Semi-Supervised Learning for Autonomous Vehicle Object Detection

Master's thesis in Computer Science - Algorithms, Languages & Logic

NIKLAS GUSTAFSSON
GUSTAV RÖDSTRÖM

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Semi-Supervised Learning for Autonomous Vehicle Object Detection

NIKLAS GUSTAFSSON
GUSTAV RÖDSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Semi-Supervised Learning for Autonomous Vehicle Object Detection
NIKLAS GUSTAFSSON
GUSTAV RÖDSTRÖM

Supervisor: Lennart Svensson, Department of Electrical Engineering
Advisors: Erik Werner & Dónal Scanlan, Zenuity
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Tracking visualization constructed in Python showing vehicles being tracked.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Semi-Supervised Learning for Autonomous Vehicle Object Detection
Niklas Gustafsson & Gustav Rödström
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous driving (AD) has the potential to make transportation safer and more efficient. Perception is a key challenge in making AD a reality, as it is needed to operate the vehicle safely. During recent years, Deep Neural Networks (DNNs) have been used to achieve great results in perception tasks relevant to AD such as object detection [13] and semantic segmentation [5]. However, these networks are commonly trained within the supervised-learning framework, requiring large, manually annotated, datasets.

In this thesis, we investigate an alternative approach, called semi-supervised learning (SSL). The idea is to use a small annotated dataset in conjunction with a large unannotated dataset. Since less manually annotated data is required, SSL reduces annotation cost. Furthermore, it allows very large unannotated datasets to be used, which could boost performance.

Specifically, this thesis focuses on object detection for autonomous vehicles. Two new SSL methods are presented; *Noisy Student for Object Detection* and *TOFS (Tracking Objects For Self-supervision)*. The first is an adaptation of the work in *Self-training with Noisy Student improves ImageNet classification* [22], which uses noise on annotated and unannotated data to train a neural network for image classification. The second method is based on Multiple Object Tracking algorithms, which attempt to leverage the time dependency across video frames.

We show an increase in Average Precision Score by about 1 percentage point when training with *Noisy Student for Object Detection*, showcasing the potential of unannotated data. For *TOFS* we show that the depth, height, length and lateral position predictions are improved compared to the predictions of a baseline network. Lastly, we show that incorporating lidar measurements greatly improves the predicted depth.

Keywords: Autonomous Vehicles, Semi Supervised Learning, Deep Learning, Object Detection, Multi Object Tracking.

Acknowledgements

First and foremost, we would like to thank our academic supervisor Lennart Svensson. Lennart has been very enthusiastic about the project and has been a constant source of new ideas and inspiration. Furthermore, he has been crucial in pointing us towards relevant research and has been very patient in our discussions.

We would also like to thank our supervisors at Zenuity; Erik Werner, Dónal Scanlan, and advisor Adam Tonderski for their support. The project would not have gone nearly as smoothly or been as interesting, were it not for their good advice. They have given us great advice and ideas on improvements. Though most importantly, they have asked critical questions throughout.

Furthermore, we would like to thank Zenuity AB and all employees for extensive support throughout the project, as well as access to good resources. Especially for the coffee, Rocket League, and 'fika', but primarily for the opportunity to conduct the thesis at the company.

Lastly, we would like to thank Alfred Gunnaregård and Daniel Odin for their support, feedback, and great company. It has been a true pleasure to conduct our thesis alongside them.

Gustav Rödström & Niklas Gustafsson, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1

Introduction

In recent years, Autonomous Driving (AD) has gained much attention. It has the potential to make transportation safer, more efficient, and cheaper. Perception of the surrounding world is a key component in making AD a reality since it is necessary for operating the vehicle safely. Vehicles perceive the world by applying algorithms on data from sensors such as cameras and radars on the vehicle. In particular, it is common to use Deep Neural Networks (DNNs) to perform such perception tasks.

DNNs are able to achieve great results in several perception tasks such as image classification [22], object detection [13] and semantic segmentation [5]. Historically, a lot of this success builds on access to large, manually annotated, datasets used to train the networks. Since DNNs have a large number of trainable parameters, they can easily overfit during training, which is why it is crucial to use a large dataset. An example of such a dataset is ImageNet [7]. It contains more than one million annotated images and is the result of a massive amount of manual work.

Manually annotating large datasets is costly. This has sparked an interest in approaches that rely less on such data. One approach is Semi-Supervised Learning (SSL), which uses a small annotated dataset and a large unannotated dataset in conjunction during training. This allows for the use of a large dataset, while only a fraction of it needs to be annotated. Furthermore, obtaining a large unannotated dataset is cheap, and in several cases, they already exist. For example, ImageNet is vanishingly small compared to the average number of photos uploaded to Facebook per day, which was 250 million by the end of 2011 [1].

Recently SSL has been used to achieve great results in several tasks [12, 15, 19, 21]. Inspired by the recent success of SSL, this thesis will apply the semi-supervised framework to the task of object detection (OD). The thesis is conducted in cooperation with Zenuity AB, an Advanced Driver Assistance Systems (ADAS) and AD software company. A fraction of the frames in Zenuity's large collection of video recordings are annotated and used for training. Changing the models to train in a semi-supervised way has the potential to increase the amount of training data used significantly. This should lead to an increase in performance, and in turn, increased safety. It would also reduce the need for manually annotating additional data.

1.1 Background and Objective

As previously mentioned, recent work such as [12, 15, 19, 21] achieves great results using various semi-supervised methods for classification. Inspired by that, this thesis aims to apply SSL to Object Detection (OD) in the domain of autonomous vehicles.

Using SSL for OD seems to have great potential. For example, obtaining an hour of unannotated data is simple since it only involves recording an hour long driving session while annotating it would require a huge amount of manual work. Thus, if the unannotated data could be used in training, the network could train on a vastly larger dataset, which should increase performance, while also reducing the need for manually annotating data.

Therefore, the overall goal of this project is to answer the question: Is it possible to improve OD performance using semi-supervised learning?

Formally, the problem can be defined as follows. Given a set of annotated data L and some detector D trained on L, are we able to increase the performance of D if we are given additional unannotated data U, from the same distribution as L, by training D on both L and U?

1.2 Motivation

In the previous section, we stated that the overall goal is to investigate if it is possible to improve OD performance using SSL. As was also mentioned, existing methods were mainly designed for other tasks. Below we further motivate this thesis by explaining why existing methods are not optimal for OD in the AD domain.

Most of the work in SSL is focused on other tasks than OD. For example, recent work such as [12, 15, 19, 21] focus on classification tasks. In contrast to classifying an image, 2D OD consists of predicting a 2D bounding box around each object in the image, while also predicting the object class. In 3D OD, the 3D size, position, and rotation are to be predicted as well. These are both harder tasks. Therefore, applying existing semi-supervised methods to OD is non-trivial, and would require modifications. It is not clear to what extent this is possible or how well it would work.

Not only are most existing methods focused on other tasks, but also tasks involving independent samples. In contrast, object detection within AD is performed on images which are consecutive frames in a video. The frames are dependent across time, since several objects in a given frame are likely to appear in the consecutive frame as well. It makes intuitive sense that instead of treating the frames independently, a semi-supervised method could benefit from leveraging the time dependency, as multiple observations of the same object should be more useful than one. Therefore, it makes intuitive sense that a semi-supervised method for OD should leverage the time dependency.

1.3 Method outline

In the previous section, we explained the difficulties of applying existing semi-supervised methods to OD. We also argued that it would make intuitive sense for a semi-supervised method for OD to leverage the time dependency between video frames. In this section, we outline two semi-supervised methods for OD proposed in this thesis. The first method is heavily based on an existing method for image classification, while the second method is new and attempts to leverage the aforementioned time dependency by using Multiple Object Tracking (MOT).

The idea behind the first method is to adapt an existing state-of-the-art semi-supervised method for image classification to OD. The noisy student algorithm [22] was used as a foundation to create a version of it adapted to OD. The algorithm uses a trained network to infer pseudo-annotations on unannotated data, which is then noised and used for training.

The second method aims to exploit the time dependency across frames. To do this, it is natural to use MOT algorithms. Given a sequence of frames and a number of detections in each frame, an MOT algorithm outputs object trajectories, which are estimates of the object state across time. As the tracking is done offline, meaning we train after all detections are inferred, all detections can be used to estimate all states, which allows the algorithm to smooth the trajectory. Such trajectories are then be used to create pseudo-annotations for the corresponding images, which the network trains on. The idea is that by taking multiple detections into consideration, the algorithm has more information, and should be able to create pseudo-annotations of higher quality.

1.4 Limitations

The main focus of this thesis is to create and evaluate the two methods proposed in the previous section. Therefore, the main limitation is that a DNN provided by Zenuity will be used as is, and modifying the architecture is considered out of scope.

1.5 Contributions and Main Results

The main contribution of this thesis is two methods for applying the semi-supervised framework to OD; *Noisy Student for Object Detection* and *TOFS (Tracking Objects For Self-supervision)*. The two methods are evaluated, and the main results of the evaluation are as follows.

1. Noisy Student for Object Detection is used to achieve an increase of one percentage point in APS compared to the baseline.
2. TOFS is used to create pseudo-annotations which are clearly better than the output of the network, showing the potential of the method.

2
Theory

This chapter provides the technical background necessary to understand the two semi-supervised methods for object detection proposed in this thesis, as well as the motivation behind the thesis as a whole. We begin by introducing supervised learning, which has historically been the dominant method for training DNNs. We continue with semi-supervised learning, a different method for training DNNs, which relies less on labeled data, and is able to utilize large amounts of unlabeled data. Two paradigms of semi-supervised learning as well as the teacher-student framework are presented. Then, Multiple Object Tracking (MOT), which the second method relies on, is introduced. We proceed to present the problem that MOT attempts to solve and its integral parts such as measurement and motion models, hypothesis reduction, and sets of trajectories. Lastly, we present a complete MOT algorithm, a Trajectory Poisson Multi Bernoulli (TPMB) filter, used by the second method.

2.1 Supervised Learning

In supervised learning, the goal is to approximate a function f using a machine learning algorithm. The function maps elements from an input domain X to elements in an output domain Y [14]. For example, the input domain could be images of pedestrians, and the output domain could be bounding boxes. In order to approximate f, the machine learning algorithm is given a set of pairs of inputs and their corresponding ground truth outputs denoted $L = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where $f(x_i) = y_i$. In the example above, a pair would correspond to an image of a pedestrian along with a bounding box. The goal of the machine learning algorithm is to approximate f as close as possible, meaning the prediction $\hat{f}(x) = \hat{y}$ should be as close to the true value $f(x) = y$ as possible.

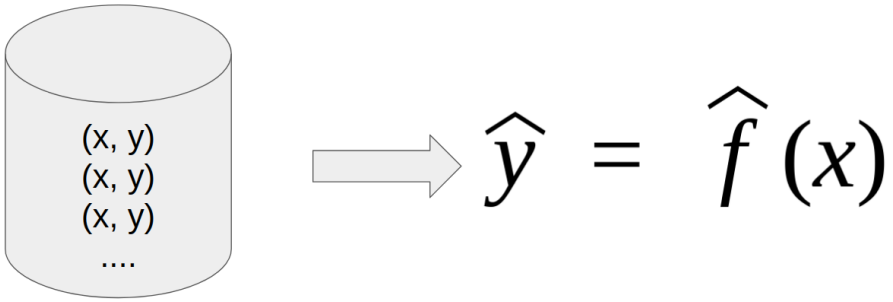

Figure 2.1: An illustration of supervised learning.

Given L, a DNN approximates f by iteratively improving its prediction through a process called training. In each iteration, the parameter weights of the network are tuned to better approximate f. In order to update the weights, a measure of how far the prediction is from the true value is needed. That is usually calculated by applying a loss function to the ground truth and the prediction.

2.2 Semi-Supervised Learning

In semi-supervised learning, a set of unlabeled data $U = \{x_1, x_2, ..., x_m\}$ is used together with the set of labeled data L [14]. Since unlabeled data often is cheap and easy to obtain, it is common that $|U| \gg |L|$. The goal is the same as in supervised learning, to approximate the function f from X to Y. However, during training, the algorithm uses both U and L to learn.

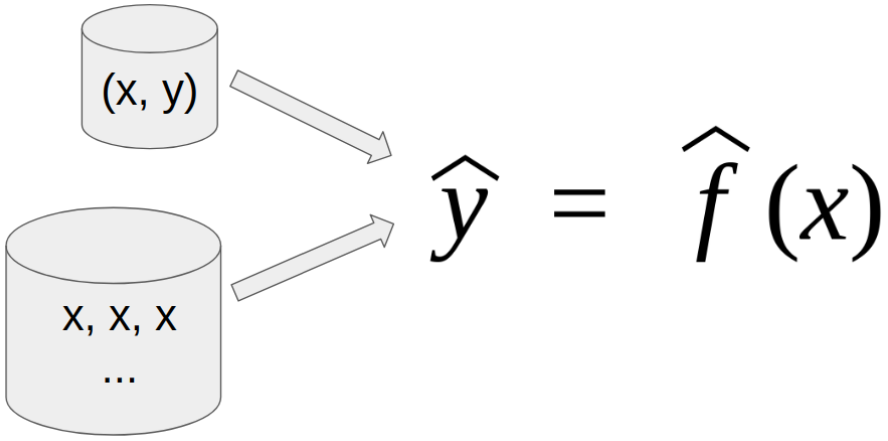

Figure 2.2: An illustration of semi-supervised learning.

In the context of DNNs, given an input x and a prediction $\hat{f}(x)$, one must define a loss function for the network to be able to learn from the sample. A commonly used loss function for supervised learning is the cross-entropy loss, but it is clear that cross-entropy can not be used for unlabeled samples as they have no ground truth label. In the subsections that follow, we explain two paradigms within semi-supervised learning, which deal with this in different ways, called self-supervision and consistency training.

2.2.1 Self-Supervision

In self-supervision, the model predictions are used to create labels for the unlabeled samples. In this thesis, we will refer to such labels as pseudo-annotations. The most straightforward way is to add the samples and pseudo-annotations to L, and treat them as ground-truth, although other approaches exist. Obviously, this comes with a risk of incorrect labels, and a key challenge in self-supervision is to ensure that the labels are correct.

An example of self-supervision is the co-training algorithm [3]. In the original co-training algorithm, two views of the same object are used to train two different learners. The two learners are used to predict the labels of unlabeled examples. The most confident predictions are then added to L, and the process is repeated. In this way, the two learners are able to learn from each other.

2.2.2 Consistency Training

In consistency training, the goal is to be accurate on the labeled samples while being consistent on the unlabeled samples. Consistency can be measured in a number of ways. One way to check for consistency is to apply data augmentation techniques to an input and then check that the outputs of all augmented inputs are the same. For example, in the case of image classification, one could apply transformations such as rotation on an input image, classify the augmented images, and check that the predicted classes of all inputs are the same. Note that it is not necessary to know the ground truth label of the image to check for consistency, it suffices to check that all outputs are equal.

An example of consistency training is *Regularization with stochastic transformations and perturbations for deep semi-supervised learning* [15] that introduces an unsupervised loss function, that penalizes the network when given inconsistent results for the same input, with different augmentations.

2.2.3 Student Teacher Framework

Another method of training, which has shown success within semi-supervised learning, is the student-teacher framework [12, 19, 22]. In this framework, a teacher generates targets that a student uses to train. Different implementations use different teachers and apply the targets differently when training the student. In *Mean Teacher* [19], the teacher is created by calculating the mean network weights over the last n student models. By doing this they hope to get a more robust teacher that produces more accurate targets. *Temporal Ensembling* [12] highly resembles Mean Teacher, but rather than calculating the mean network weights to create a teacher, they use the average predictions from the last n student models to directly produce the targets.

A more recent approach was presented in the *Noisy Student* [22] paper. They suggest applying noise, such as dropout, stochastic depth, and augmentations, when training the student to make it generalize better. In their experiments, they iterate the training by making the student the new teacher and retraining a new student model from scratch with the new teacher's targets. This was iterated 3 times, where the initial teacher was trained on a pre-existing labeled dataset, and showed an increase in performance, although diminishing, for each iteration.

2.3 Multiple Object Tracking

A semi-supervised method which considers time dependencies should have a larger potential, since sequential images are likely to be similar. In this thesis, we propose such a method, which relies on Multiple Object Tracking (MOT). Therefore, this section provides a brief introduction to the foundations of MOT.

2.3.1 Introduction and Intuition

Let us introduce multiple object tracking by formulating the problem we are trying to solve. Multiple object tracking consists of tracking an unknown, varying, number of objects across time. This is done given a number of detections, also called measurements, in each time step, as visualized in Figure 2.3. In the context of this thesis, we are interested in performing MOT on objects in a video. Hence, the sensor in this case is a camera. Furthermore, as the frames are not annotated, a DNN will be used as detector. Thus the input to the MOT algorithm will be a number of object detections from the DNN at each point in time. The output of the MOT algorithm is an estimate of the state of the tracked objects.

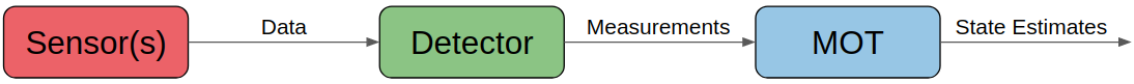

Figure 2.3: An illustration of MOT [18].

From the description above, it is clear that MOT is not an easy task, and that there are several challenges. To motivate the remaining contents of this chapter, let us attempt to identify the key challenges of MOT. First of all, consider the input to the algorithm. In most interesting settings, the detector is not perfect. Therefore an MOT algorithm must be able to deal with three things.

1. The detector might provide false detections, meaning detections from other things than objects.
2. The detector might not provide a detection for an object even though it is present. Hence the algorithm must consider the possibility of misdetection.
3. The provided detections might not be perfect. Therefore, the MOT algorithm must be able to deal with noisy measurements.

Furthermore, in MOT we assume that the detector cannot provide multiple detections for the same object. That is, we assume that every object yields at most one detection.

Above we have described what aspects of the detector that the MOT algorithm must consider. However, even if one would assume a perfect detector, there are several difficult challenges remaining. First of all, the number of objects is unknown and varying. This means that at each point in time, previously present objects can disappear, and new objects appear. An MOT algorithm must consider all of these possibilities and somehow identify the most likely one. Furthermore, even if the

detector is perfect, and even if the number of objects would be known and constant, a huge challenge remains. Assume n tracked objects and n true, perfect, detections. How should we associate the detections to the objects? There are clearly $n!$ possibilities, and it is often intractable to compute all of them. Therefore, one could argue that the association problem is the biggest challenge in MOT.

Above we have given a brief introduction to the main challenges of MOT. In the rest of this chapter, we will learn how to deal with them. We begin by formalizing the problem and move on to show how we can use Bayesian statistics, and in some cases, the Kalman filter to solve it. We proceed to show how to use that to form a complete MOT algorithm.

2.3.2 Problem formulation

In this section, we present a formal definition of the problem. As explained in the previous section, the algorithm outputs state estimates of the tracked objects. In what follows we will begin by considering a single object and generalize to multiple objects later. Denote the state of the object at time k by x_k. We seek x_k for all $k = 1...n$.

Note that the true state is hidden. We seek to estimate the states $\{x_0, ... \ x_n\}$ using noisy measurements of the object denoted $\{y_0, ..., y_n\}$, where y_k is the measurement of the object at time k. Calculating $\{x_0, ... \ x_n\}$ using $\{y_0, ..., y_n\}$ is a statistical inversion problem [16, Section 1.3]. For the sake of brevity, in what follows, we will use $y_{1:k}$ to denote $y_1, y_2, ... \ y_k$.

In this thesis we take a Bayesian approach, and in that sense, the statistical inversion problem consists of computing $p(x_{0:k}|y_{1:k})$. The problem of estimating the state at a particular time k using the measurements up until time k, that is estimating $p(x_k|y_{1:k})$, is called **filtering** [16, p. 11].

An alternative to filtering is **smoothing**, which is concerned with estimating the state at time k using measurements at a time later than k. That is, using $y_1, ... \ y_k$, $... \ y_t$ where $t > k$. Thus, one seeks to estimate $p(x_k|y_{1:t})$ [16, p. 11]. Within the scope of this thesis, the tracking will be done offline with all detections available at all times, and therefore we can perform smoothing. In the sections that follow, we will describe how these densities can be estimated, and the tools needed to do so.

2.3.3 Motion Model

In the previous section we formalized the problem, and concluded that we seek $p(x_k|y_{1:n})$ for all $k = 1...n$. It makes intuitive sense that the state x_k should somehow depend on state x_{k-1}. To incorporate that into our estimation, we need a model that describes how the object state evolves over time. This is called a motion model.

To motivate the use of a motion model, consider the two-dimensional example in Figure 2.4. One way to estimate the position, x_k, would be to use $x_k = y_k$. That would result in the blue trajectory, and would explain the data perfectly. However, cars usually don't move like that, and it seems more likely that the detections are noisy. Instead, by considering the velocity, one can form the green trajectory, which seems more probable. In this case, that aligns with the true position of the object. By having a motion model, we can incorporate information such as the relation between position and velocity in our estimation of x_k.

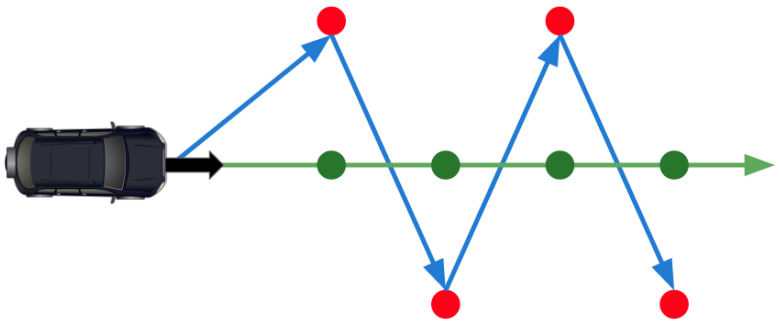

Figure 2.4: An example of an object with a velocity vector (black arrow). The green dots show the objects true position across time, and the red dots show the detected position across time. The coloured arrows show two alternative trajectories.

The purpose of the motion model is to describe how x_{k-1} transitions to x_k, denoted $p(x_k|x_{k-1})$. We assume that the object state evolves according to a Markov chain, meaning state x_k only depends on state x_{k-1} [16, p. 51].

As mentioned, the motion model describes how the state evolves across time, and this could be done using the equation

$$x_k = f_{k-1}(x_{k-1}, q_{k-1}) \tag{2.1}$$

where f is a function that describes how the state evolves, x is the state and q_{k-1} is Gaussian noise [16, p. 46]. The purpose of incorporating noise is to account for the fact that the state evolves stochastically.

It is common to use linear models with additive Gaussian noise. This could be described using the equation

$$x_k = F_{k-1} \cdot x_{k-1} + q_{k-1} \tag{2.2}$$

where F_{k-1} is a matrix describing how the state evolves at time $k-1$ [16, p. 45]. Furthermore, it is common to assume Gaussian noise with zero mean. That is, $q_{k-1} \sim \mathcal{N}(0, Q_{k-1})$ where Q_{k-1} describes the noise covariance. Using that assumption along with a linear model, we get

$$p(x_k|x_{k-1}) = \mathcal{N}(x_k; F_{k-1} \cdot x_{k-1}, Q_{k-1}) \tag{2.3}$$

where x_k is the variable, $F_{k-1} \cdot x_{k-1}$ the mean and Q_{k-1} the covariance [16, p. 56].

2.3.4 Measurement Model

As mentioned in the introduction to this section, the detector is usually not perfect. An example of this is shown in Figure 2.4, where the detections are clearly different from the true state. Furthermore, not all elements of the state are directly observable. For example, we might be interested in estimating velocity from positions. To account for this, we use a measurement model.
In particular, the purpose of the measurement model is to describe the relation between the true state and the measurement, $p(y_k|x_k)$ [16, p. 10]. This can be done using the equation

$$y_k = h_k(x_k, r_k) \tag{2.4}$$

where h describes the relation to the state x_k and measurement noise r_k [16, p. 46].

It is common to use linear models with additive Gaussian noise. This could be described using the equation

$$y_k = H_k \cdot x_k + r_k \tag{2.5}$$

where H_k is a matrix describing how the measurement relates to the state [16, p. 45].

It is common to assume Gaussian measurement noise with zero mean. That is, $r_k \sim \mathcal{N}(0, R_k)$ where R_k describes the noise covariance. Under this assumption, we get that

$$p(y_k|x_k) = \mathcal{N}(y_k; H_k \cdot x_k, R_k) \tag{2.6}$$

where y_k is the variable [16, p. 56].

Furthermore, the measurement y_k is conditionally independent from previous states and measurements given x_k, which is expressed as

$$p(y_k|x_{0:k}, y_{0:k-1}) = p(y_k|x_k) \tag{2.7}$$

mathematically. Simply put, given the x_k, y_k is independent from all previous states and measurements [16, p. 52].

2.3.5 Bayesian Statistics

In the previous sections, we concluded that we seek $p(x_k|y_{1:n})$. We also concluded that we would like to use a motion model to incorporate prior knowledge of how the state evolves over time, and that we would like to use a measurement model to account for the fact that measurements can be noisy. However, it is not clear how one calculates this density given the measurements. For this purpose, we will make use of Bayesian statistics.

As mentioned, we seek $p(x_k|y_{1:n})$. It is clear that applying the law of total probability to the density $p(x_k|y_{1:n})$ according to

$$p(x_k|y_{1:k}) = \int p(x_{0:k}|y_{1:k})dx_{0:k-1} \tag{2.8}$$

results in the sought density. From that we must find $p(x_{0:k}|y_{1:k})$. We begin by applying Bayes' rule

$$p(x_{0:k}|y_{1:k}) = \frac{p(y_{1:k}|x_{0:k}) \cdot p(x_{0:k})}{p(y_{1:k})} \tag{2.9}$$

to get a simplified expression [16, p. 9]. Then, by applying the law of conditional probability, while also using that y_i is conditionally independent on x_i, and that x_i is conditionally independent on x_{i-1}, we get

$$\frac{p(y_{1:k}|x_{0:k}) \cdot p(x_{0:k})}{p(y_{1:k})} = p(x_0) \prod_{i=1}^{k} p(y_i|x_i) \cdot p(x_i|x_{i-1}) \tag{2.10}$$

which gives us a solution to our problem [16, p. 53].

However, the computational complexity grows as k grows, making this solution unfeasible for a large k [16, p. 53]. Instead, we would like a more computationally efficient solution. Recall that we are interested in calculating the posterior density for all time steps. An idea that comes to mind is to reuse the posterior $p(x_{k-1}|y_{1:k-1})$ to calculate the posterior $p(x_k|y_{1:k})$. It turns out that we can compute the posterior for time k recursively from the posterior at time $k-1$ by splitting the calculation into a prediction and update step [16, p. 54]. This is visualized in Figure 2.5.

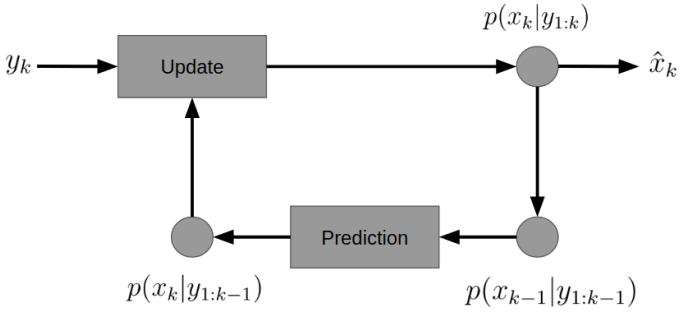

Figure 2.5: An illustration of the prediction and update step.

The intuition is that we should be able to predict the state at time k from the posterior at time $k-1$. Once a measurement is received, we should be able to correct our prediction based on that measurement to get the posterior. Using this approach, the number of calculations does not increase with k.

The prediction step consists of computing $p(x_k|y_{1:k-1})$ from $p(x_{k-1}|y_{1:k-1})$. To do this, we apply the Chapman-Kolmogorov equation [16, p. 54]

$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1}) \cdot p(x_{k-1}|y_{1:k-1}).dx_{k-1} \tag{2.11}$$

which is derived in appendix C. The second factor in the resulting integral is the posterior from time $k-1$, while the first factor corresponds to the motion model. Hence, we should be able to calculate this integral.

The update step consists of computing $p(x_k|y_{1:k})$ from y_k and the prediction $p(x_k|y_{1:k-1})$. We can do this by applying Bayes' rule according to

$$p(x_k|y_{1:k}) = \frac{1}{Z_k} \cdot p(y_k|x_k) \cdot p(x_k|y_{1:k-1}) \tag{2.12}$$

where the normalization constant Z_k is given by

$$Z_k = \int p(y_k|x_k) \cdot p(x_k|y_{1:k-1}) dx_k \tag{2.13}$$

as shown in [16, p. 54]. It is worthwhile to note that the prediction and update equations provide a general, recursive, solution to any filtering problem. Still, calculating the resulting densities could be hard. However, by restricting the problem one can obtain a closed-form solution, as shown in the next section.

2.3.6 The Kalman Filter

In the previous sections, we showed that splitting the calculation into a prediction and update step resulted in a computationally efficient expression for the posterior. Unfortunately, using the expression to solve for the posterior in the general case is not trivial. However, it turns out that restricting the problem to linear models that assume Gaussian noise with zero mean results in a simplified problem. In fact, there is a closed-form solution, which is given by the Kalman filter [16, p. 56].

Deriving the Kalman filter is out of scope for this thesis. However, since the algorithms used are highly dependent on the Kalman filter, the relevant equations are presented. Since the Kalman filter assumes linear models and Gaussian noise with zero mean, equations 2.3 and 2.6 describe the motion and measurement model.

Let $\bar{x}_{k|k}$ denote the mean and $P_{k|k}$ the covariance of the posterior object state density at time k. The prediction step is given by

$$\bar{x}_{k|k-1} = F_{k-1} \cdot x_{k-1|k-1} \tag{2.14}$$

$$P_{k|k-1} = F_{k-1} \cdot P_{k-1|k-1} \cdot F_{k_1}^T + Q_{k-1} \tag{2.15}$$

where $\bar{x}_{k|k-1}$ is the predicted mean and $P_{k|k-1}$ the predicted covariance [16, p. 57]. The intuition behind these equations is simple - the motion model is used to predict the mean and covariance of the next state.

The update step involves a bit more detail. Based on the predicted state, a predicted measurement $\bar{z}_k$ is calculated, which is used with the observed measurement z_k to calculate the innovation ϵ_k

$$\bar{z_k} = H_k \cdot \bar{x}_{k|k-1} \tag{2.16}$$

$$\epsilon_k = z_k - \bar{z_k}. \tag{2.17}$$

One way to interpret the innovation is that it represents how much information the measurement contains. The innovation is used to update the mean, but to do that we also need the Kalman gain K_k. Essentially, the Kalman gain is a measure of how much the new measurement should be trusted and is calculated according to

$$S_k = H_k \cdot P_{k|k-1} \cdot H_k^T + R_k \tag{2.18}$$

$$K_k = P_{k|k-1} \cdot H_k^T \cdot S_k^{-1}. \tag{2.19}$$

As can be seen in equation 2.19, K_k is based on S_k, which is called the innovation covariance. Finally, the updated mean and covariance are given by

$$\bar{x}_{k|k} = \bar{x}_{k|k-1} + K_k \cdot \epsilon_k \tag{2.20}$$

$$P_{k|k} = P_{k|k-1} - K_k \cdot H_k \cdot P_{k|k-1} \tag{2.21}$$

where $\bar{x}_{k|k}$ is the updated mean and $P_{k|k}$ is the updated covariance [16, p. 57].

2.3.7 Random Finite Sets

In the previous sections, we formalized the problem and concluded that in the case of linear and Gaussian models, we can use the Kalman filter to get an optimal solution to the filtering problem. However, we have assumed a single measurement, while in practice the algorithm must deal with multiple detections that might not include detections from the object or might be false. Similarly, we have assumed that a single object is tracked, while in practice we want to track multiple objects. In this section, we introduce the concept of Random Finite Sets (RFS) to tackle this.

An RFS is a random variable. The possible values are (unordered) finite sets. Both the cardinality (number of elements), as well as the elements themselves, are random [20, p. 20]. Assuming the elements of the set are vectors $x \in R^n$, the RFS of such elements $\mathbb{X}$ belongs to the space $\mathcal{F}(R^n)$. That is, $\mathbb{X} \in \mathcal{F}(R^n)$, where $\mathcal{F}(R^n)$ is the space of all finite subsets of R^n [9, p. 1884].

To describe the distribution of values for the RFS, one can use a probability density function (PDF). The PDF captures the distribution of the cardinality of the set, as well as the distribution over the elements [20, p. 20-21].

The Bernoulli RFS is a simple example of an RFS and is reminiscent of a Bernoulli random variable. The PDF for a Bernoulli RFS $\mathbb{X}$ with elements $x \in \mathcal{X}$ is given by

$$p(\mathbb{X}) = \begin{cases} 1 - r & \text{if } \mathbb{X} = \emptyset \\ r \cdot f(x) & \text{if } \mathbb{X} = \{x\} \\ 0 & \text{else} \end{cases} \tag{2.22}$$

where r is the existence probability and $f(x)$ a PDF describing the distribution of values for the random element x [9, eq. 10].

A Bernoulli RFS could be used to model e.g a single object and the uncertainty of its existence (r) and state ($f(x)$). However, we are interested in tracking multiple objects. This could be modeled using the union of several independent Bernoulli RFSs. Such a set is called a Multi-Bernoulli RFS.

Assume that $\mathbb{X}_1$, ..., $\mathbb{X}_n$ are n independent Bernoulli RFSs with densities p_1, ..., p_n. Then $\mathbb{X}$ as given by

$$\mathbb{X} = \bigcup_{i=1}^{n} \mathbb{X}_i \tag{2.23}$$

is a Multi-Bernoulli RFS [20, p. 29]. The PDF is given by

$$p(\mathbb{X}) = \sum_{\mathbb{X}_1 \uplus \ldots \uplus \mathbb{X}_n = \mathbb{X}} \prod_{i=1}^{n} p_i(\mathbb{X}_i) \tag{2.24}$$

where $\uplus$ is the union of mutually disjoint sets [9, eq. 11].

Another important RFS is the Poisson RFS, also known as the Poisson Point Process (PPP). Later in this chapter, PPPs will be used to model false detections and the appearance of objects. A PPP $\mathbb{X}$ with elements $x \in \mathcal{X}$ is characterized by an intensity function $\lambda(x) \geq 0$ [20, p. 28].

To provide an intuition for the intensity function, assume that a PPP is used to model false detections. Then we would use a high intensity in the parts of the space where we expect several false detections and a low intensity in the rest of the space.

The cardinality of the set is Poisson distributed with mean $\bar{\lambda}$, which is defined as

$$\bar{\lambda} = \int \lambda(x) dx. \tag{2.25}$$

That is, $|\mathbb{X}| \sim Po(\bar{\lambda})$ [20, p. 28]. Lastly, the PDF of a PPP is given by

$$p(\mathbb{X}) = e^{-\bar{\lambda}} \cdot \prod_{i=1}^{n} \lambda(x_i) \tag{2.26}$$

where $\mathbb{X} = \{x_1, \ldots\ x_n\}$ [9, eq. 8].

2.3.8 Dealing with Missed Detections

As mentioned previously, an MOT algorithm must be able to handle a varying number of objects. Furthermore, it must also be able to handle that the detector might not detect an object, even though it is present. To tackle this we will use the Bernoulli RFS introduced in the previous section.

The existence probability is in this case the probability of detecting the object given that it is present. For this purpose, it is common to use a function denoted $P^D(x_k)$ which, given an object state x_k, outputs the probability of detecting that object. The probability of not detecting the object is $1 - P^D(x_k)$.

Each object is clearly either detected or misdetected, and hence it makes sense to use a Bernoulli RFS to model detections of objects. Let $\mathbb{O}_k^j$ be a Bernoulli RFS which holds the detection of object j with state x_k^j at time k. That is, it potentially holds an element o_k^j from the space of detections. Conditioning on the state, and plugging $P^D(x_k)$ into equation 2.22, yields the density

$$g_k^j(\mathbb{O}_k^j|\{x_k^j\}) = \begin{cases} 1 - P^D(x_k^j) & \text{if } \mathbb{O}_k^j = \emptyset \\ P^D(x_k^j) \cdot f(o_k^j|x_k^j) & \text{if } \mathbb{O}_k^j = \{o_k^j\} \\ 0 & \text{else} \end{cases} \tag{2.27}$$

where $f(o_k^j|x_k^j)$ is the measurement density [9, eq. 8]. Assuming a linear measurement model with zero-mean Gaussian noise according to equation 2.6 yields the density

$$g_k^j(\mathbb{O}_k^j|\{x_k^j\}) = \begin{cases} 1 - P^D(x_k^j) & \text{if } \mathbb{O}_k^j = \emptyset \\ P^D(x_k^j) \cdot \mathcal{N}(o_k^j; H_k \cdot x_k^j, R_k) & \text{if } \mathbb{O}_k^j = \{o_k^j\} \\ 0 & \text{else} \end{cases} \tag{2.28}$$

The set of all object detections at time k, assuming n_k objects, denoted O_k, is a Multi Bernoulli RFS given by

$$\mathbb{O}_k = \bigcup_{j=1}^{n_k} \mathbb{O}_k^j. \tag{2.29}$$

Using equation 2.24 and the density for $\mathbb{O}_k^j$ from equation 2.27 we get the density

$$g_k(\mathbb{O}_k|\{x_k^1, ..., x_k^{n_k}\}) = \sum_{\mathbb{O}_k^1 \uplus ... \uplus \mathbb{O}_k^n = \mathbb{O}_k} \prod_{i=1}^{n} g_k^i(\mathbb{O}_k^i|\{x_k^i\}). \tag{2.30}$$

2.3.9 Dealing With False Detections

In the previous section, we introduced a Multi Bernoulli RFS to deal with missed detections. In addition, an MOT algorithm must be able to handle false detections, i.e detections from other things than objects. To do this, we will use a PPP.

Let the PPP $\mathbb{C}_k$ be the set of false detections at time k. From Section 2.3.7, we know that the number of false detections $|\mathbb{C}_k| = m_k^c$ is Poisson distributed according to $m_k^c \sim Po(\bar{\lambda_c})$ where $\bar{\lambda_c}$ is the mean cardinality of the set. That is, $\bar{\lambda_c}$ is the expected number of false detections.

Furthermore, as mentioned in Section 2.3.7, a PPP is characterized by its intensity function. We have already denoted the expected number of false detections by $\bar{\lambda}_c$, and we know that it must abide by equation 2.25. One way to define the intensity λ_c is thus

$$\lambda_c(C_k^j) = \bar{\lambda}_c \cdot f_c(C_k^j) \tag{2.31}$$

where $f_c(.)$ is a spatial PDF and C_k^j a false detection. Clearly this abides to 2.25 since $f_c(.)$ integrates to one. The density $p(\mathbb{C}_k)$ is given by equation 2.26

2.3.10 A Complete Measurement Model

In the two previous sections, we introduced two RFS - the set of observations, $\mathbb{O}_k$, and the set of false detections $\mathbb{C}_k$. Obviously, given a detection, the algorithm will not know if it belongs to $\mathbb{O}_k$ or $\mathbb{C}_k$. Therefore, we introduce $\mathbb{Z}_k$ given by

$$\mathbb{Z}_k = \mathbb{O}_k \cup \mathbb{C}_k \tag{2.32}$$

as the set of all detections. Since all the Bernoullis in $\mathbb{O}_k$ are independent this can also be written as [9, p. 1884]

$$\mathbb{Z}_k = \mathbb{C}_k \cup \mathbb{O}_k^1 \cup ... \cup \mathbb{O}_k^{n_k}. \tag{2.33}$$

Given the n_k objects, the corresponding density is [9, eq. 5]

$$p(\mathbb{Z}_k|\{x_k^1, ..., x_k^{n_k}\}) = e^{-\bar{\lambda_c}} \cdot \sum_{\mathbb{C} \uplus \mathbb{O}_k^1 \uplus ... \uplus \mathbb{O}_k^{n_k} = \mathbb{Z}_k} \prod_{C \in \mathbb{C}} \lambda_c(C) \cdot \prod_{i=1}^{n_k} g_k^i(\mathbb{O}_k^i|\{x_k^i\}). \tag{2.34}$$

2.3.11 Modelling Appearing and Disappearing Objects

In the two previous sections, we described a full measurement model based on RFSs which was able to handle both missed detections of objects as well as false detections. In this section, we will continue to use the concept of RFSs to introduce a complete motion model, which is able to handle the appearance and disappearance of objects.

In particular, we define objects that are within the area of interest as true objects, and denote them by $\mathbb{X}_k$. In each time, step objects can both enter and leave the area of interest. These events are referred to as object birth and death. When an object is born, the algorithm should start to track it, and when an object dies, the tracking should stop.

The set $\mathbb{X}_k$ can be thought of as the union of two subsets

$$\mathbb{X}_k = \mathbb{X}_k^s \cup \mathbb{X}_k^b \tag{2.35}$$

where $\mathbb{X}_k^s$ holds the surviving objects from the previous time step and $\mathbb{X}_k^b$ holds the objects born in the current time step [23, p. 2]. In what follows, we present models for object death and object birth.

An object is considered dead at the point in time when it leaves the area of interest. We model this using a survival probability as a function of the object state, denoted $P^S(x_k^i)$. Objects in $\mathbb{X}_k$ transition to $\mathbb{X}_{k+1}^s$ with probability $P^S(x_k^i)$ [23, p. 2].

It is common to let $P^S(x_k^i) = 0$ for objects outside of the area of interest to let $P^S(x_k^i)$ be high for objects inside the area of interest.

An object is born at the point in time when it enters the sensor's field of view. We model the birth as a PPP, represented by an intensity function $\lambda_b(.)$ [23, p. 2]. The intensity function could be represented as

$$\lambda_b(x_k^j) = \bar{\lambda}_b \cdot f_b(x_k^j) \tag{2.36}$$

where $f_b(.)$ is a spatial PDF describing where in the space objects are likely to be born, while the Poisson rate $\bar{\lambda}_b$ describes the expected number of objects born at each time.

2.3.12 Sets of Trajectories

Up until now, we have assumed that we want to estimate the state of each object, denoted x_k^j. However, one could argue that at time k, we are not only interested in x_k^j, but in fact x_b^j, ... x_k^j where b is the time of birth of the object. That is, we are interested in the sequence of all object states, as opposed to only the current state.

For this purpose, we will instead represent each object with a trajectory, holding the sequence of states of a single object through time. We base our definition on the work in [10]. A trajectory at time k can be defined as

$$X_k = (\beta, \epsilon, x^{\beta:\epsilon}) \tag{2.37}$$

where β is the time of birth, ϵ the time of the most recent state, and $x^{\beta:\epsilon}$ is the state vector for the object from time β to ϵ. If $\epsilon = k$ the trajectory is still alive, while if $\epsilon < k$ the trajectory is dead and ϵ then represents the time of death.

2.3.13 Trajectory Poisson Multi Bernoulli

In this section, we combine all the building blocks presented this far to present a complete MOT algorithm. In particular, we introduce the Trajectory Poisson Multi Bernoulli (TPMB) [23] filter, a simple yet powerful MOT algorithm. We begin by giving an introduction to the basic idea and purpose of TPMB. Then, we describe

the steps of the TPMB algorithm in further detail. However, for the exact details, we refer the interested reader to the original TPMB paper.

The basic idea behind TPMB is to estimate a Poisson Multi Bernoulli Mixture (PMBM) density using a Poisson Multi Bernoulli density (PMB). A PMB is an RFS, which is the union of a PPP and a Multi Bernoulli (MB) RFS, while a PMBM is a mixture of such RFSs. Furthermore, objects are represented using trajectories. The set of trajectories at time k, denoted $\mathbb{X}_k$, is given by

$$\mathbb{X}_k = \mathbb{X}_k^d \cup \mathbb{X}_k^u \tag{2.38}$$

where $\mathbb{X}_k^u$ corresponds to a PPP, representing undetected objects, while $\mathbb{X}_k^d$ corresponds to a MB RFS representing detected objects [23, eq. 3]. The purpose of $\mathbb{X}_k^u$ is to model, for example, occluded objects. These are objects which have entered the area of interest without being detected.

At a high-level, the algorithm can be described using the pseudo-code below. The input is a list consisting of a number of measurements at each time step. The algorithm then iterates over all time steps. At each iteration, it uses a motion model and the posterior PMB density from the last iteration to predict the next state for all trajectories. Then, using the measurements and the measurement model, it performs an update. In the update step each measurement is associated to either an existing object, a new object or a false detection. We create and calculate the probability for the N most likely global hypotheses Murty's algorithm, resulting in a PMBM density. In order to make the PMBM into a PMB, the mixture is projected into a single PMB density.

```
function tpmb(measurements: [Measurement]) -> PMB:
    density = PMB()

    for t = [1...T]:
        density = update(density, measurements[t])
        density = projection(density)
        density = predict(density)

    return density

function update(density: PMB, measure: Measurement) -> PMBM:
    ...

function projection(density: PMBM) -> PMB:
    ...

function predict(density: PMB) -> PMB:
    ...
```

Within the scope of this thesis, we will restrict ourselves to linear models assuming Gaussian noise with zero mean. This allows us to perform parts of the prediction and update step using the Kalman filter equations described in Section 2.3.6. Assuming such models, the full prediction and update steps of the algorithm are given in lemmas eight and nine in [23]. The projection step is given by proposition 2 in [23].

Since the algorithm estimates trajectories, the covariance matrix describes the covariance across all states in the trajectory. The algorithm smooths the trajectories as a part of the update step using this covariance matrix. Since the matrix grows quadratically, L-scan, which only considers the joint covariance of the last L time steps, is used. States prior to the last L time steps are considered independent [23, p. 8].

2.4 Average Precision

Precision is an evaluation metric that is often used in object detection. Precision describes how often the predicted objects are objects of the predicted class. Formally, precision can be defined as

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}. \tag{2.39}$$

Together with precision it is common to use recall to measure performance. Recall describes what portion of the objects are found. Recall can be defined as

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}. \tag{2.40}$$

The precision depends on the recall and the recall depends on the precision. It is therefore common to calculate the Average Precision Score (APS), that describes the average precision over all values of recall. It is calculated as follows

$$\text{APS} = \int_0^1 p(r)dr \tag{2.41}$$

where p is the precision given a specific recall. Practically, it can be difficult to calculate and it is instead often calculated as a sum over a ranked sequence. A common practical method for calculating the APS is described for *The Pascal Visual Object Classes (VOC) Challenge* [8].

3
Methods

In this thesis, two methods for training object detection models in a semi-supervised fashion are developed and evaluated. In this chapter, we introduce the two methods. We begin with Noisy Student for Object Detection. This is an adaption of an existing semi-supervised method to object detection which assumes that all images are independent. We then move on to the second method, Tracking Objects For Self-supervision (TOFS), which leverages the time dependency across frames using a Multiple Object Tracking (MOT) algorithm.

3.1 Noisy Student for Object Detection

Noisy Student for Object Detection builds on the teacher-student framework introduced in Section 2.2.3, and is heavily inspired by the method in *Self-training with Noisy Student improves ImageNet classification* [22]. Just like in the original method, an initial teacher model f_t is trained on an annotated dataset L. The teacher model is then used to infer pseudo-annotations, on an unannotated dataset U. Using both L and U, a student model f_s is trained. During the training, noise and augmentations are applied, as detailed by Table 3.1. Note that no noise is applied during the inference. This is to make sure that we infer the most accurate pseudo-annotations. Once the training is complete, the student becomes the new teacher, and the process can be repeated.

Noise	**Description**
HSV	Hue saturation and value augmentation
Scale	Random scale of image
Crop	Random crop of image
PIL	Augment the image brightness, color, contrast and sharpness

Table 3.1: Augmentations and noise applied during training.

However, this method differs from the work in [22] in a number of ways. Firstly, the method is applied to the task of object detection rather than image classification. Thus, instead of creating a single pseudo-label for each image, the algorithm will create a set of pseudo-annotations.

Secondly, a hard label is used for the class of each pseudo-annotation, while the original method was evaluated with both hard and soft-labels. This means that the class of each pseudo-annotation will be the argmax of the model's class prediction.

Thirdly, inspired by FixMatch [17], a confidence threshold $0 \leq p_t \leq 1$ is used for the class. In particular, assuming that the predicted class is represented by a vector s.t the sum of its elements is one, and p is the probability for the most likely class, then objects s.t $1 - p_t \leq p \leq p_t$ are marked as 'unclear', as visualized in Figure 3.1. Areas marked as 'unclear' will not contribute to any loss during the training, regardless of the model's prediction. Since the model will predict incorrect classes occasionally, it is important to only create pseudo-annotations from confident predictions. Similarly to the results in FixMatch [17], a threshold of $p_t = 0.95$ produced the best results. Finally, the algorithm can be summarized as follows.

```
U = Unannotated dataset
L = Annotated dataset
Train teacher model ft on L

for i in [1...n]:
    Infer pseudo-annotations on U using ft
    Train new student model fs with noise on U & L
    Make student the new teacher, ft = fs
```

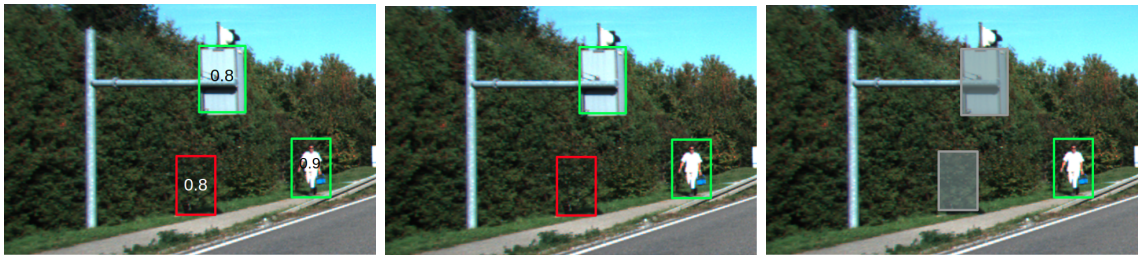

Figure 3.1: An illustration of unclear pseudo-annotations. The images represent detections and pseudo-annotations with and without unclear regions. Whether a pseudo-annotation is marked as unclear is based on the detection's confidence.

3.2 Tracking Objects For Self-supervision

In the previous section, we described a novel method for training object detection networks in a semi-supervised manner. The method treated all images independently. However, within Autonomous Driving (AD), and several other domains, data is captured as video, meaning the input images are consecutive frames with a dependency across time. In this section, we propose a method that aims to exploit this dependency: Tracking Objects For Self-supervision (TOFS).

The proposed method uses Multiple Object Tracking (MOT) to account for time dependencies. The output of the MOT algorithm is a set of smoothed object trajectories, which are used to create pseudo-annotations. The algorithm is visualized in Figure 3.2 and at a high-level it works as follows.

1. Train an object detection network on annotated data.
2. Infer detections on unannotated sequences of frames.
3. Apply a MOT algorithm to create smoothed object trajectories.
4. Extract pseudo-annotations from the trajectories.
5. Use the pseudo-annotations to re-train the network from scratch.

In the rest of this section, the method is described in further detail. First, an intuition for the types of improvements the method should lead to is given. Then the MOT algorithm and its parameterization are described. Lastly, the algorithm for extracting pseudo-annotations from the output of the MOT algorithm is given.

3.2.1 Expected Improvements

There are two things the method is intended to improve. Firstly, recall that the output of the algorithm is a set of smoothed object trajectories. In other words, each object trajectory is based on multiple observations across time, accounts for measurement noise, and is smoothed according to a motion model. The idea is that by taking these factors into account, object properties such as position, dimension and rotation should be closer to ground truth than the sequence of raw detections.

Secondly, the MOT algorithm can account for missed detections. That is, if an object is not detected by the network even though it is present, the MOT algorithm will still output an estimate of the object state. Because of that, pseudo-annotations can be created for objects which are not detected, which should lead to a higher recall for the network.

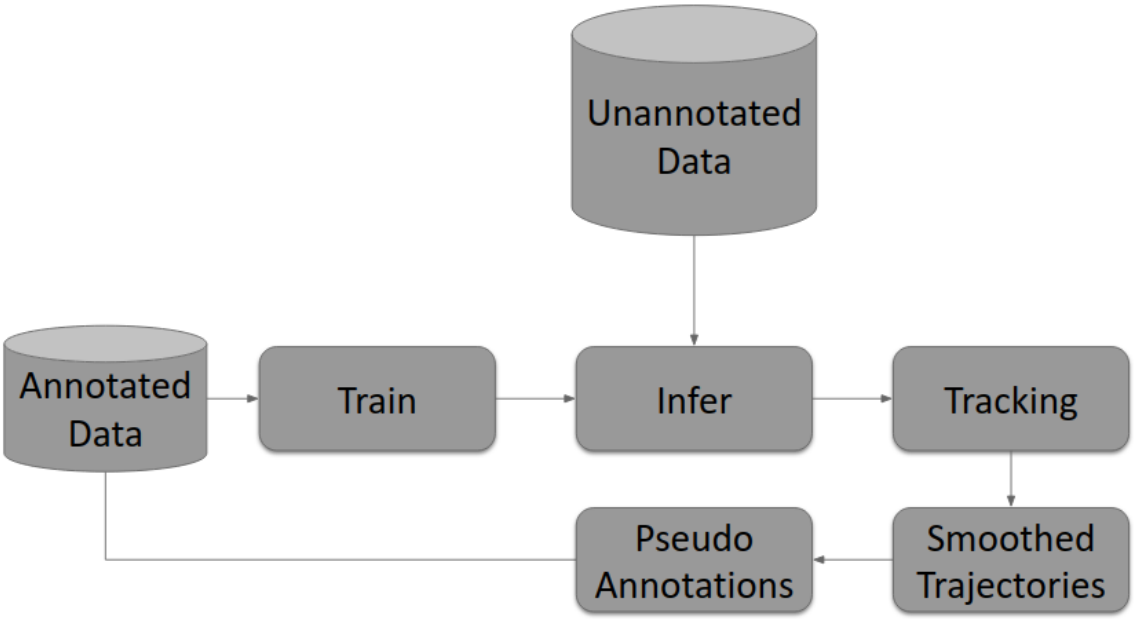

Figure 3.2: An overview of the method. A network is trained on annotated data, and then used to infer detections on unannotated data. Those detections are fed to a MOT algorithm, which outputs smoothed object trajectories. Finally, pseudo annotations are extracted from those trajectories and used for training.

3.2.2 Tracking Coordinate System

Tracking was performed in a world coordinate system. All tracking was performed on continuous sequences, and for each point in time, an estimate of the ego-vehicle state was available via GPS. For simplicity, the ego-vehicle state at the first time step was used to define origin in the world coordinate system. In particular, the longitudinal and lateral axes of the ego-centric coordinate system in the first time step was used to define the z and x axes of the world coordinate system, as visualized in Figure 3.3.

The detections provided by the network were given in the ego-vehicle coordinate system. Hence, the lateral and longitudinal positions had to be converted to world coordinates. This was done using the ego vehicle state across time. In particular, all detections at a given time step were rotated using the heading of the ego vehicle, compensating for the ego vehicles rotation relative to the first time step. Then, the position of the ego vehicle was added to the position of the detection to compensate for the ego vehicles movement from the origin. Figure 3.4 shows an example of the ego vehicle, three trajectories and multiple detections in the world coordinate system.

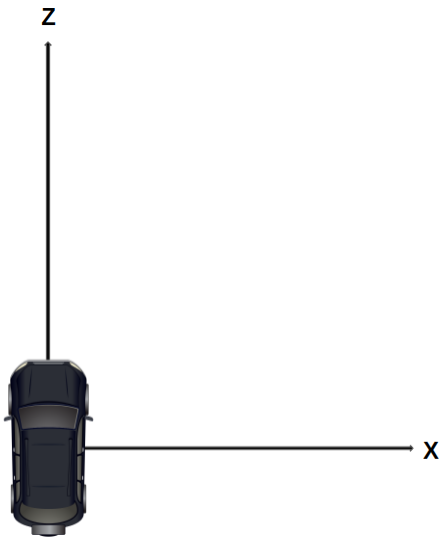

Figure 3.3: The lateral and longitudinal axes of the ego vehicle coordinate system at the first time step were used to create the world coordinate system.

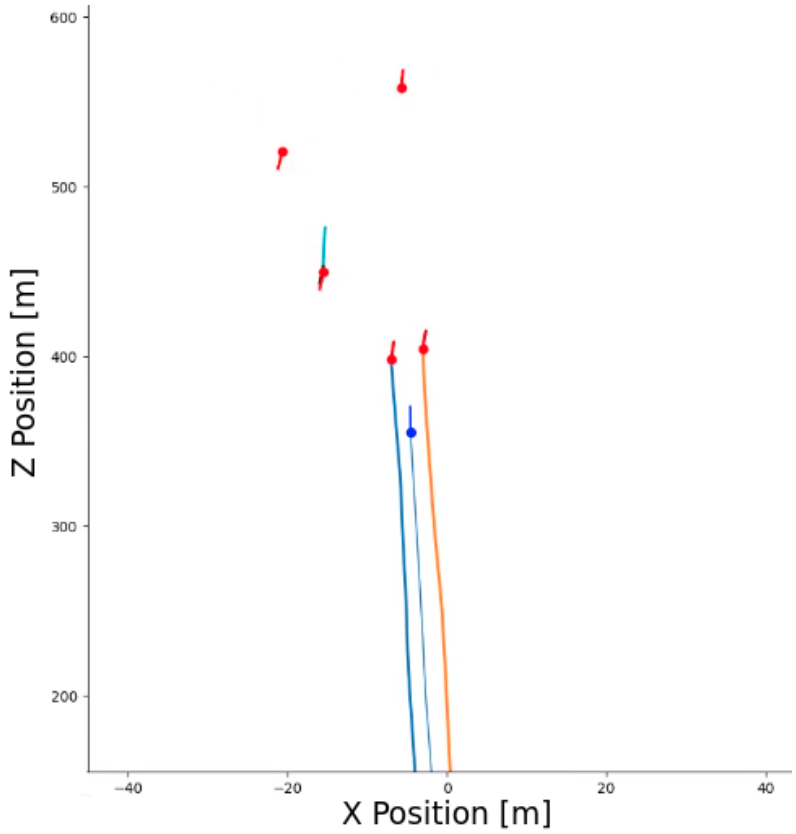

Figure 3.4: A visualization of tracking in the world coordinate system. The thin blue line shows the ego vehicle trajectory, and the blue dot shows the ego vehicle state at this time step. The orange, cyan and thicker blue lines are three trajectories. The red dots are detections. The bar of each dot represents the heading of the object.

3.2.3 Lidar Based Depth Estimation

Estimating depth from a single camera is not an easy task and is often described as an ill-posed problem since there is no unique depth solution to a 2D image [2]. Therefore, it is common to utilize lidar sensors to estimate the distance to objects in the surroundings of the ego vehicle. Since both images and lidar point clouds were available, it was decided to attempt to use the point clouds to better estimate the depth of the tracked objects.

The method can be described as follows. For each object detected by the network, the predicted bounding box of the object is used to extract a subset of points from the point cloud. Then, the subset of points is used to estimate the object depth. However, there are two practical challenges to deal with. First, each image has to be matched to a point cloud. Then, one must decide how to estimate the depth given a subset of lidar points. Let us begin by considering the first challenge.

The lidars used spin at a frequency of either 9Hz or 11Hz, while the camera captures images at 30Hz. Thus, there is no one-to-one correspondence between point clouds and images. Therefore, it was decided to match each point cloud to the image which is the most in sync. By the most in sync, we refer to the image that minimizes the relative angle between camera and lidar. A full lidar scan (360 degrees) was extracted at the time which the image was captured, and used for depth estimation.

For a given image and point cloud, the point cloud is projected into the image plane. To map lidar points to objects, the 2D bounding box of each object is used. To compensate for the fact that the 2D bounding box sometimes also captures parts of the background, the box size is reduced by 10% along each dimension. Then, the median of all of the points inside the resulting bounding box is used to determine the depth. For objects with no lidar points the camera-based depth is used. An example of an image and its corresponding point cloud is shown in Figure 3.5.

In order to estimate the measurement noise for the lidar measurements, we use 57 annotated frames. The algorithm described above is used to create a lidar-based estimate of the depth for all detected objects. Since the frames are annotated, the error can be calculated. The mean and standard deviation of these errors were used to model the measurement noise, and are given in Table 3.2.

Number of annotated frames	μ	σ
57	0.182m	0.345m

Table 3.2: The mean, μ, and standard deviation, σ of the lidar based depth estimate error.

Figure 3.5: An image and the corresponding lidar point cloud.

3.2.4 Trajectory Poisson Multi-Bernoulli Tracking Algorithm

In order to perform MOT, a Trajectory Poisson Multi-Bernoulli (TPMB) filter was used. The TPMB algorithm is described in Section 2.3.13, and we describe the settings used below.

To describe the motion of objects, a constant velocity model was used. The width, height, and length of the object were considered constant through time, i.e the vehicle does not change size during tracking. Furthermore, only longitudinal and lateral velocity were considered, meaning the altitude was modeled as a random walk. In addition, the yaw angle of the object was also modeled as a random walk. Object states are represented as column vectors with nine elements, detailed by Table 3.3.

Output	Description	Additive Noise
(w, h, l)	Three dimensional size: width, height and length	No
θ	Yaw angle	Yes
(x,y,z)	The center point of the object in world coordinates	Yes
(v_x, v_z)	Lateral & longitudinal velocity	Yes

Table 3.3: Initial model of the object state.

The motion model is given by

$$f\left(\begin{bmatrix} w \\ h \\ l \\ \theta \\ x \\ y \\ z \\ v_x \\ v_y \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w \\ h \\ l \\ \theta \\ x \\ y \\ z \\ v_x \\ v_y \end{bmatrix}. \tag{3.1}$$

The quantity T refers to the elapsed time in between measurements. The model was chosen based on the assumption that the objects which a vehicle commonly encounters move with a roughly constant velocity. The idea was that any minor accelerations will be captured by motion noise.

Some of the parameters were set using a calibration dataset. In particular, 40000 annotated images were selected uniformly at random. This dataset was disjoint from both the training and evaluation dataset. The network which is referred to as the baseline in Section 4 was used to infer detections on the images, and the mean and standard deviation of the errors along with precision and recall were used to parameterize some of the models, as detailed below.

A DNN based on ResNet [11] was used as the detector. For each object, the network outputs several properties. The properties relevant to the tracking algorithm are detailed in Table 3.4. From this, it is clear that all elements of the state except for the velocity are observable. Thus, the measurement matrix is given by

$$\begin{bmatrix} w \\ h \\ l \\ \theta \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} w \\ h \\ l \\ \theta \\ x \\ y \\ z \\ v_x \\ v_y \end{bmatrix}. \tag{3.2}$$

Output	Description
$p(c)$	The probabilities of each class for the object
(w, h, l)	Three dimensional size: width, height length
θ	Rotation (yaw) of the object
x, y, z	Three dimensional position

Table 3.4: The information provided by the object detector for each detection.

To account for the noise in the detections, the covariance matrix

$$\begin{bmatrix} \sigma_w^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_h^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_l^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_\theta^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_x^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_y^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sigma_z^2 \end{bmatrix} \tag{3.3}$$

with values from Table 3.5 was used to form a complete measurement model.

Parameter	Value
σ_w	0.174
σ_h	0.238
σ_l	1.457
σ_θ	$\pi/180$
σ_x	$0.00543 \cdot d + 0.3998$
σ_y	0.165
σ_z	$0.123 \cdot d + 0.0962$

Table 3.5: Values of the measurement model covariance parameters. σ_x and σ_z is dependent on the distance d to the object, see appendix B.

The values in Table 3.5 were obtained by calculating the mean and variance of the error using the calibration dataset. The exceptions are the yaw, for which the variance was tuned manually, and the x and z variance that was calculated as a function of the distance to the object, see appendix B.

A constant detection probability was used. The value was found by evaluating the network on the calibration dataset and using the resulting recall. This was found to be 0.672, and hence $P^D(x) = 0.672$ was used.

A constant intensity function was used to parameterize the Poisson Point Process which models clutter detections. The overall idea was to attempt to estimate the intensity fairly well in the part of the detections space where a large majority of the detections occurred and then use that intensity in the entire space.

As intensity represents the expected number of false detections per unit of area, it was decided to first estimate the number of false detections and then attempt to estimate which area they occurred in. To do this, all false detections resulting from applying the network to the calibration dataset were extracted. The longitudinal and lateral coordinates (in the ego-centric frame) of the detections were used to form a two-dimensional histogram with 100 bins along each dimension, as visualized in Figure 3.6. Then, a triangle was formed using the maximum longitudinal coordinate as height, and the maximum absolute lateral coordinate as width, also visualized in 3.6. Dividing the number of false detections by this area resulted in the clutter intensity being estimated to $\lambda_c(x) = 0.00003179$.

A Gaussian Mixture was used to represent the intensity of the PPP birth model. Four components were used; one for objects not moving, one for objects moving in the same direction as the ego vehicle, and two for objects moving in the opposite direction. Two components were used since such detections could have a yaw of either π or $-\pi$. Note that these were given half the weight of the other components.

The values in Table 3.6 specifies the values for the birth models. The values for x, z, and v_x were tuned manually, and the rest were found by calculating the mean values from the annotations in the calibration set.

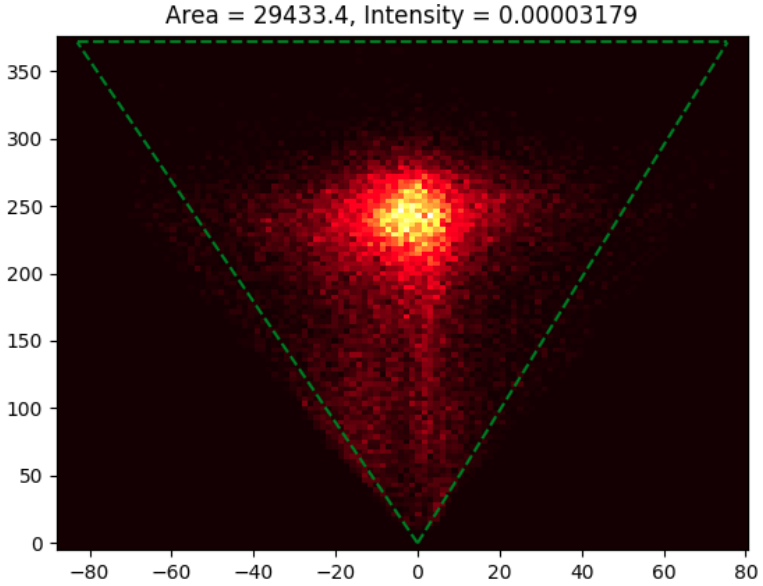

Figure 3.6: A visualization of the clutter estimation.

Property	μ	σ
width	2.23m	0.5m
height	2.2m	3.0m
length	6.63m	12.0m
x	0.0m	20.0m
y	0.24m	0.95m
z	25.0m	150.0m
v_x	0.0m/s	5.0m/s
v_z	*	15.0m/s
θ	*	1.82

Table 3.6: Mean values and variance used for the birth model. Both v_z and θ were set to allow births of stationary and objects traveling in either the same or opposing direction. For v_z $\mu \in \{0, 19, -19\}$ for θ $\mu \in \{0, \pi, -\pi\}$.

3.2.5 Creating Pseudo Annotations from Trajectories

In the previous subsection, we described an MOT algorithm and how it is applied to the output of an object detection network, resulting in a set of smoothed object trajectories. In this subsection, we describe a simple method for extracting pseudo-annotations from the object trajectories.

The algorithm is visualized in Figure 3.7 and can be summarized as follows. From the set of all object trajectories found by the MOT algorithm every trajectory with an existence probability $r_s \geq 0.8$ is extracted and considered for pseudo-annotation.

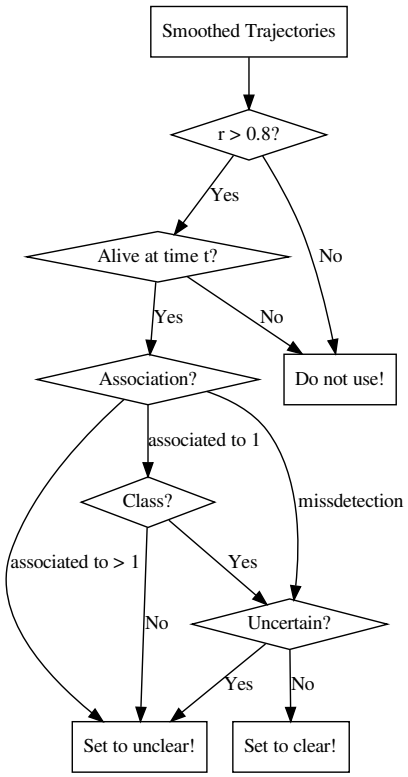

Figure 3.7: A visualization of the algorithm deciding if a pseudo-annotation should be created for a given time t and if it should be set to clear or unclear.

Because the object class is not a part of the tracking algorithm, the class for the trajectory is found by averaging the class probabilities of the detections associated to the trajectory. In the resulting vector, the class with the highest probability is used as a hard class for the trajectory. If the probability of that class is not larger than a threshold γ the entire trajectory is set to unclear. The algorithm then proceeds to iterate over all the time steps.

For each time step t, the algorithm iterates over all extracted trajectories and checks if it is alive at t. If so, a pseudo-annotation will be created using the mean of the

estimated state. The purpose of the remaining steps is to decide if the pseudo-annotation should be set to clear or unclear.

The algorithm proceeds to check if the trajectory has an associated detection. It does so by considering the normalized weights, or probabilities, of each association, and there are three cases.

1. If the misdetection hypothesis is the most likely the algorithm proceeds to check if the trajectory is uncertain.
2. If two detection hypotheses are the two most likely hypotheses, the algorithm considers the weight of the most likely one.
   (a) If it is larger than a threshold α, the algorithm proceeds to check the predicted class.
   (b) If not, the algorithm assumes that TPMB is not able to discern which of the two associated detections are from the object, and sets the pseudo-annotation as unclear.

In the case of a single associated detection, the class probabilities of that detection are considered. If the class with the highest probability differs from the trajectory class, the pseudo-annotation is set to unclear.

A trajectory is considered to be uncertain if the standard deviation of the positional estimate exceeds a threshold τ, and if if so the pseudo-annotation is set to unclear.

4
Results

The methods introduced in chapter 3 were evaluated by conducting a number of experiments. In this chapter, we introduce those experiments and present the results. We begin by explaining how the data used in the experiments was selected, and proceed to describe the object detection model. We then move on to detailing how the baseline models were trained. After that, we present the evaluation of our first method, the Noisy Student for Object Detection, and we show that it achieves an improvement over the baseline. Finally, we present the evaluation of the tracking based method. We show that the pseudo-annotations produced by our method are closer to ground truth than the raw detections.

4.1 Object Detection Model

In all experiments a deep neural network with an architecture based on ResNet [11] was used. The network is trained to perform object detection on images. For each image the network outputs a set of detections, and for each of these it predicts a number of properties, as detailed by Table 3.4. During training a batch-size of 32 was used.

4.2 Data Selection

In all experiments a combination of an annotated dataset, denoted L, and an unannotated dataset, denoted U were used. In order to achieve a fast training time it was decided to restrict the amount of data used such that $|U| + |L| = 20000$. In semi-supervised methods it is common to select $|U|$ such that it is much larger than $|L|$. Therefore, it was decided to choose U and L such that $|U| = 16000$ and $|L| = 4000$.

After the size of U and L were determined, the actual data was selected. L was chosen uniformly at random from the set of annotated images and the same sample was used for both Noisy Student for Object Detection and TOFS.

As detailed in Section 4.4, Noisy Student for Object Detection, was evaluated using several datasets, with different sampling strategies and settings. For the base, and most extensive experiments, 16000 unannotated image were sampled uniformly at random from a dataset of one million images.

In Section 4.7 TOFS is evaluated using one dataset. For TOFS, U had to be chosen differently since it operates on a sequence of images, and therefore images cannot be chosen entirely at random. It was decided to use 300 video sequences which were 10 seconds long at 30 fps. The start times of the sequences were chosen uniformly at random in a sequential manner. Once a sequence had been chosen, the time spanning from five minutes before the start, to five minutes after the end of the sequence, were made unavailable for upcoming selection. That is, sequences were chosen such that there were at least five minutes between each sequence. Given the length and frame rate of each sequence, 300 sequences amounts to 90000 images. Out of these, 16000 were chosen to form U by sampling images either at an even interval or randomly.

Furthermore, since the tracking uses the ego-vehicle state throughout each sequence, accurate position was needed. In some sequences, the position, which is determined by GPS, was noisy and inaccurate. This has a large negative impact on the performance of the tracking algorithm. Therefore it was decided to only select sequences with accurate and smooth positional data. To achieve this, all sequences where the position of the ego-vehicle differed by more than 1.6 meters between two consecutive frames, equivalent to about 175 km/h, were discarded. Furthermore, all sequences where the standard deviation of the GPS-position was above 0.4 m were also discarded.

4.3 Baseline Performance

In order to evaluate the effectiveness of the proposed methods it was decided to compare them to a baseline model trained only on L. This model was trained using the same augmentations and noise as for the Noisy Student method, detailed in Table 3.1. It was trained 5 times on the same data, but with different random seeds, meaning different training order. The baseline achieved a mean APS of 70.2% with a standard deviation of 0.17 p.p..

To investigate how performance scaled with the amount of annotated training data, the baseline model was also trained with additional annotated data. Table 4.1 shows the results, and gives an indication on how performance scales with $|L|$.

		Yaw	MAE Position (norm)			MAE Size (m)		
$\|L\|$	APS	(degrees)	X	Y	Z	W	H	L
4000	$70.2\% \pm 0.17$ p.p.	3.95 ± 0.07	0.001	0.006	0.065	0.113	0.144	0.758
5440	$71.5\% \pm 0.08$ p.p.	3.95 ± 0.02	0.001	0.006	0.062	0.112	0.140	0.732
6944	$72.4\% \pm 0.09$ p.p.	3.73 ± 0.02	0.001	0.006	0.061	0.111	0.139	0.727
8448	$73.3\% \pm 0.05$ p.p.	3.76 ± 0.06	0.001	0.006	0.059	0.110	0.137	0.730

Table 4.1: Baseline performance with different amounts of training data. The experiment with 4000 samples was repeated five times and the rest three times, all with different random seeds. The values are the mean calculated across the repetitions. For the APS and yaw the standard deviation is also given.

4.4 Noisy Student for Object Detection

In this section we present the results for the Noisy Student for Object Detection method. Firstly, the experiments for the unclear threshold is presented followed by the main results and finally further experiments run on other datasets.

4.4.1 Unclear Threshold

To find a suitable value for the unclear-threshold, five experiments were run for one teacher-student iteration. From the results, see Table 4.2, it was decided to use a threshold of 0.95 for further experiments. This means that only detections where the class confidence, except for background, is above 0.95 will be marked as clear. All detections that are within the confidence interval $(0.5, 0.95)$ will be marked as unclear. If the class-confidence is below 0.05 for all classes, no detection will be created and the area will be set as background.

Threshold	APS
0.0	70.1%
0.7	69.5%
0.8	70.5%
0.9	70.8%
0.95	71.3%
0.98	71.3%

Table 4.2: Results when evaluating the effectiveness of the unclear-threshold. The experiments were run once but still show a clear impact on the performance.

4.4.2 Performance

The main Noisy student for OD experiments were run for three teacher-student iterations. Before each iteration the student model was initialized from an ImageNet [7] pre-trained model. The experiment was run five times on the same data, dataset 0 in Table 4.5, but with different random seeds. From the results below we can see that on average the Noisy Student for OD increase the APS with 1 percentage point. This improved APS is similar, 71.3% and 71.5%, to the APS of the baseline trained with 5440 annotated images, see Table 4.1, which used 26% more manually annotated images. Furthermore, in Figure 4.1 we can see that the the Noisy Student for OD model is performing slightly worse than the baseline for early batches, but when converged it performs better on average.

	Baseline	Noisy Student for Object Detection		
		Iteration 0	Iteration 1	Iteration 2
APS	$70.2\% \pm 0.17$ p.p.	$71.3\% \pm 0.14$ p.p.	$71.1\% \pm 0.22$ p.p.	$70.9\% \pm 0.23$ p.p.

Table 4.3: APS performance for Noisy Student for Object Detection across the three student-teacher iterations.

	Baseline	Noisy Student for Object Detection		
		Iteration 0	Iteration 1	Iteration 2
Z Error (norm)	0.065 ± 0	0.068 ± 0.001	0.069 ± 0.001	0.068 ± 0
X Error (norm)	0.001 ± 0	0.001 ± 0	0.001 ± 0	0.001 ± 0
Y Error (norm)	0.006 ± 0	0.006 ± 0	0.006 ± 0	0.006 ± 0
Yaw (degrees)	3.95 ± 0.066	4.32 ± 0.072	4.42 ± 0.096	4.35 ± 0.093
Width Error (m)	0.113 ± 0.001	0.114 ± 0.002	0.113 ± 0.001	0.114 ± 0.001
Height Error (m)	0.144 ± 0.001	0.150 ± 0.002	0.151 ± 0.001	0.152 ± 0.002
Length Error (m)	0.758 ± 0.008	0.767 ± 0.015	0.768 ± 0.007	0.766 ± 0.003

Table 4.4: Object state estimate error for Noisy Student for Object Detection across the three student-teacher iterations.

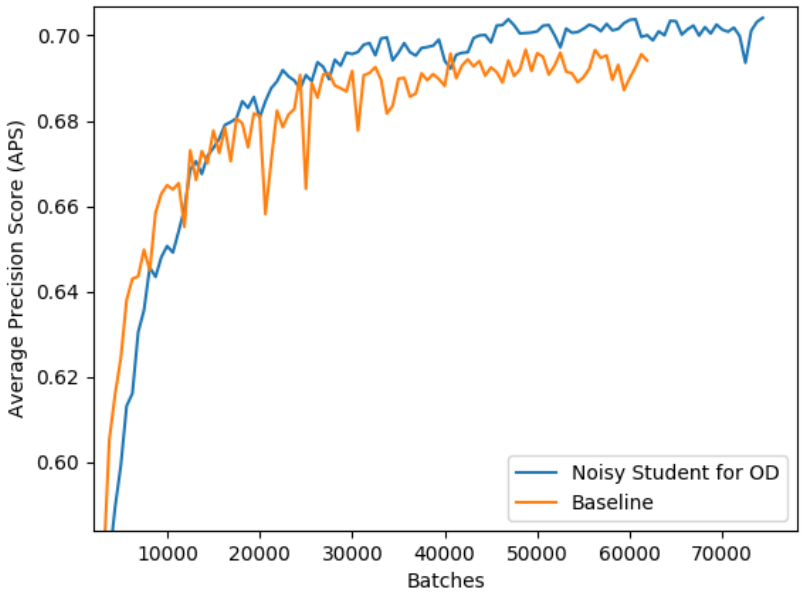

Figure 4.1: Noisy Student for OD vs Baseline average APS performance across the five experiments. Note that this is the average at each time, which means that the individual experiments can, and do, have higher APS. The experiments were run until no more APS improvement was seen.

4.4.3 Datasets & Extra experiments

Furthermore, we ran additional experiments on other datasets, sampling strategies and settings for the first iteration, see Table 4.5. The *Dataset* column indicates which dataset we sample from and the *Size* indicates the size of that dataset. All experiments still use a sample of 16000 images.

Dataset	Size	Sampling Strategy	GPS	Repetitions	APS
0 (Main)	1000000	Uniform Random	No	5	$71.3\% \pm 0.14$ p.p.
0	1000000	Uniform Random	Yes	1	$70.3\% \pm 0$ p.p.
1	100000	Uniform Random	No	5	$71.2\% \pm 0.11$ p.p.
2	100000	Uniform Random	No	1	$70.7\% \pm 0$ p.p.
3	90000	Uniform Random	Yes	3	$70.6\% \pm 0.21$ p.p.
3	90000	Even Interval	Yes	2	$70.3\% \pm 0.30$ p.p.

Table 4.5: The datasets used for the Noisy student for OD experiments. For each experiment a set of 16000 images were sampled from the dataset using the specified sampling strategy. The GPS column indicates whether images around points in time where the GPS-data have been unreliable have been filtered out and blacklisted from being sampled. This was done to make the sample compatible with TOFS. The APS columns provides the mean and standard deviation of the APS across the repetitions.

4.5 Tracking Algorithm Setup

As explained in Section 3.2, there are a number of hyperparameters associated to the method. In this section, we give a brief explanation of which values were used for which parameters and why. The values are summarized in Table 4.6.

Firstly, Lscan was set to 60 frames. Ideally, one would like to use as many frames as possible in order to propagate information further back in time. However, using large values increase the run time, and to allow for fast tracking it was decided to use 60 frames. In terms of time it is a reasonable value, as 60 frames correspond to 2 seconds.

Secondly, it was decided to set all trajectories with a standard deviation larger than 2 meters for the position (Z and X) to unclear. The reasoning behind this was to allow for some uncertainty, while removing the most uncertain trajectories. Lastly, the association threshold and class threshold were both set to 0.8.

Setting	Value	Description
α	0.8	Association threshold
γ	0.8	Class threshold
τ	2 m	Positional uncertainty
Lscan	60	No. time steps to smooth

Table 4.6: Settings used for tracking.

4.6 Quality of Smoothed Object Trajectories

Previously we have argued that, compared to the direct output of the network, the pseudo-annotations produced by TOFS should be closer to ground truth. The purpose of the experiment presented in this section is to examine if that holds or not. In what follows, the direct output of the network is referred to as 'detections' and the tracking based pseudo-annotations as 'pseudo-annotations'.

This experiment focuses on comparing the error of the detections to the error of the pseudo-annotations. The error is defined as the difference between the pseudo-annotations/detections and the ground truth (GT). In order to examine this, an annotated sequence described in Table 4.7 was used. To reduce the computational demands the sequence was divided into 9 equally large subsequences. The tracking was performed on these independently, using the settings in Table 4.6.

Frame rate	Length	Annotation frequency
30 fps	270 s	2 Hz

Table 4.7: Properties of the annotated sequence used to compare pseudo-annotations to detections.

From the resulting sets of trajectories, pseudo-annotations were created using the settings in Section 4.5. For each frame with a ground truth annotation, all pseudo-annotations were extracted. The pseudo-annotations that fulfilled the following requirements were used to perform a comparison of the error between the pseudo-annotations and the associated detection.

1. Pseudo-annotation is not unclear.
2. Pseudo-annotation has an associated detection.
3. Pseudo-annotation can be matched to a ground truth annotation.
4. Ground truth annotation is not unclear.
5. Ground truth annotation has 3D properties.

Furthermore, it is of interest to study how often an object is tracked without being detected by the network. Therefore, the total number of times a trajectory was not associated with a detection is given. However, this does not necessarily correspond to a misdetection by the network. Recall that if an object is detected, it is possible that the corresponding trajectory is not associated with it if the detection is sufficiently different from it. Therefore, when evaluating, all detections, and all pseudo-annotations corresponding to trajectories with no associated detection, attempts to be matched with GT using its 2D bounding box IoU. The number of GT objects which were not matched to a detection, but were matched to a pseudo-annotation, are reported as the number of false negatives found.

Lastly, to highlight the difference between using only camera and camera in conjunction with lidar, the experiment was conducted with both setups. The results are presented in the two subsections that follow.

4.6.1 Camera Based Detections

In this subsection we present the result of the comparison between camera based pseudo-annotations/detections and ground truth. Table 4.8 shows the Mean Absolute Error (MAE) for each 3D metric. Table 4.9 shows the number of pseudo-annotations used for the comparison along with the total number detections and ground truth objects in the sequence. Table 4.10 shows the total number of pseudo-annotations, the number of pseudo-annotations with and without an associated detection, and the total number of false negatives found.

Note that not all pseudo-annotations with an associated detection were used for comparisons, as some of the corresponding ground truth annotations were either marked as unclear or missing 3D properties.

Metric	Pseudo-Annotation	Detection	Relative Improvement
X position [m]	0.35	0.36	+2.8 %
Y position [m]	0.18	0.18	±0 %
Z position [m]	4.11	4.74	**+13.3 %**
Width [m]	0.16	0.16	±0 %
Height [m]	0.17	0.18	+5.6%
Length [m]	0.39	0.42	+7.1 %
Yaw [rad]	0.1	0.1	±0 %

Table 4.8: The mean absolute errors relative to ground truth for the pseudo annotations produced by TOFS and the detections. Values have been rounded to three decimal places. The last column shows the relative improvement as a percentage calculated using the rounded values.

Ground Truth Annotations	Detections	Comparisons
3265	2283	673

Table 4.9: The first two columns show the number of ground truth objects in the sequence and the number of detected objects. The last column shows the number of comparisons used to create the means in Table 4.8.

Pseudo Annotations	713
Associated	684
Not Associated	29
False Negatives Found	8

Table 4.10: The first row shows the total number of clear pseudo-annotated objects for the sequence when using camera based detections. The second row shows the number of pseudo-annotations associated to a detection, while the third row shows the opposite. The last row shows the number of found false negatives.

As seen in Table 4.8, the the Z position (depth) shows the largest improvement. However, it is also the metric with the largest MAE for the pseudo-annotations. To investigate why the error was not reduced further, the metric was studied in further detail. Figure 4.2 shows the distribution of the error across all trajectories. Figure 4.3 shows the moving average of the error, both for the detections and pseudo-annotations, for a number of trajectories across time. As seen in these figures, the error across all trajectories is, roughly, normally distributed with a mean of -1.93. However, for several of these in Figure 4.3, the mean seems to be farther from zero.

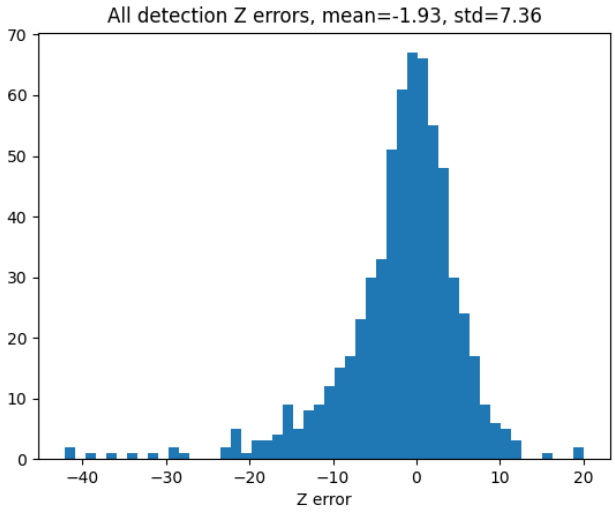

Figure 4.2: The error of the detections for the Z position plotted as a histogram with 50 bins. The x axis shows the bin limits and the y axisamplings shows the number of errors in each bin. The mean and standard deviation is shown at the top.

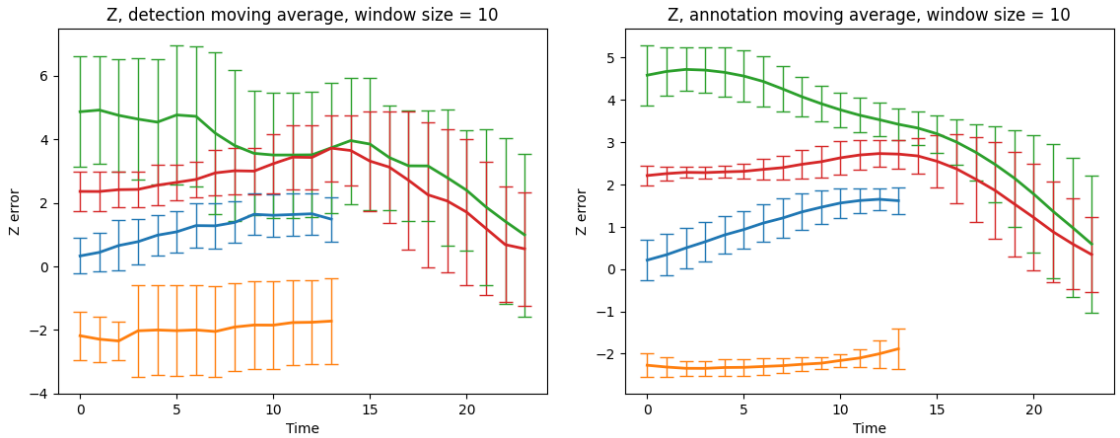

Figure 4.3: The moving average of the Z position error per trajectory for one sequence. Each line shows the moving average of a single trajectory across time. The previous ten values were used to calculate the average at each point.

4.6.2 Lidar and Camera Based Detections

In this subsection we present the result of the comparison of pseudo-annotations and detections, based on both camera and lidar, to ground truth. Table 4.11 shows mean absolute error for each 3D metric. Table 4.12 shows the number of objects used for the comparison along with the total number detections and ground truth objects in the sequence, while Table 4.13 shows statistics for the number of pseudo-annotations. Similarly to the last section, not all pseudo-annotations with an associated detection were used for the comparisons.

Metric	Pseudo-Annotation	Detection	Relative Improvement
X position [m]	0.32	0.36	+11.1%
Y position [m]	0.19	0.19	±0%
Z position [m]	1.01	2.12	**+52.4%**
Z position lidar [m]	0.35	0.39	+10.3%
Width [m]	0.17	0.17	±0%
Height [m]	0.17	0.17	±0%
Length [m]	0.4	0.43	+7%
Yaw [rad]	0.06	0.06	±0%

Table 4.11: The mean absolute errors relative to ground truth for the pseudo annotations produced by TOFS and the detections based on both lidar and camera. Values have been rounded to three decimal places. The last column shows the relative improvement as a percentage calculated on the rounded values. Note that the third row corresponds to the error for all detections, while the camera based detections have been excluded on the fourth row.

Ground Truth Annotations	Detections	Comparisons
3265	2283	514

Table 4.12: The first two columns show the number of ground truth objects in the sequence and the number of detected objects. The last column shows thsamplinge number of comparisons used to create the means in Table 4.11.

Pseudo Annotations	654
Associated	528
Not Associated	126
Found False Negatives	8

Table 4.13: The first row shows the total number of clear pseudo-annotated objects for the sequence when using both camera and lidar based detections. The second row shows the number of pseudo-annotations associated to a detection, while the third row shows the opposite. The last row shows the number of found false negatives.

As seen in Table 4.11, the Z position (depth) shows the most improvement, which was also the case in Section 4.6.1. Clearly, introducing the lidar resulted in a lower MAE for both detections and pseudo-annotations. To examine how much of the improvement is due to the lidar itself, the values in the fourth row of Table 4.11 were added. Furthermore, as shown in Figure 4.4, the distribution of detection errors has a lower standard deviation and a mean closer to zero.

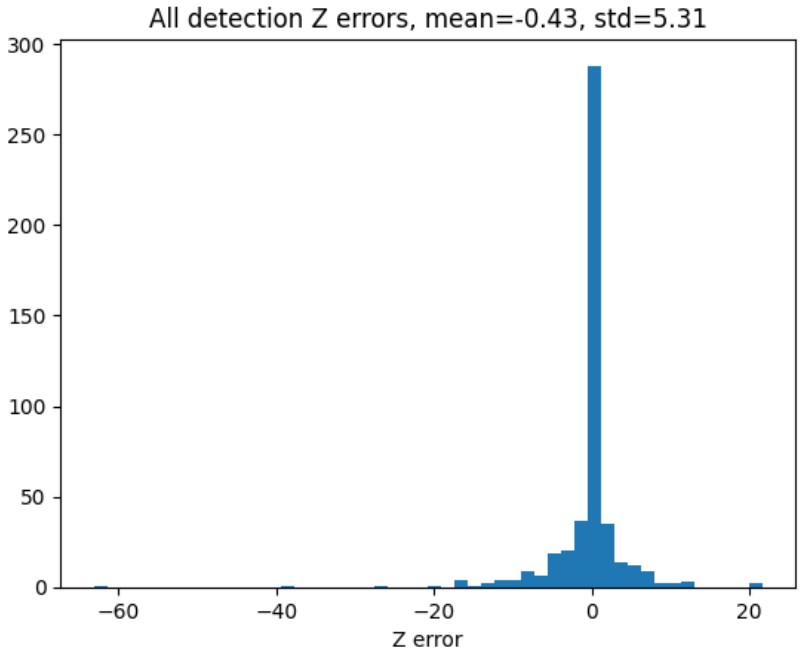

Figure 4.4: The Z position error plotted as a histogram with 50 bins. The x axis shows the bin limits and the y axis shows the number of errors in each bin. The mean and standard deviation of the errors are shown at the top.

A consequence of using both lidar and camera based detections is that the error changes more across time. Figure 4.5 shows the Z position error for several trajectories from the same subsequence as in Figure 4.3. The plot on the left in Figure 4.5 shows the error across time. Clearly, there are several spikes due to the lidar based detections having a lower error than the camera based detections. The plot to the right in Figure 4.5 shows the error across time for the lidar based detections only.

Lastly, Figure 4.6 shows the moving average of the Z position error across time for both the detections as well as the pseudo-annotations. This is the same form of visualization as in Figure 4.3.

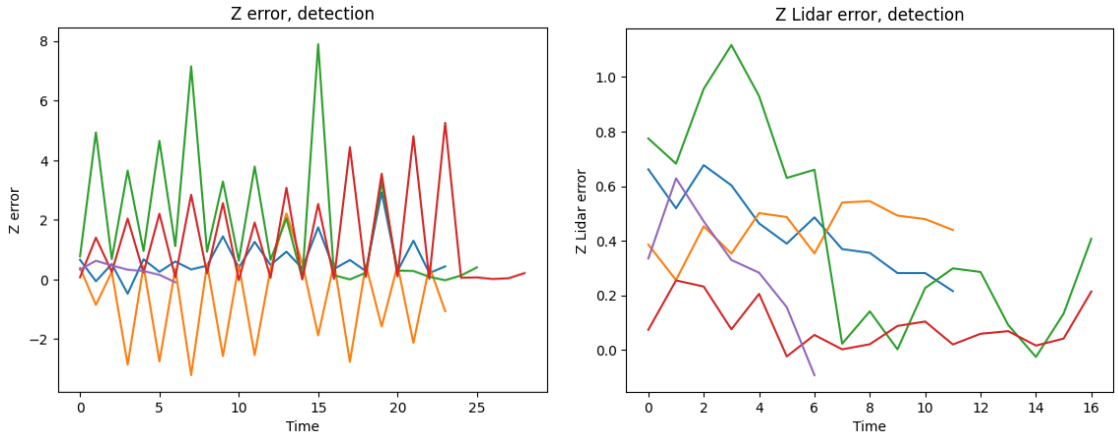

Figure 4.5: The left figure shows the Z position error per trajectory for all detections in a subsequence of the annotated sequence. The right figure shows the error for the same detections when excluding all camera based detections. Lidar is sampled at a lower frequency than the camera, which is the cause for the difference in length of the sequences.

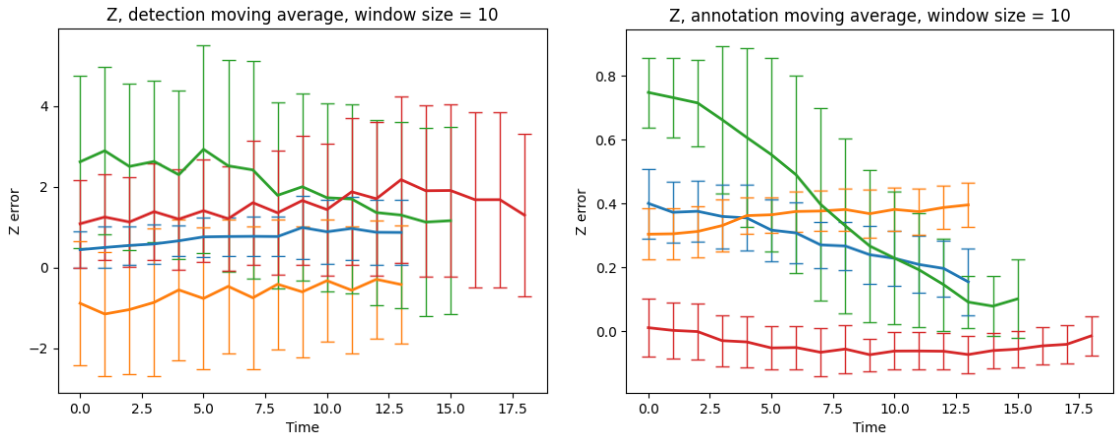

Figure 4.6: The moving average of the error in Z position per trajectory. Each line shows the moving average of a trajectory across time. The previous ten values were used to calculate the average at each point.

4.7 Training using Smoothed Object Trajectories

In this section we present the results obtained when training the OD network using 4000 annotated images and 16000 images which have been pseudo-annotated using TOFS. An example of a pseudo-annotated image can be seen in Figure 4.7, which shows both clear and unclear annotations.

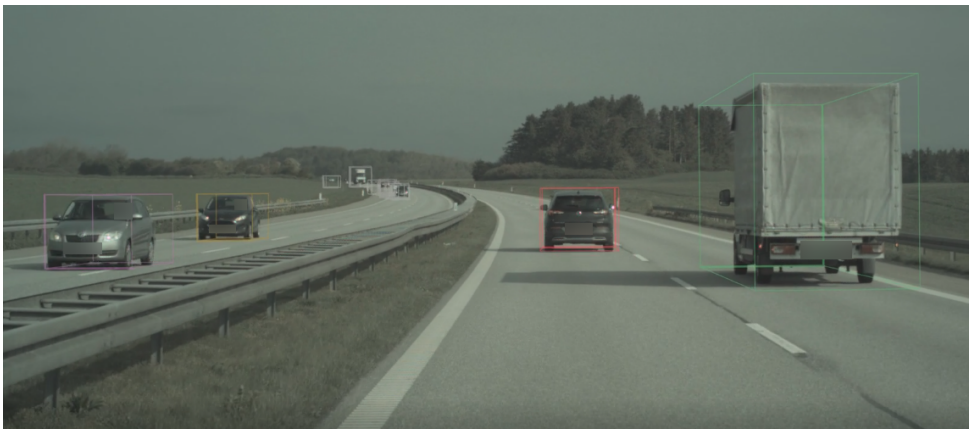

Figure 4.7: An example of a pseudo-annotated image. The colored bounding boxes in the foreground correspond to clear pseudo-annotations, while the gray ones in the background are unclear.

As in the previous section, the tracking in TOFS was performed using two setups. In the first setup TOFS was applied to only camera measurements, while both camera and lidar measurements were used in the second setup. Table 4.14 and 4.15 shows the results obtained when training on the extracted pseudo-annotations respectively. The first column in both tables shows the performance of the baseline, which was trained on 4000 annotated images, while the remaining columns show the results from training on both annotated as well as pseudo-annotated images.

Furthermore, different strategies for selecting which frames to include in the training were tested. The 'Interval' strategy selects 16000 images by using every nth image frames at an even interval. The 'Greedy' strategy selects the 16000 images with the most pseudo-annotations. However, once an image has been selected, all images which are 0.25 seconds before and after it are excluded from future selection. The networks were trained for 400 epochs with a batch size of 32.

It is clear from Tables 4.14 and 4.15 that the APS decreases in both setups and for all selection strategies. Furthermore, it is worthwhile to note that the length seems to be slightly improved in all experiments. Additionally, the depth shows minor improvement when using only camera measurements, as shown in Table 4.14.

To investigate why the APS decreased, it was decided to study the number of clear and unclear annotations. Table 4.16 shows the average number of clear and unclear pseudo-annotations for the different datasets used to evaluate Noisy Student for OD.

Metric	Baseline	Interval	Greedy
APS	$70.2\% \pm 0.17$ p.p.	$68.6\% \pm 0.3$ p.p.	$68.7\% \pm 0.3$ p.p.
Z position (norm)	0.065 ± 0	0.064 ± 0.01	0.064 ± 0
X position (norm)	0.006 ± 0	0.006 ± 0	0.006 ± 0
Y position (norm)	0.001 ± 0	0.001 ± 0	0.001 ± 0
Width [m]	0.113 ± 0	0.111 ± 0	0.111 ± 0
Height [m]	0.144 ± 0	0.142 ± 0	0.141 ± 0
Length [m]	0.758 ± 0.01	0.732 ± 0	0.733 ± 0
Yaw [deg]	4.01 ± 0.07	4.07 ± 0.02	4.22 ± 0.03

Table 4.14: The resulting performance when training on pseudo-annotations from TOFS. The pseudo-annotations were created using only camera based measurements. The results in the 'Interval' and 'Greedy' columns were collected from three experiments, while the results for the baseline was collected from five experiments. The standard deviation has been rounded to two decimal numbers.

Metric	Baseline	Interval	Greedy
APS	$70.2\% \pm 0.17$ p.p.	$67.8\% \pm 0.1$ p.p.	$68.1\% \pm 0.1$ p.p.
Z pos. (norm)	0.065 ± 0	0.065 ± 0	0.065 ± 0
X position (norm)	0.006 ± 0	0.006 ± 0	0.006 ± 0
Y position (norm)	0.001 ± 0	0.001 ± 0	0.001 ± 0
Width [m]	0.113 ± 0	0.113 ± 0	0.115 ± 0
Height [m]	0.144 ± 0	0.144 ± 0	0.144 ± 0
Length [m]	0.758 ± 0.01	0.745 ± 0.01	0.745 ± 0.01
Yaw [deg]	4.01 ± 0.07	3.96 ± 0.06	3.89 ± 0.04

Table 4.15: The resulting performance when training on pseudo-annotations from TOFS. The pseudo-annotations were created using both camera and lidar based measurements. The results in the 'Interval' and 'Greedy' columns were collected from three experiments, while the results for the baseline was collected from five experiments.

Dataset	Size	Sampling Strategy	Filter GPS	Clears	Unclears
0	1000000	Uniform Random	No	8.2 ± 0.3	2.5 ± 0.8
0	1000000	Uniform Random	Yes	7.5	0.9
1	100000	Uniform Random	No	13.16 ± 0.5	2.88 ± 0.8
2	100000	Uniform Random	No	12.1	1.8
3	90000	Uniform Random	Yes	8.1 ± 0.6	3.6 ± 1.6
3	90000	Even Interval	Yes	7.7 ± 0.3	3.5 ± 0.8

Table 4.16: Shows the average number of clear and unclear pseudo-annotations per image for Noisy Student for Object Detection.

Dataset	Size	Sampling Strategy	Lidar	Clears	Unclears
3	90000	Even Interval	No	2.3	8.8
3	90000	Greedy	No	3.43	10.0
3	90000	Even Interval	Yes	2.9	10.3
3	90000	Greedy	Yes	2.4	7.9

Table 4.17: Shows the average number of clear and unclear pseudo-annotations per image for TOFS.

5
Discussion

In this chapter, we discuss the findings of the experiments and results introduced in chapter 4. We begin by discussing some potential issues with the data sampling, continue with the Noisy Student for Object Detection and it is results. Finally, we discuss the tracking-based method and attempt an analysis of its results and future work.

5.1 Bias in the Data Sampling

Ideally, the unannotated samples used would have been sampled truly randomly, but because of the needs of the tracking algorithm, a bias had to be introduced in the sampling by filtering out sequences with bad GPS data. The sequences where the GPS position is unreliable are probably highly correlated. For example, driving through a tunnel, the GPS position would almost always be unreliable if not non-existent. This would mean that the model would never be trained on any of these scenarios since they would all be filtered out. The opposite could also be true, that some scenarios are over-represented because the GPS position is unusually reliable. It was noticed when inspecting the video data that several scenarios are on top of tall bridges or similar scenes. This could be a coincidence but it could also be that these scenarios often have highly accurate GPS data, which makes them more common. Looking at the performance of the Noisy Student for Object Detection experiments in 4.5, there seems to be a correlation between filtering GPS-data and a lower APS which could hint that the sample is biased.

5.2 Noisy Student for Object Detection

In this section, we will discuss our results and findings for Noisy Student for Object Detection. We begin by discussing the impact of the unclear threshold, move on to how fast the method improves during training compared to the baseline, and finish with a discussion of how the noise used can impact the result.

5.2.1 The use of unclear pseudo-annotations

Similarly to the findings in *FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence* [17], we found it important to mark low-confidence annotations as unclear, see Table 4.2. Furthermore, in line with their findings, we found the optimal threshold, with regards to APS, around 95% confidence of the detections and that increasing the threshold further did not seem to hurt the APS but neither did it improve it. This is most likely because the networks quickly start predicting very high confidence for most detections.

5.2.2 Speed of Improvement

Comparing the training of the baseline and Noisy Student for Object Detection, see Figure 4.1, we see that for early stages the baseline slightly outperforms the Noisy Student for Object Detection model while later on Noisy Student for Object Detection surpasses the baseline performance. Intuitively this makes sense since the pseudo-annotations are most likely not as good as the human-made annotations. The baselines, which only train on manually annotated data, will have higher quality annotations fed to it and will be able to make better and more informed decisions when updating its weights. The Noisy Student for Object Detection model has far more data available, though of lesser quality. What is interesting is that it still managed to surpass the baselines APS performance, which should indicate that the pseudo-annotations are good enough to be able to extract useful information.

5.2.3 Importance of Noise

Compared to the noise used in the original Noisy Student model, we are using fewer noise sources. Firstly, we do not use stochastic depth, and RandAugment [6], which is used in the original paper, uses far more augmentations than what was used in our experiments. It would be interesting to investigate if adding more noise could improve the performance of the algorithm. A problem with implementing this would be that it is not clear how all augmentations used for image classification can be adapted to OD. Furthermore, not all augmentations commonly used in OD are beneficial in the AD domain, and if not, they will likely not increase the performance of the Noisy Student for Object Detection either. This could also be why the performance seems to be declining after the first student-teacher iteration.

5.3 Tracking Objects for Self-supervision

In this section, we discuss the comparison of pseudo-annotations and detections to ground truth, presented in Section 4.6. We begin by discussing the results for the 3D metrics and focus on the impact of the lidar data. Then, we move on to the assumption of Gaussian measurement noise with zero mean. The section ends with two subsections discussing the choice of state representation, tracking algorithm and motion model.

5.3.1 Using Lidar Improves Depth Estimation

In this subsection, we discuss the impact of using lidar point clouds to estimate the depth. In particular, we discuss the improvement observed in Section 4.6, and attempt to explain the results.

Firstly, an infinite number of 3D scenes can result in the same 2D image, which is why monodepth is commonly considered an ill-posed task. Because of this, the network has to rely on cues such as occlusion, perspective, texture, as well as object size and position to predict the depth [2].

Therefore, when replacing the camera-based depth with the lidar-based depth for several of the frames, it is not surprising that the MAE for the detections is reduced greatly, as made clear from Tables 4.8 and 4.11. This is perhaps best exemplified by the error for some trajectories, as shown in Figure 4.5. Clearly, the pseudo-annotations are improved as well. Simply using better detections when tracking surely helps, but there is more to it.

Firstly, recall that both lidar and camera-based measurements are used, as made clear by Figure 4.5. Yet, the error for the resulting trajectories in Figure 4.6 is not as spiky as in Figure 4.5. The reason is the difference in measurement noise for the lidar and camera measurements. A large measurement noise is used for the camera-based measurements, while the measurement is low for lidar-based measurements.

Consequently, the depth is dominated by the lidar measurements throughout the entire sequence. That is, at time steps when no lidar measurement is available, the depth is largely based on the Kalman prediction. It is also updated using future lidar measurements as a part of the smoothing. Note that there is still a point in keeping the camera-based depth for frames without a point cloud, as it helps in associating trajectories to measurements.

Secondly, it is clear from Figure 4.4 that the mean is shifted closer to zero while the standard deviation is lower compared to 4.2. In addition, Figures 4.3 and 4.6 show that the moving average of several trajectories have been shifted closer to zero. This is important, since the tracking algorithm assumes Gaussian noise with zero mean.

Since the mean is closer to zero, the estimated depth should be better, which is the case both when considering all measurements as well as lidar measurements only according to Table 4.11. As seen in Figure 4.5, the error per track is still biased, which partially explains the remaining error. The method for extracting depth from point clouds also contributes to the error, and improvements are discussed in Section 5.5.1

5.3.2 Assuming Gaussian Noise

An important assumption made by the tracking algorithm is that the measurement noise is Gaussian with zero mean, as mentioned in Section 2.3.13. However, when studying Figures 4.2 and 4.3, it is clear that for the particular metric, the assumption is violated. In this section, we discuss the impact of that on the results.

Firstly, when using only camera-based measurements, it makes intuitive sense that the error is not zero mean for a fixed time window and track, especially considering the callenges with monodepth explained in the previous section. As an example, consider the difference between the two consecutive frames in Figure 5.1. It is hard to discern any difference for the two objects marked with red bounding boxes. Therefore, one would expect a network to make similar predictions in the two frames.

Figure 5.1: An example of two consecutive frames. Two objects of interest have been marked with red bounding boxes.

Secondly, consider the distribution of errors for continuous targets over the network's training data. That distribution should be zero mean. If the opposite was true, simply adding or subtracting the mean from the network output would improve the average prediction. The network should be able to learn such a rudimentary rule. Thus, it makes intuitive sense that the mean of the distribution of all errors is close to zero for all metrics in the annotated sequence used in Section 4.6. As can be seen in Figures 4.2, 4.4 and the corresponding Figures in appendix A, the mean is usually close to zero. The reason that it is not exactly zero is partially that only tracked objects are considered, and partially because the distribution of samples in the annotated sequence is not the same as in the training data.

We have now established that the error distributions we see make sense. What are the implications for our method? The fact that the mean is not zero for a given track is clearly a problem since the filter assumes that it is. In fact, it means that there is a limit on how much the performance can be increased. The filter is able to solve the Gaussian errors, but not constant error, represented by the offset of the mean from zero.

As additional examples of this, consider the object dimensions. Figures A.6, A.8 and A.8 shows a moving average of the width, height and length errors per track. Clearly

the width and height means are closer to zero than the length, but the underlying issue is the same. One way to deal with this issue is to attempt to estimate the offset from zero. This is discussed further in Section 5.5.2.

5.3.3 Choice of State and Tracking Algorithm

In this thesis, we chose to represent the state with a single Gaussian. A key characteristic of TPMB is that the Poisson Multi Bernoulli Mixture (PMBM) density obtained from the update step is projected to a PMB density in each iteration. Since state is represented with a single Gaussian, this corresponds to projecting a mixture of Gaussians into a single Gaussian. This allows for lower computational complexity and a simplified prediction step. However, using a single Gaussian to represent the state is not as powerful as a mixture.

As an example, consider Figure 5.2, which shows a vehicle and several measurements across time. When considering all measurements, it seems like the vehicle could either have traveled along the 'upper' or 'lower' path. In this case, it makes intuitive sense to represent the state as a mixture of two Gaussians, one for each track as visualized in Figure 5.3. However, in our case, a single Gaussian will be used. Assuming the measurements are equally probable, the resulting Gaussian would have a mean located in between the measurements with a larger standard deviation, as visualized in Figure 5.4.

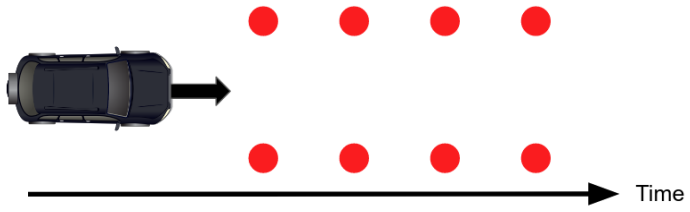

Figure 5.2: A vehicle and measurements represented as red dots. The red dots indicate that there are two possible trajectories. This is a potentially problematic situation for TPMB.

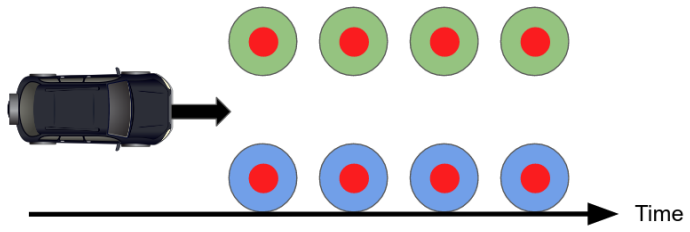

Figure 5.3: An example of how using a mixture representation for the state could mitigate the problem shown in Figure 5.2

As this example shows, mixture representations are better at capturing situations where several possibilities arise. This was encountered in practice as well. Figure 5.5

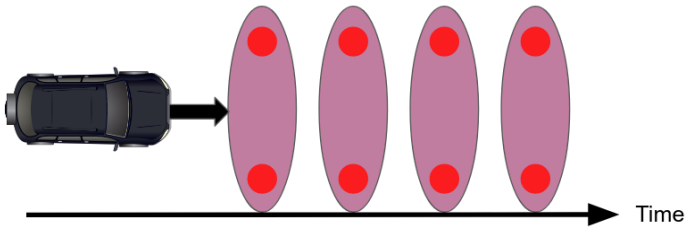

Figure 5.4: A vehicle and measurements represented as red dots. The red dots indicate that there are two possible trajectories. This is a potentially problematic situation for TPMB.

shows two bounding boxes being 'pulled' towards each other due to the detections being close to both trajectories. We refer to this as 'merged' trajectories.

To mitigate this problem a threshold α for the minimum detection association weight was used, as described in Section 3.2. This worked well in the sense that the particular states corresponding to merged trajectories were often marked as unclear.

However, there are also cases where they were not, as shown in Figure 5.5. This partially explains why the estimated position can be worse than the detections. An alternative would have been to represent the state as a Gaussian mixture within the TPMB framework. Another possibility would be to keep the state as a single Gaussian, and instead use a TPMB Mixture filter (TPMBM), resulting in a PMBM density. This is discussed further in Section 5.5.3.

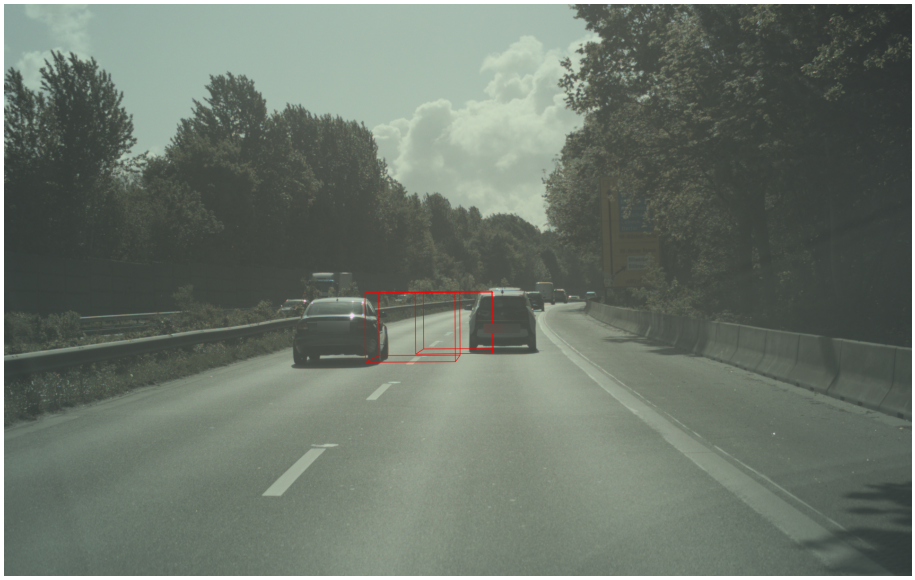

Figure 5.5: An example of a case which TPMB struggles with. The two red bounding boxes represent two tracked vehicles. Due to the detections being close to each other, both detections are used to update both trajectories. This results in the trajectories becoming merged, placing them in between the vehicles.

5.3.4 The Constant Velocity Motion Model

As described in Section 3.2 a constant velocity (CV) motion model is used. Using a CV model allowed for a simple implementation since its linear and thus allows for the use of a Kalman filter. One of the drawbacks with CV is that it does not include the yaw. Clearly the yaw is related to both position and velocity. However, in the CV model described in Section 3.2 it is modeled as a random walk. This explains why TOFS did not improve the estimated yaw, as seen in Tables 4.8 and 4.11.

An alternative to CV is the coordinated turn model (CT), which models the relation between yaw, velocity and position. Since the model is not linear, an Extended Kalman Filter (EKF) was implemented. Some experiments were done in a fairly late stage of the project. However, the results did not seem promising, and because of the lack of time, no further experiments were done.

As exemplified in Figure A.12, the yaw for a fixed track seems to be biased, as shown previously for the depth and dimensionality. This corresponds well to what was observed in the experiments - several tracks had a biased yaw, which led to worse positional estimates.

However, given more time, it seems likely that one should be able to tune the noise parameters such that the CT model achieves as good positional estimates as the CV model. For example, setting the measurement noise for the yaw to zero, while tuning the motion noise for the position for CT should give performance similar to CV. Therefore, future work should most likely attempt to use a CT model instead. This would hopefully allow TOFS to improve the yaw as well.

5.4 Comparing the Methods

In this section, we discuss the main differences between Noisy Student for Object Detection and TOFS in terms of performance. We attempt to explain the differences and to highlight the advantages and disadvantages of the two methods.

5.4.1 Average Precision

There is a clear difference in performance in regards to the precision, as made clear by Sections 4.4 and 4.7. Noisy Student for Object Detection improves the APS, while TOFS reduces it, which was not expected. When creating TOFS, it was thought that the use of unclear pseudo-annotations would mean that any object which was not tracked would be fully excluded from the image, and would thus not impact the network's confidence. Similarly, by including tracked objects, the network's confidence in those objects would be increased.

By comparing Tables 4.16 and 4.17 we see that the number of clear pseudo-annotations differs substantially between the two methods. Because both methods primarily use the 2D bounding box from the network output (TOFS uses a projection of the 3D

bounding box to 2D when there is no associated detection), the main difference seems to be the number of unclear pseudo-annotations, rather than the 2D bounding boxes. We believe that the low number of clear pseudo-annotations for TOFS is what caused the decrease in APS.

In particular, images with pseudo-annotations from TOFS include vastly fewer objects than images with pseudo-annotations from Noisy Student for Object Detection. This is most likely true for a majority of the images with ground truth as well. Therefore, it makes intuitive sense that the network should predict fewer objects overall. That is, if the network is trained on a dataset with one to two objects in a vast majority of the images, the network will most likely predict one to two object during inference as well.

In addition, since the 2D bounding boxes are not perfect, it is possible that even though objects are marked as unclear, certain parts of the vehicles are outside the 2D bounding box. For example, in Figure 5.6, the 2D bounding box of the truck in the middle of the picture does not include the top or the far right of the vehicle. The network will be taught not to predict an object there, even though there are clear object features present. It makes intuitive sense that this should lead to a decrease in performance. Since there are far more unclear objects for TOFS than for Noisy Student for Object Detection, these cases are more common there.

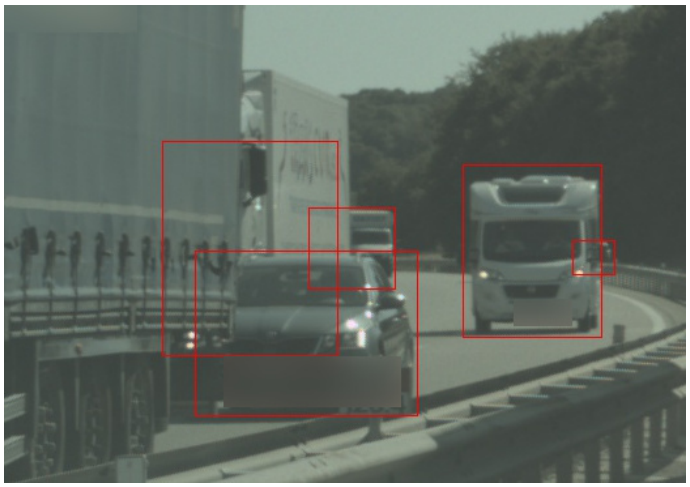

Figure 5.6: An example of an unclear annotation with a 2D bounding box which does not fully cover the object.

5.4.2 Position, Dimension and Rotation

While TOFS did not achieve as high APS as Noisy Student for Object Detection, it performed better for the 3D metrics. As shown in Tables 4.4 and 4.14, TOFS achieves slightly better length estimates, and show insignificant or no changes for the rest of the 3D metrics compared to the baseline, while Noisy Student for Object Detection increases the error.

The main reason for the high error of the Noisy Student for Object Detection's is that it does not have an uncertainty measure for these metrics. That is, the network does not provide a confidence for the prediction of these values. This can be compared with the class probability, which allowed the algorithm to set uncertain detections to unclear, which clearly lead to a performance increase. Thus, for the algorithm to achieve a performance increase for these metrics, a similar uncertainty estimate is needed.

TOFS on the other hand relies on an MOT algorithm. Using the measurement and motion model, the algorithm can account for measurement noise while having a prior on the motion of objects. This, in combination with the smoothing, is why TOFS can produce higher quality pseudo-annotations. However, the 3D metrics performance of the model trained using TOFS show small or no improvements compared to the baseline. This is surprising considering the pseudo-annotations from TOFS are better than the raw detections. Especially for the depth, which showed large improvements.

However, while the pseudo-annotations are better, they still have a non-zero error, and thus training on them will introduce biases. Hence, it is not obvious that this will result in better performance than the baseline. Furthermore, even though the depth of the pseudo-annotations are better than the detections, the majority of the pseudo-annotations are unclear. Because few pseudo-annotations are clear, it is possible that they will not have a significant impact during training.

5.5 Improving Trajectory Uncertainty

In this section, we discuss what we believe are the most promising improvements to TOFS. We begin by discussing a more robust depth estimation from point clouds. Then, we touch upon a simple way of estimating constant errors, which could potentially solve the issues discussed in 5.3.2. We move on to TPMBM, which is a natural step in improving the tracking performance of TOFS.

5.5.1 Robust Depth from Point Clouds

Ideally, the MAE of the lidar-based detections should be zero, simply because lidar is used by the annotator to create ground truth. One reason that the error is not zero is that our method fails to identify the center point of the vehicle. Several things could be done to improve this.

One thing that would greatly improve this would be to have a better method for extracting the subset of lidar points used for the estimation. Currently, we are using all the points inside the objects reduced 2D bounding box. This does not work well for occluded objects.

Once a better method for selecting the subset of points has been created, a better method for identifying the center of the vehicle is needed. Currently, half of the predicted length is added to the median, hoping that this is somewhat close to the center. This works well if the vehicle is headed straight towards the ego-vehicle or straight away from the ego-vehicle and the length estimate of the vehicle is accurate. Obviously, these conditions are often not met.

An alternative to our approach would be to cluster the lidar points into objects, as suggested in [4]. Obtaining a more robust estimate of the depth should lead to a lower error, and increased certainty in the estimate. This means that more object could be tracked, and that more pseudo-annotations would be set to clear.

5.5.2 Estimating Constant Errors

In Section 5.3.2 we discuss the impact of the assumption of Gaussian noise with zero mean. As discussed, one way to deal with the fact that the mean is not zero is to attempt to estimate the offset. For several variables, this is very hard since we do not get to directly observe the offset. However, it is possible to do this for the depth, if one assumes that the offset is zero for the lidar measurements.

In practice, this could be done by assuming that the depth is the sum of the actual depth and the offset. The offset could be modeled as a random walk, which would involve adding it to the motion model. For camera-based measurements, a measurement model which treats the depth measurement as the sum of the offset and the actual depth would need to be used. For lidar-based measurements, a measurement model which treats the offset and the depth as directly observable would need to be used. Creating these measurement model would involve creating new measurement matrices and tuning the measurement noise.

This is conceptually simple, but was not done due to time constraints. However, if implemented, it could potentially improve the depth estimate even further. This would result in improved state estimates and potentially more clear pseudo-annotations.

5.5.3 Using a Mixture Representation

As mentioned in Section 5.3.3 using a single Gaussian to represent the state is problematic. To improve the state estimate, and to produce more clear pseudo-annotations, one could either use a Gaussian mixture to represent the state, or keep the state as a single Gaussian, but instead use a TPMBM filter which estimates a PMBM density, which uses a mixture of Multi Bernoullis to represent the objects.

If the state would be to be replaced by a mixture of Gaussians, both the prediction and update steps in the current implementation would need to be modified accordingly. Similarly, the projection step would need to be modified such that the projection is done to a mixture instead of a single Gaussian.

Using a TPMBM filter would not require changes to the update step, as it can already handle a PMBM as input, and since it outputs a PMBM as well. However, the prediction step would need to be modified. Furthermore, there is no need for the projection step. Instead, one would need to decide on a strategy for keeping the number of mixture components low. A possible solution is to merge similar components, while also setting an upper limit for the number of components kept.

Besides increasing the representational power, both alternatives would simplify TOFS as a whole, since there hopefully would no longer be a need for the association threshold α. The idea is that no trajectories would be merged due to a mixture representation being used, meaning the threshold could be removed. This should result in an increased number of clear pseudo-annotations.

6
Conclusion

In this thesis two new Semi-Supervised algorithms for object detection is presented; *Noisy Student for Object Detection* and *Tracking Objects for Self-supervision (TOFS)*. *Noisy Student for Object Detection* uses the network's detections as pseudo-annotations and relies on noise to improve the model. *TOFS* uses an MOT algorithm to form object trajectories from which pseudo-annotations are extracted in an attempt to make use of the time dependency between frames.

In our experiments, we show that *Noisy Student for Object Detection* improves the APS with 1 percentage point. Similar performance is achieved by increasing the annotated data by $\approx 25\%$, showing the potential of unannotated data.

Furthermore, we show that the position and size estimates of TOFS are superior to the network's output. Although training on them does not result in the expected performance increase, the quality of the pseudo-annotations warrants further investigation of the method.

These insights lead to three main conclusions. Firstly, given the increase in APS in the Noisy Student for Object Detection experiments, there is no denying that unannotated data can be used to improve performance. Given that the performance is similar to that of a model trained on a substantially larger annotated dataset, using unannotated data could reduce the need for manual annotation and thereby cost.

Secondly, the results show that using unclear pseudo-annotations has a large impact on the performance of the algorithms. When no objects are marked as unclear, the Noisy Student for Object Detection performs similar to the baseline. However, when marking a few objects as unclear in each image, the method outperforms the baseline. This indicates that marking detections with a low class-confidence as unclear is an effective way of excluding incorrect detections.

Lastly, TOFS shows potential by improving the size and positional metrics, but for it to increase the APS additional work is needed. A likely cause for the decrease in APS is the number of unclear pseudo-annotations, which could be reduced by tracking objects using another algorithm, for example using a TPMBM filter, or by using more suitable motion and measurement models.
calculated

Bibliography

[1] Facebook registration statement. `https://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm`, 2018. Accessed: 2019-11-26.

[2] Amlaan Bhoi. Monocular depth estimation: A survey, 2019, arXiv:1901.09402.

[3] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[4] Igor Bogoslavskyi and Cyrill Stachniss. Efficient online segmentation for sparse 3d laser scans. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 85(1):41–52, 2017.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[6] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019, arXiv:1909.13719.

[7] Jia Deng, Wei. Dong, Richard Socher, Li-Jia Li, Kai Li, and Li. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[8] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[9] Ángel F García-Fernández, Jason L Williams, Karl Granström, and Lennart Svensson. Poisson multi-Bernoulli mixture filter: direct derivation and implementation. *IEEE Transactions on Aerospace and Electronic Systems*, 54(4):1883–1901, 2018.

[10] Á. F. García-Fernández, L. Svensson, and M. R. Morelande. Multiple target tracking based on sets of trajectories. *IEEE Transactions on Aerospace and Electronic Systems*, 56(3):1685–1707, 2020.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015, arXiv:1512.03385.

[12] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *CoRR*, abs/1610.02242, 2016, arXiv:1610.02242.

[13] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015, arXiv:1506.02640.

[14] Stuart Russel, Peter Norvig, et al. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2013.

[15] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1163–1171, 2016.

[16] Simo Särkkä. *Bayesian filtering and smoothing*, volume 3. Cambridge University Press, 2013.

[17] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence, 2020, arXiv:2001.07685.

[18] Xia Svensson, Granström. Multi-object tracking for automotive systems. University Lecture, 2019.

[19] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NIPS*, 2017.

[20] Ba Tuong Vo. *Random finite sets in multi-object filtering*. PhD thesis, Citeseer, 2008.

[21] Qizhe Xie, Zihang Dai, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation. *CoRR*, abs/1904.12848, 2019, arXiv:1904.12848.

[22] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. 2019, arXiv:1911.04252.

[23] Ángel F. García-Fernández, Lennart Svensson, Jason L. Williams, Yuxuan Xia, and Karl Granström. Trajectory poisson multi-Bernoulli filters, 2020, arXiv:2003.12767.

A

Appendix 1

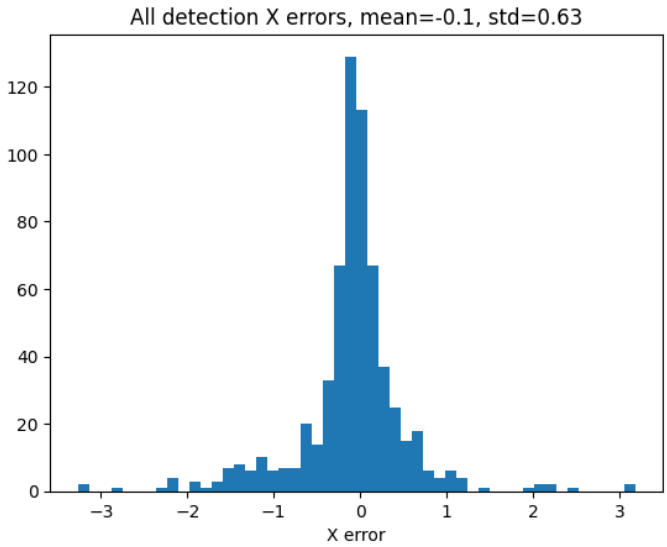

Figure A.1: The X position error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

Figure A.2: The moving average of the X position error for all tracks in the entire annotated sequence from section 4.6

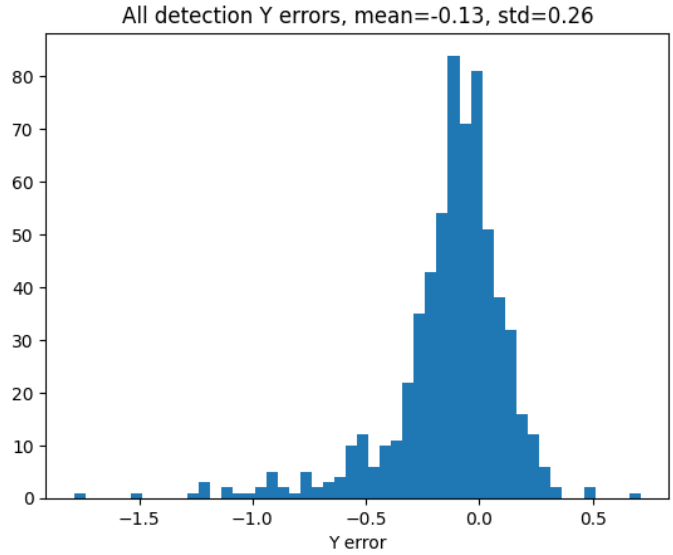

Figure A.3: The Y position error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

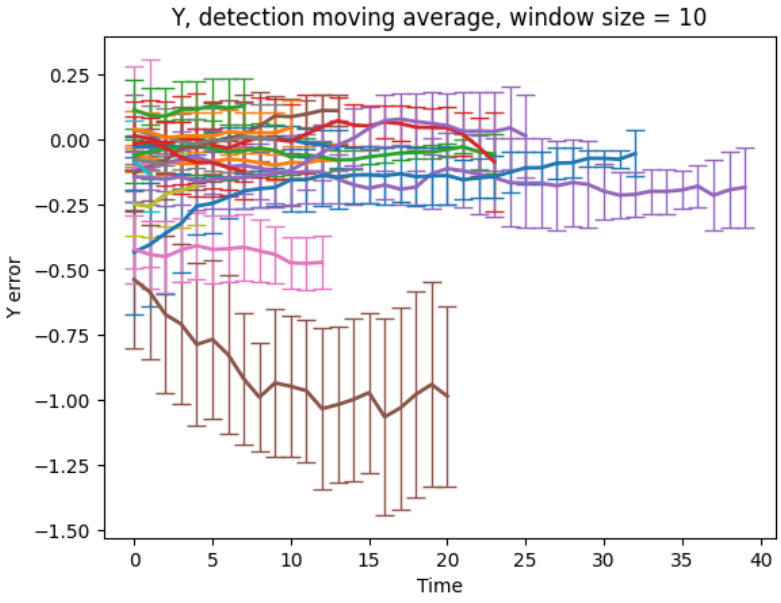

Figure A.4: The moving average of the Y position error for all tracks in the entire annotated sequence from section 4.6

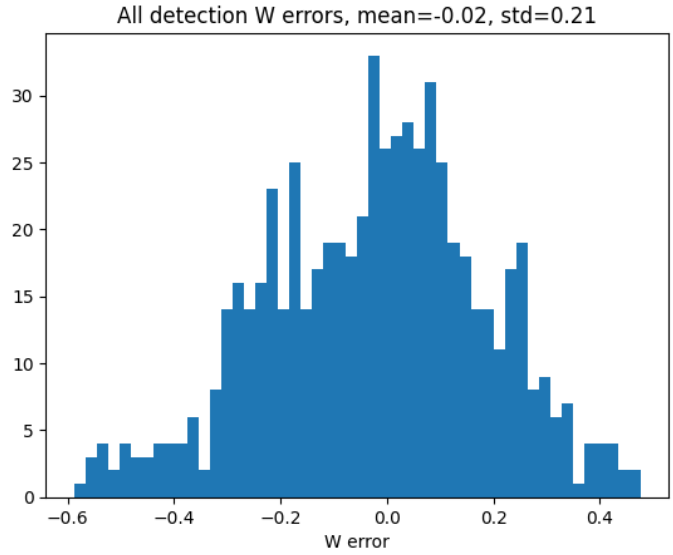

Figure A.5: The width error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

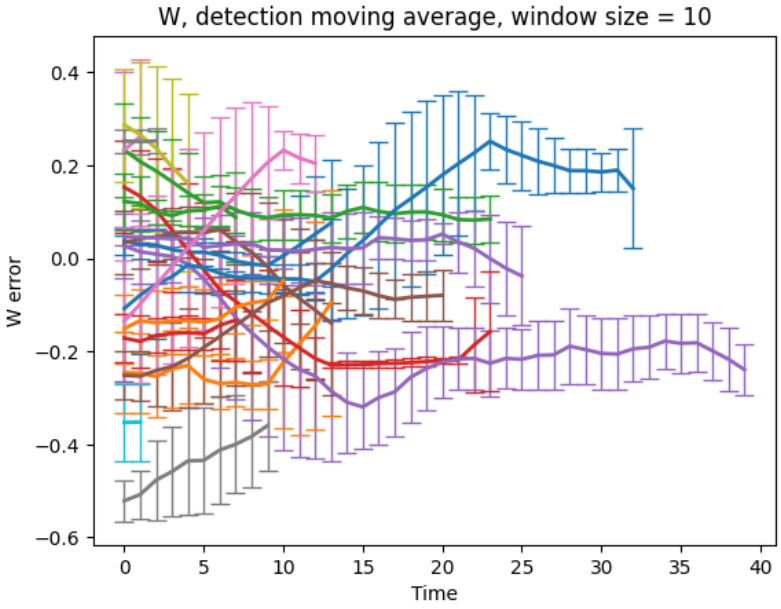

Figure A.6: The moving average of the width error for all tracks in the entire annotated sequence from section 4.6

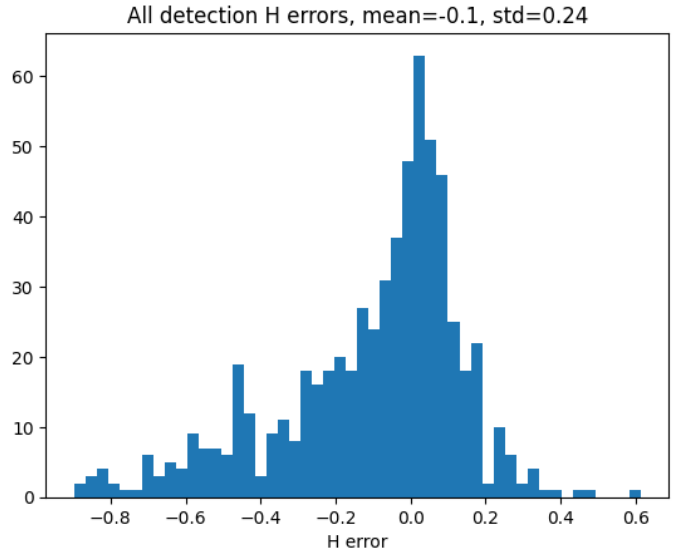

Figure A.7: The height error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

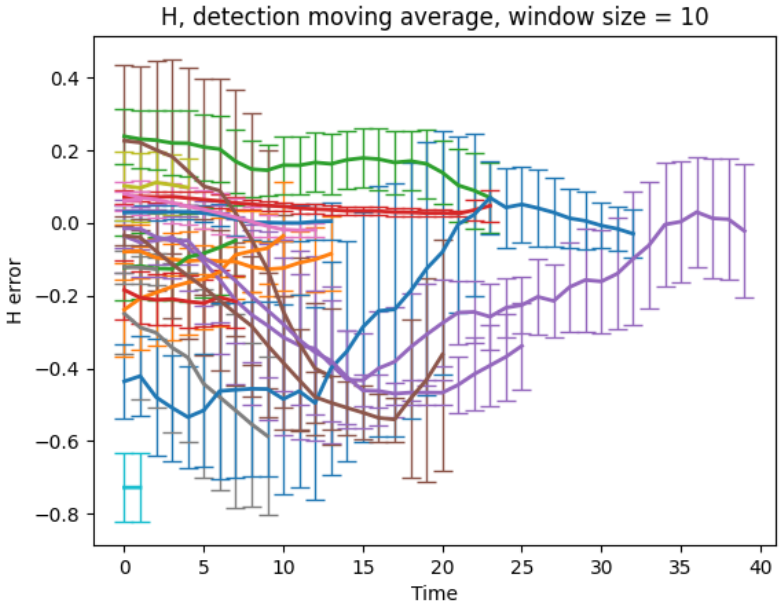

Figure A.8: The moving average of the height error for all tracks in the entire annotated sequence from section 4.6

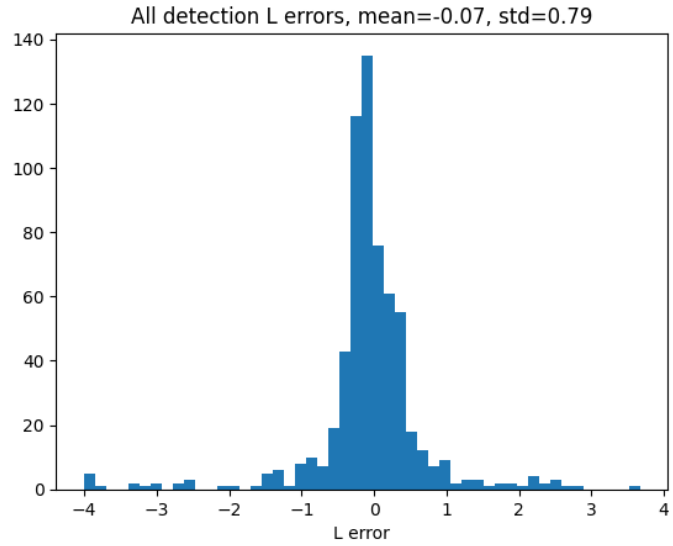

Figure A.9: The length error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

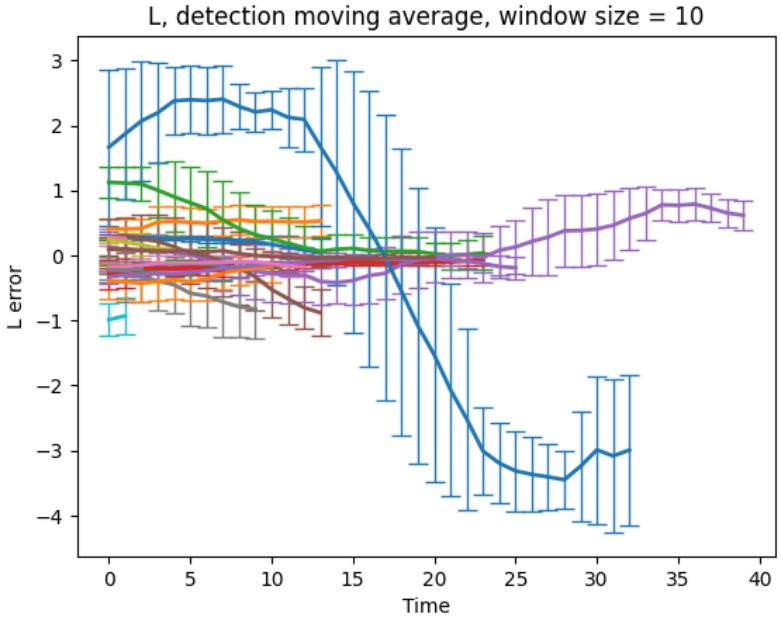

Figure A.10: The moving average of the length error for all tracks in the entire annotated sequence from section 4.6

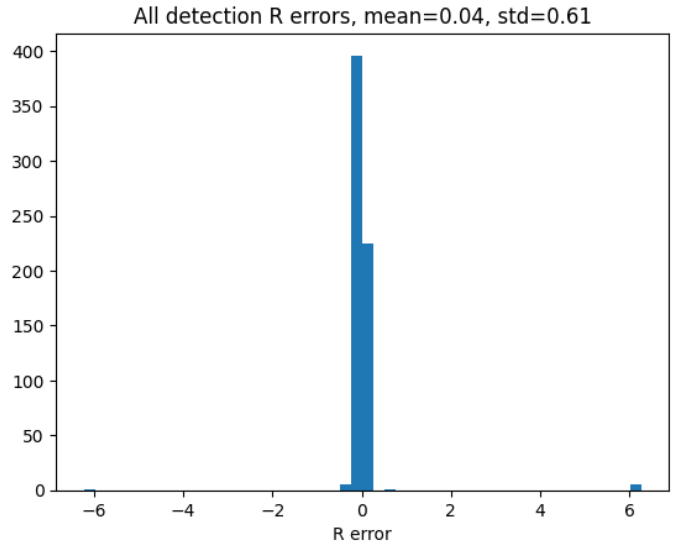

Figure A.11: The length error of the detections in the entire annotated sequence from section 4.6 plotted as a histogram with 50 bins.

Figure A.12: The moving average of the length error for all tracks in the entire annotated sequence from section 4.6

B

Appendix 2

The average error for the z and x measurements seemed to grow linearly with regards to the distance to the object. Therefore the average error was estimated to be a function of the distance to the object. This was done using linear regression.

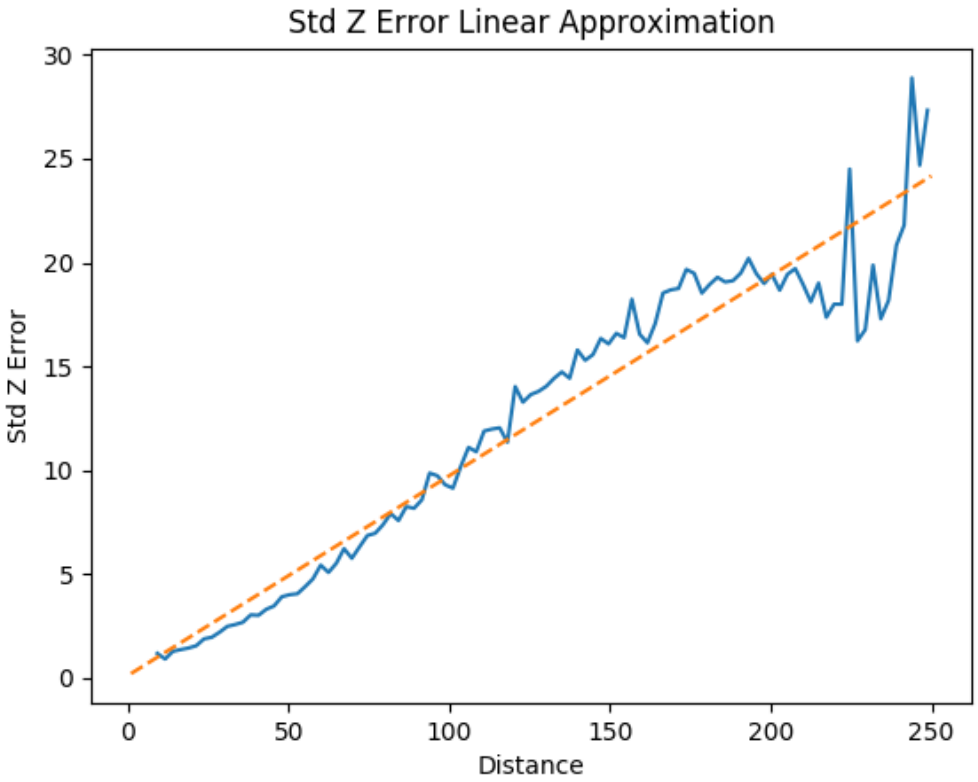

Figure B.1: The Z position error of the detections as a function of the distance to the objects. Is used for the measurement model to calculate the confidence in a given measurement.

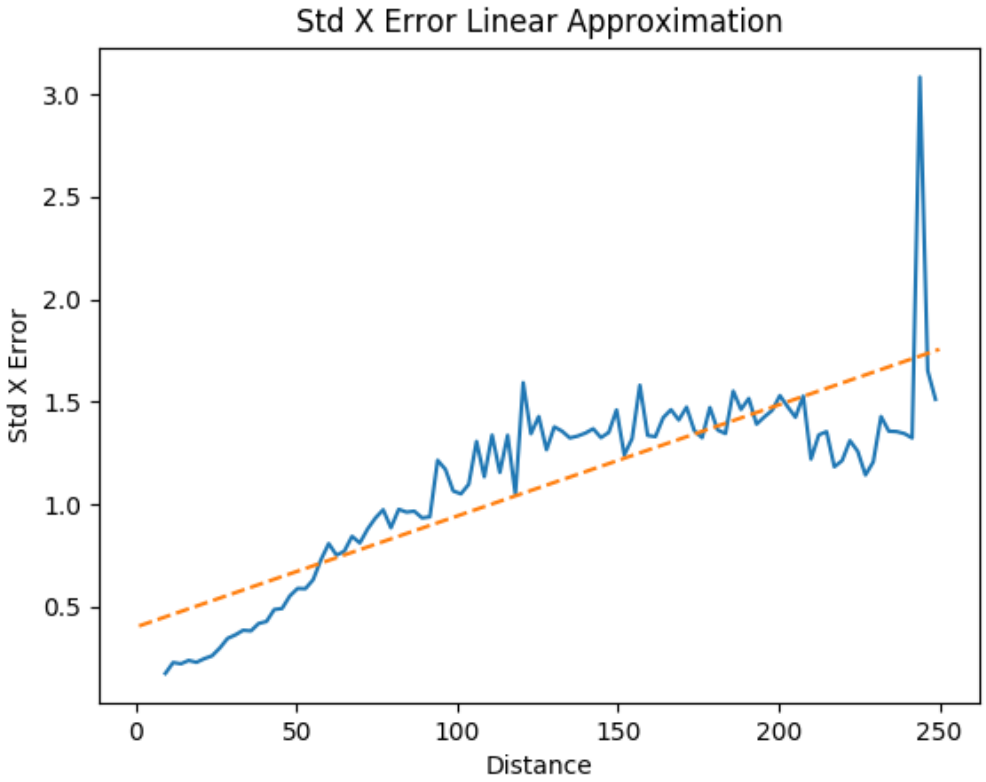

Figure B.2: The X position error of the detections as a function of the distance to the objects. Is used for the measurement model to calculate the confidence in a given measurement.

C

Appendix 3

C.1 Deriving the Chapman-Kolmogorov Equation

To derive the Chapman-Kolmogorov equation, we begin by applying the law of total probability according to the first step in equation C.1. Then, we use the product rule on the resulting integral according to equation C.2. Finally, we use the fact that x_k is conditionally independent on x_{k-1} according to equation C.3. This is commonly known as the Chapman-Kolmogorov equation

$$p(x_k|y_{1:k-1}) = \int p(x_k, x_{k-1}|y_{1:k-1})dx_{k-1} \tag{C.1}$$

$$= \int p(x_k|x_{k-1}, y_{1:k-1}) \cdot p(x_{k-1}|y_{1:k-1})dx_{k-1} \tag{C.2}$$

$$= \int p(x_k|x_{k-1}) \cdot p(x_{k-1}|y_{1:k-1})dx_{k-1}. \tag{C.3}$$