output is shown in Figure A.1. You can see that the network correctly finds that the back of the truck is closer than the rest, you can see that the trailer looks slightly closer than the surroundings, but it is nowhere near accurate enough.

A.6 Areas of Exploration

Had we continued with this approach, the first thing we would have examined is how the loss function could be tweaked in a way that would promote accuracy at the edges of the objects in particular. We would also experiment with the weighting of the respective classes in the loss function, and try to see if/how the network architecture could be adjusted so that the network makes better use of the high resolution it has in order to remedy the blockiness we observe in the result.

Since the labeling network is trained to give a relative depth map, that is not to have the same scale for every image, we would also have to explore how a known distance to some reference point on the truck could be used to map the depth output to actual distance.

We also think it is worth exploring whether there are other base models, such as vision transformers or similar, that could give better results. We also would have to consider if it is beneficial to let the training adjust the weights of the base model or only the head. Doing so would give the models more flexibility and a better ability to learn, but also increase the probability of overfitting on our small dataset, and increase the time required to train the model.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY