4D Radar-Camera Fusion for Enhanced Point Cloud Clustering

Point-Wise Semantic Annotations Using Pre-trained Image Models

Master's thesis in Complex Adaptive Systems

Anton Carlsson & Aron Enliden

Department of Electrical Engineering

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

Master's thesis 2025

4D Radar-Camera Fusion for Enhanced Point Cloud Clustering

Point-Wise Semantic Annotations Using Pre-trained Image Models

Anton Carlsson
Aron Enliden

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

4D Radar-Camera Fusion for Enhanced Point Cloud Clustering
Point-Wise Semantic Annotations Using Pre-trained Image Models
Anton Carlsson & Aron Enliden

Academic Supervisor: Lars Hammarstrand, Department of Electrical Engineering
Industrial Supervisor: Robert Füllemann, Sensrad AB
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2025

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Sensrad AB
Falkenbergsgatan 3
SE-412 85 Gothenburg

Cover: Two vehicles detected in an image with their corresponding radar clusters.

Typeset in LaTeX
Gothenburg, Sweden 2025

Abstract

Detecting and tracking objects in the environment is a central task in radar perception systems. However, due to the radar's relatively low resolution, limited semantic information, and sensor-specific artifacts such as multipath reflections, a perception framework relying solely on radar data is likely to deliver limited performance. By fusing the radar data with information from a complementary sensor, such as a camera, the additional semantic information can mitigate these issues. Recent contributions to the field of radar-camera fusion focus mainly on deep learning methods, which perform well but require large amounts of annotated data. Collecting and labeling such data can be infeasible because of time or budget constraints.

This thesis instead explores an approach to 4D radar-camera fusion utilizing pretrained image models, with the goal of achieving better tracking performance. The suggested method projects a radar point cloud onto a corresponding image, associates radar points with objects detected in the image, and adds this information to the point cloud. The added information is used to distinguish points that are part of objects from those that are not. It is also used when grouping the point cloud, by suggesting which points likely belong to the same object. Evaluation indicates that the suggested method improves the tracking performance compared to using radar alone. Furthermore, the method shows potential for real-time deployment in runtime evaluations.

Keywords: 4D radar, radar-camera fusion, sensor fusion, point cloud, clustering, object detection, multiple target tracking

Acknowledgements

We would like to express our gratitude to the team at Sensrad AB, in particular to our supervisor Robert Füllemann. His guidance, support, and supervision have been a substantial help during the thesis. We also thank Jacob Klintberg and Tony Gustafsson for their radar expertise and helpful demeanor.

Andreas Åberg served as opponent for the project, and deserves our appreciation for his feedback and proofreading. Lastly we would like to thank Lars Hammarstrand, our examiner, for providing us with advice in pivotal moments.

Anton Carlsson, Gothenburg, June 2025
Aron Enliden, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BEV	Bird's-Eye View
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DoA	Direction of Arrival
FMCW	Frequency Modulated Continuous Wave
GOSPA	Generalized Optimal Sub-Pattern Assignment
GPU	Graphics Processing Unit
IoU	Intersection-over-Union
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MTT	Multiple Target Tracking
NMS	Non-Maximum Suppression
NN	Neural Network
RAM	Random Access Memory
YOLO	You Only Look Once

Contents

List of Figures

List of Tables

1

Introduction

A real-world perceptual system's ability to perceive and interpret its surroundings is fundamental to its operations across a range of applications, including advanced surveillance, autonomous driving, or robotics. This task is reliant on data collected from different sensing modalities, each providing meaningful information about the operating environment. One such modality is millimeter wave (mmWave) radar, which has grown to be a staple in many modern perception systems.

There are several advantages of using radar. Compared to other widely adopted sensors like LiDAR or camera, a radar is more robust in adverse weather while carrying a relatively low cost [1]. Additionally, radars have the ability to measure radial velocity, also known as Doppler. The Doppler information enables easier distinction between static and dynamic objects, which is an important perception capability. The emergence of high resolution 4D radars in recent years have contributed to the increased adoption of radar technology.

Despite this popularity increase, radars still have certain limitations. Compared to LiDAR and camera, they suffer from a measurement sparsity that impedes the performance of common tasks such as object detection. This drawback, considered alongside the information which radar provides that other sensors do not, makes fusion of radar and a complementary sensor an interesting research topic.

In the literature, radar-camera fusion has been gaining traction in recent years. Both radars and cameras are cost-effective [2], which may be a contributing factor. They also complement each other well, as the radar offers great estimations of radial velocity and range, at the cost of poorer resolution. Conversely, the camera contains high-level semantic information but is prone to depth ambiguities. As the 4D radar has emerged alongside recent years' machine learning (ML) boom, most of the literature discussing 4D radar-camera fusion focuses on implementations using neural networks (NNs).

Training neural networks requires vast amounts of quality labeled data, which is expensive and time-consuming to produce. Although public 4D radar datasets have been published [3]–[5], they run the risk of quickly becoming outdated as sensor technology evolves. Changes to a sensor's design can change its' data characteristics and require expensive retraining. With that in mind, this thesis explores another fusion method, utilizing pre-trained image detection models to extract information from the camera domain. This is integrated into an existing radar perception framework,

avoiding the training and use of end-to-end NNs.

1.1 Background

Sensrad AB is a startup company originating in Gothenburg, focusing on 4D imaging radar development and its applications. The company has developed a perception pipeline used for detecting, classifying and tracking objects in the surrounding environment using only radar data. However, due to limitations in the radar modality, the perception pipeline displays some issues. These include:

- A singular object being perceived and tracked as two or more separate objects, also known as object splitting. This is common in for example cars, where the front and the back of the vehicle are tracked as separate objects.
- Two objects being perceived and tracked as one object, also known as object merging. This occurs when two objects are moving close to each other with similar velocities.
- Ghost objects or tracks owing to the multipath effect. The multipath effect occurs when radar transmissions reflect off of multiple surfaces in the surroundings before returning to the receiver antenna. This results in detections being incorrectly perceived as arriving from the original transmission's direction [6]. An illustrative example of this is seen in Figure 1.1.

Sensrad believes that these problems can be mitigated, in part or entirely, by adding camera information to the perception pipeline. The main objective of this thesis is to examine a way of fusing camera and radar information, and determine to what extent this can help with the aforementioned issues.

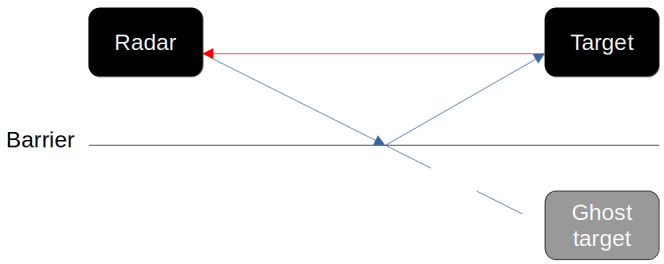


Figure 1.1: Top-down illustration of a multipath phenomenon. Transmitted radar signal in blue, received signal in red.

1.2 Research Questions

The thesis aims to answer the following questions:

- In what ways can camera information be used to improve performance of a 4D radar tracking application?

- What tracking perfomance gains, with regards to the Generalized Optimal Subpattern Assignment (GOSPA) [7] metric, can be achieved when utilizing information from a camera in a 4D radar tracking application?
- How does the runtime performance of the proposed methods scale with image size and scene complexity, and how does this affect the possibility of real-time deployment of the 4D radar tracking system?

1.3 Limitations

To maintain a feasible scope and support a timely completion the thesis, the following limitations will be adhered to:

- Pre-trained NNs will be used to extract information from the camera domain. These networks will not be re-trained or fine-tuned to the task at hand.
- End-to-end NNs for radar and camera fusion will not be considered during the thesis.
- The proposed methods will be integrated into, and evaluated with, Sensrad's existing perception pipeline. The primary focus is to improve the quality of detected objects in the pipeline, but the object tracker itself will not be modified.
- Static objects will not be handled, as they are already filtered out by the current tracker.
- Evaluation will only be carried out on internal datasets, collected with a camera and Sensrad's *Hugin D1* 4D radar. There exist open datasets that include 4D radar measurements, but all of them use radars inferior to the Hugin D1. In combination with Sensrad's existing perception framework, which is tailored to their data representation, evaluation on open datasets is considered to be out of scope.

1.4 Contributions

The contributions of this thesis consist of implementation and evaluation of fusion methods with a 4D radar-camera sensor setup. A method for using radar information to approximate the range of objects found in the camera is proposed, enabling point cloud annotations with higher precision. Additionally, a way of estimating the importance of individual radar points through the camera is implemented, aiding in the subsequent radar tracking. Lastly, the proposed framework undergoes an experimental evaluation. The impact of switching out or disabling pieces of the framework is tested against a baseline implementation.

1.5 Ethical considerations

A perception system is essentially another name for a surveillance system. Whenever the environment is surveilled, it needs to be done with the integrity of people in mind. The system should not be able to discriminate people based on things such as sex or skin color. It is also important that distinguishing features such as faces and license plates on cars are filtered out if any data is stored. The situation which the system surveils also has a big impact on the ethical implications. Consider for example monitoring the traffic on a highway versus the case of monitoring critical infrastructure for hostile drones in a war zone. Miscounting the cars on the highway would create misleading statistics, while failing to detect a hostile drone could affect the lives of thousands of people in a significantly negative way.

Another potential use case with significant ethical considerations is that of tracking software for self-driving cars. It is of utmost importance that autonomous vehicles have a good representation of the surrounding objects, since any errors could have fatal consequences. An interesting extension to the ethics in this case is the question of responsibility in the case of an accident. There are arguments to be made for many parts, such as the owner of the car, the manufacturer, and the engineers who designed the tracking system.

2

Theory

This chapter provides the technical foundation for understanding topics relevant to the thesis. The DBSCAN algorithm and YOLO image models are introduced, followed by an explanation of the tools used to handle images. An overview of the radar is given, as well as an introduction to the research spaces of multi-target tracking and radar-camera fusion.

2.1 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [8] is a clustering algorithm. Starting with an unlabeled set of data points, it groups the points in clusters or labels them as noise. This continues until all points have been assigned a label. The number of clusters does not need to be specified beforehand.

DBSCAN has two tunable parameters, *minPts* and ϵ. The algorithm can then be explained by the following definitions:

- **Core point:** A core point is a point that has at least *minPts* points within a distance of ϵ.
- **Reachability:** A point **R** is reachable from a point **Q** if it is possible to go from **Q** to **R** by only taking steps with a maximum length of ϵ, and only traversing core points.
- **Cluster:** A cluster is a set of points that are all reachable from one another.
- **Noise point:** Any point that is not reachable from any other point is called a noise point.

Figure 2.1 illustrates how a set of points would be clustered for a specific set of parameters. Algorithm 1 (from [9]) shows a heavily abstracted description of the workflow.

Algorithm 1 Abstracted DBSCAN Algorithm

```
Compute neighbors of each point and identify core points      ▷ Identify core points
Join neighboring core points into clusters                    ▷ Assign core points
for all non-core points do
    Add to a neighboring core-point if possible               ▷ Assign border points
    Otherwise, add to noise                                   ▷ Assign noise points
end for
```

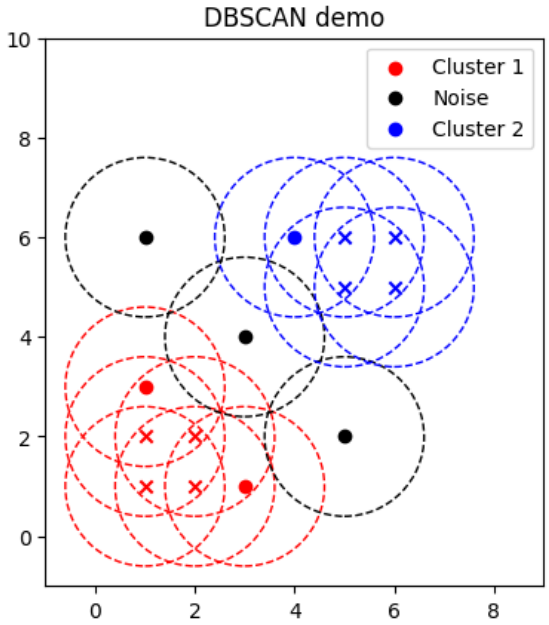


Figure 2.1: Example of DBSCAN with ϵ=*1.6*, *minPts*=*3*. Dashed circles indicate eps-neighborhood, crosses and dots indicate core-points and reachable points respectively.

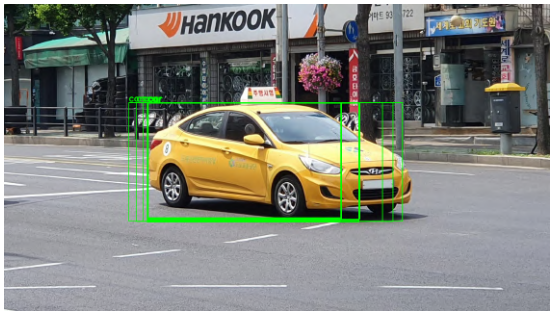
2.2 YOLO Image Models

You Only Look Once (YOLO) [10] is a neural network model for object detection. Since its introduction in 2015, the architecture has been refined through successive versions, with the most recent being YOLOv12 [11]. The first YOLO model, YOLOv1, is a single-stage detector. This means that an image is only run through the network once. This differed from the two-stage detectors that were popular at the time, such as R-CNN [12], Fast R-CNN [13], and FPN [14]. These detectors first run the image through the network to separate the image into regions of interest, and then localize and classify objects in these regions in a second run through the network. The advantage of single-stage detectors is that they are faster than two-stage detectors, since the image only needs to be run through the network once.

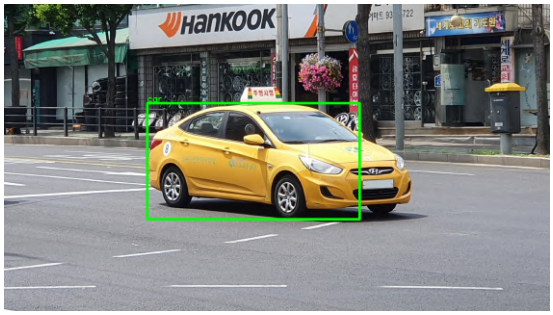
Development of YOLO models has virtually exploded in the latest years [15], with YOLOv6 through YOLOv12 being introduced since 2022. Each new iteration has yielded better accuracy or inference speed. This rapid development can be explained

in part by the ease of access to the models, being available as pre-trained versions in a Python package. The newer YOLO versions also support a wider range of tasks, such as segmentation, pose estimation, and tracking.

The YOLO library models are highly configurable with regards to parameter values, such as the minimum confidence for an object to count as a detection. It is also possible to run built-in tracking on the image objects, using BoT-SORT [16] or ByteTrack [17]. To try and ensure minimal duplicate detections, the thesis will utilize the agnostic non-maximum suppression (NMS) [18]. Non-maximum suppression takes a high-confidence bounding box and discards lower confidence ones of the same class with an overlap. This is done by computing the intersection-over-union (IoU) and comparing to a threshold set by the user. Agnostic NMS extends this idea by performing NMS on all overlapping bounding boxes regardless of class. A visual example of how the IoU threshold affects NMS when detecting a car can be seen in Figure 2.2.



(a) IoU= 1.0. Only full overlaps are removed.



(b) IoU= 0.4.

Figure 2.2: Object detection results with different values of IoU for the NMS step.

2.3 Image Processing

Processing images and extracting information from them is one of the central tasks of this thesis. This section outlines how to convert from 3D coordinates to image coordinates, along with other image processing techniques such as resizing, compensation for camera-induced distortions, and convolutions for enhancing certain features.

2.3.1 World-to-Image Coordinate Conversion

A camera describes the world using a set of 2D points $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$ commonly referred to as image points or pixels [19, Chapter 2]. It is also common to extend the 2D points to homogeneous coordinates, such that $\hat{\boldsymbol{x}} = (\hat{x}, \hat{y}, \hat{z}) \in \mathbb{P}^2$, where $\mathbb{P}^2$ is called the 2D projective space. The reason for this is that depth is ambiguous to the camera, meaning that any 3D points that lie on a line through the focal point will project to the same image point. To convert $\hat{\boldsymbol{x}}$ from homogeneous coordinates

to an image point, the first two coordinates are divided by the third such that $\hat{\boldsymbol{x}}_p = (\hat{x}/\hat{z}, \hat{y}/\hat{z})$. If the third coordinate is zero, the conversion is mathematically undefined and the corresponding image point is said to lie at infinity.

Given a 3D point $\boldsymbol{X} \in \mathbb{R}^3$ in world coordinates, the following information is needed to convert to image points:

- Calibration matrix $\boldsymbol{K}$: A 3×3 matrix describing the intrinsic parameters of the camera, such as focal length and skew. Usually, $\boldsymbol{K}$ is expressed in upper-triangular form:
$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$
where f_x and f_y are independent focal lengths for the x and y dimensions of the sensor, respectively. Skew is represented by s and (c_x, c_y) is the image center in image point coordinates.

- Extrinsic matrix $[\boldsymbol{R}|\boldsymbol{t}]$: Consists of a 3×3 rotation matrix $\boldsymbol{R}$ and a 3×1 translation vector $\boldsymbol{t}$ which describe the camera's position in the world coordinate system.

The intrinsic and extrinsic matrix are often multiplied to form the 3×4 camera matrix
$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}].$$
It is then possible to convert a 3D point $\boldsymbol{X}$ to an image point by using:
$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}] \begin{bmatrix} \boldsymbol{X} \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} \boldsymbol{X} \\ 1 \end{bmatrix}. \tag{2.1}$$
Note that $\boldsymbol{x}$ in (2.1) is given in homogeneous coordinates.

2.3.2 Image Undistortion

Depending on a camera's lens, the image formation introduces distortions that are not present in the real world. The most prominent type is radial distortion [19], which adds curvature to the projection of otherwise straight lines. Radial distortions appear as barrel or pincushion effects, depending on whether the distortion pushes points outward or inward.

To amend the distortions, each pixel can be mapped from the distorted image to a corresponding undistorted image. This is built into many computer vision software libraries, and results in straightening of lines and correction of geometric inconsistencies. The simplest distortion models use low-order polynomials [19], like
$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4) \tag{2.2}$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4), \tag{2.3}$$
where $r_c^2 = x_c^2 + y_c^2$. κ_1 and κ_2 are called radial distortion parameters or distortion coefficients. The distortion coefficients are estimated alongside $\boldsymbol{K}$ during the camera

calibration process. For more accurate approximations, higher order polynomials with additional coefficients can be used.

Undistortion is important when running image inference with image models. Image models are usually trained on undistorted images, since they would otherwise have a dependency on specific distortions for optimal performance. By undistorting the images before running image inference, the image model becomes camera agnostic.

2.3.3 Bilinear interpolation

Bilinear interpolation is a method for estimating values in a 2D grid, which is widely used as a way of scaling images up or down to a desired size. It is an extension of linear interpolation, which works as follows:

Let f be a function defined at least at points $a, b, c \in \mathbb{R}$, with $a \leq b \leq c$. The function values $f(a)$ and $f(c)$ are known, and we want to estimate $f(b)$. With linear interpolation, this is done as a weighted average. The weights are calculated to be inversely proportional to the distance between the known points and the unknown point. Using the abbreviation $w = \frac{b-a}{c-a}$, $f(b)$ is linearly interpolated by the following formula:

$$f(b) = f(a) \cdot (1 - w) + f(c) \cdot w. \tag{2.4}$$

Using Figure 2.3 as a visual aid, bilinear interpolation instead uses a function g and four points $\boldsymbol{Q_1}, \boldsymbol{Q_2}, \boldsymbol{Q_3}, \boldsymbol{Q_4} \in \mathbb{R}^2$. First, two horizontal linear interpolations are done using (2.4) to estimate g at the intermediate points $\boldsymbol{H_1}$ and $\boldsymbol{H_2}$. These points are then used to vertically linearly interpolate the target point X, again with the help of (2.4). Note that the same results are achieved if the linear interpolation is done in the opposite order, first vertically and then horizontally. When using bilinear interpolation for image resizing, the coordinates are specific locations in the array representing the image, and the corresponding function value is the value of the pixel [20].

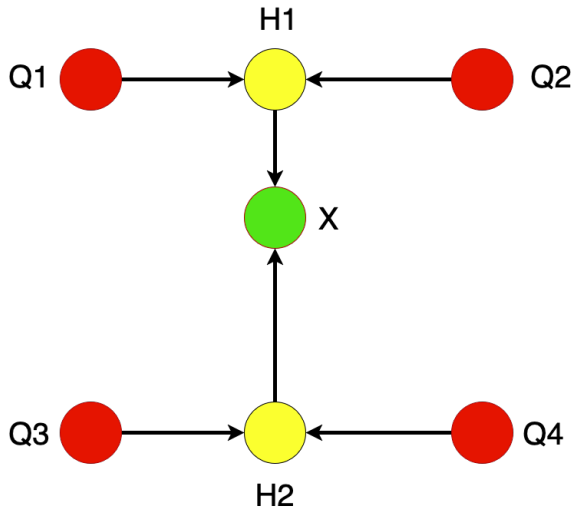


Figure 2.3: A visual example of bilinear interpolation.

2.3.4 Convolutions

Convolution in an image context involves applying a small matrix, known as a kernel or filter, to extract specific features [19, Chapter 3]. This could for example be edges, textures, or patterns. The kernel is moved across each pixel in the image, and the values in the kernel are multiplied with the corresponding overlapping values in the image. The products are then processed to produce a single output. This output is the value of the pixel in the transformed image. In this thesis, dilation followed by convolution with a Gaussian kernel will be used.

A dilation is a convolution with a kernel filled with ones. It extracts the values in the image that overlap the kernel, and then uses a max operation to assign the largest value to the corresponding pixel in the transformed image. This has the effect of enlarging features in the image. See Figure 2.4 for a visual example.

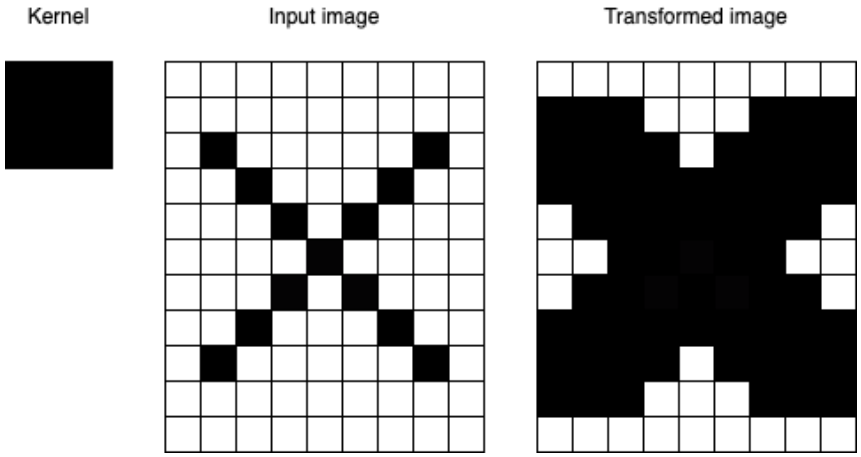


Figure 2.4: Example of dilation. White pixels have value zero, and black pixels have value one.

When convolving with a Gaussian kernel, the values inside the kernel have a Gaussian distribution. After multiplying the values that overlap the kernel, the corresponding pixel in the transformed image is assigned the sum of the products. In practice, this results in a smoothing effect around the edges of objects in the image. Figure 2.5 for a visual example.

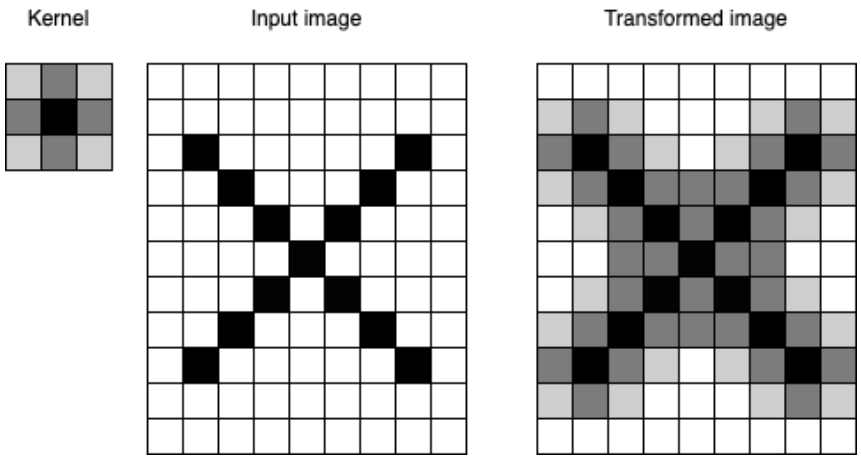


Figure 2.5: Example of convolution with a Gaussian kernel. White pixels have value zero, and darker pixels have values closer to one.

2.4 4D Imaging Radar

A 3D radar provides information in three dimensions: range, azimuth, and Doppler. The 4D radar expands upon this by incorporating elevation as the "fourth" dimension. Converting measurements through trigonometry allows the detections to be represented as cartesian coordinates along with their radial velocity and received power. The *imaging* term refers to the radar resolution, being high enough to construct a sensible world representation.

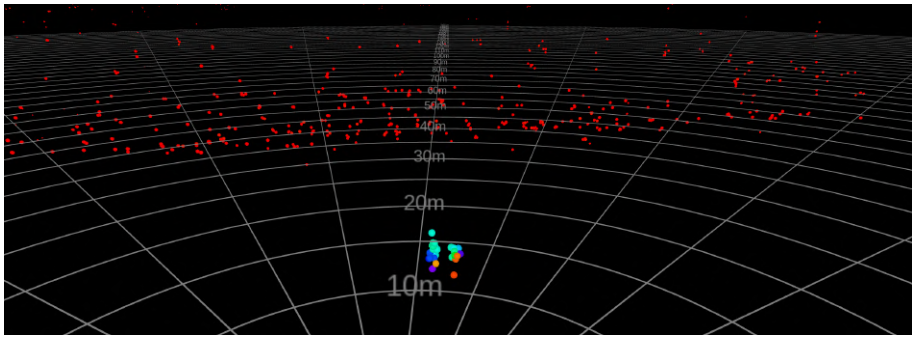
Most commercial 4D radars utilize multiple-input-multiple-output (MIMO) antenna arrays with a waveform called frequency-modulated continuous wave (FMCW) [21]. The 4D radar antenna array consists of transmitting and receiving antennae, referred to as Tx and Rx, configured in a two-dimensional pattern. FMCW radar works by transmitting *chirps*, continuous waves with a linearly increasing frequency. The received signals are mixed with their transmitted counterparts and the Fast Fourier Transform (FFT) [22] is applied, both across singular and multiple chirps. Looking for peaks in this two-dimensional matrix allows for extraction of a detection and its range and Doppler values. What constitutes a detection is usually defined as a peak in signal strength relative to a noise floor. Often times the noise floor is set adaptively, e.g. using the Constant False Alarm Rate (CFAR) [23] algorithm.

The direction of arrival (DoA) in both azimuth and elevation is resolved through the MIMO antenna array [21]. Depending on the DoA, the returning signal contains slight phase shifts for each antenna offset in the horizontal or vertical plane. The

DoA can be resolved from the phase shifts by interpreting it as the extra distance a wave has had to travel to the adjacent antenna, as the antenna distances are known from production.

The knowledge of range, azimuth, and elevation allows for formation of points in 3D space with their associated Doppler value, commonly referred to as a *point cloud*. An example of this can be seen in Figure 2.6, showcasing a scenario where two people are walking away from a radar-camera setup. In the point cloud, both persons are clearly visible along with the slope in the background. Note the varying Doppler values from each person, stemming from reflections off of body parts in different stages of motion.

(a) Camera view.



(b) Radar point cloud view, points colored by Doppler value.

Figure 2.6: Scene depicted by camera with the corresponding radar point cloud.

2.5 Multiple Target Tracking

Target tracking is an essential part of the control flow in autonomous systems. Multiple target tracking (MTT) refers to the simultaneous estimation of a number of targets and each target's individual state, as well as the maintaining of these over time [24]. These estimations are commonly referred to as *tracks*.

In an MTT system, one or several sensors generate multiple detections. Some detections are associated to existing tracks and updated with a filter, while the rest move on to track creation or deletion. All of these tracks are then propagated through time using a predictive model, and new associations are made after the next measurement. Complexity of the association problem grows rapidly as more tracks are initiated, since each detection needs to be associated with a track and the amount of combinations increases factorially. In a radar setting, the data association is further made difficult by radar sparsity and the prevalence of false detections.

Radars have a long history of usage in MTT problems. At first the applications were predominantly used in the aerospace and defence industry, but have since branched out to applications in remote sensing, autonomous vehicles and robotics, as well as biomedical research [25]. Some examples of common radar MTT methods include multiple hypothesis tracking [26] and probability hypothesis density filtering [27], both of which have been widely studied and extended since their introduction.

Radar tracking requires extracting objects from the radar data. One way of doing this is to cluster the point cloud using a clustering algorithm. DBSCAN is especially well suited for this, since it isn't necessary to specify how many clusters to form, and noise points are automatically filtered out. The clusters can then be considered objects and be input to the tracker of choice. Notably, it can be hard to set the DBSCAN parameters appropriately since the radar's detection range is very long, and the distance between points increases with range. This may result in the radar points of two objects becoming one cluster or vice versa. Since clusters are what form the tracking targets, an increase in clustering quality should lead to an increase in tracker performance.

2.5.1 GOSPA

Generalized Optimal Sub-Pattern Assignment (GOSPA) [7] is an MTT evaluation metric. Provided with a ground truth along with tracks, it associates the two and applies two types of penalties: distance-based localization on valid associations, and flat penalties for missed or false tracks.

With the following conditions and definitions:

- Parameters $c \in (0, \infty)$, $\alpha \in (0, 2]$, and $p \in [1, \infty)$
- A metric $d(x, y)$ for any $x, y \in \mathbb{R}^n$ with cut-off metric $d^{(c)}(x, y) = min(d(x, y), c)$
- Π_n is the set of all permutations of $\{1, ..., n\}$ for any $n \in \mathbb{N}$, and any $\pi \in \Pi_n$ is a sequence $\{\pi(1), ..., \pi(n)\}$
- $X = \{x_1, ..., x_{|X|}\}, Y = \{y_1, ..., y_{|Y|}\}$ are finite $X, Y \subset \mathbb{R}^N$, $|X| \leq |Y|$

the GOSPA metric is defined as

$$d_p^{c,\alpha}(X, Y) := \left(\min_{\pi \in \Pi_{|Y|}} \sum_{i=1}^{|X|} d^{(c)}(x_i, y_{\pi(i)})^p + \frac{c^p}{\alpha}(|Y| - |X|) \right)^{\frac{1}{p}} \tag{2.5}$$

If instead $|X| > |Y|$, it is defined as

$$d_p^{(c,\alpha)}(X,Y) := d_p^{(c,\alpha)}(Y,X) \tag{2.6}$$

In practice, it is suggested to use the following conditions and definitions:

- $\alpha = 2$
- γ is a unique assignment set between $\{1, ..., |X|\}$ and $\{1, ..., |Y|\}$
- Γ is the set of all possible assignment sets γ

This leads to the slightly modified definition

$$d_p^{(c,2)}(X,Y) := \left(\min_{\gamma\in\Gamma} \sum_{(i,j)\in\gamma} d(x_i, y_j)^p + \frac{c^p}{2}(|X| - |\gamma|) + \frac{c^p}{2}(|Y| - |\gamma|)\right)^{\frac{1}{p}} \tag{2.7}$$

With equation 2.7, the original GOSPA metric is transformed into containing three distinct terms, each of which is easily interpretable in a tracking context. If we let X denote the set of proposed objects and Y denote the set of true objects, the terms are interpreted as follows:

- **Localization:** For each successful assignment between true and proposed object, a term $d(x_i, y_i)^p$ is added, penalizing localization errors.
- **False detections:** For each proposed object that is not matched to a true object, i.e. all false detections, a flat penalty of $\frac{c^p}{2}$ is added by the term $\frac{c^p}{2}(|X| - |\gamma|)$.
- **Missed detections:** For each true object that is not matched to a proposed object, i.e. all missed detections, a flat penalty of $\frac{c^p}{2}$ is added by the term $\frac{c^p}{2}(|Y| - |\gamma|)$.

The metric $d(x, y)$ used for calculating the localization error can differ depending on which type of tracking algorithm is used.

2.6 Radar-Camera Fusion

Radar-camera fusion is an area that has seen significant contributions over the later years, as sensor quality and processing power have increased. There are several different design choices that have to be made when constructing a radar-camera fusion system, the first of which being when to fuse the data. In broad terms, there are three different options [1]:

- **Early fusion:** Sometimes called data-level fusion, raw or pre-processed data from both sensors is fused and then further processed depending on the task at hand. This leads to strongly fused data and minimal information loss, at the expense of increased data volume and computational cost.
- **Middle fusion:** Also known as feature-level fusion, features extracted from both sensors are fused. This type of fusion has seen great progress in the latest

years due to the prevalence of deep learning, which is well suited to these types of tasks. However, it necessitates a lot of data to train on, which is resource intensive to collect and annotate.

- **Late fusion:** Often referred to as decision-level or object-level fusion, each sensor is allowed to take decisions on its own, and their output is then combined to create the fused decision. In a tracking context this means that each sensor outputs a set of objects detected in the environment, which are then correlated to create a set of fused objects. This approach is very flexible and in theory sensor invariant, but fails to take all available information into account and thus could lead to subpar results.

Due to the relative recency of the 4D radar, the field of 4D radar and camera fusion is relatively sparse compared to that of e.g. LiDAR and camera fusion. One of the latest additions is [28], which utilized middle fusion to achieve state of the art performance on the View of Delft (VoD) [3] and TJ4DRadSet [4] upon release. VoD and TJ4DRadSet are the standard datasets when benchmarking fusion methods for object detection and tracking. To combat the sparsity of radar point clouds, [28] aggregates several subsequent radar frames and uses them to extract bird's-eye view (BEV) 4D radar features. These features are then fused with features extracted from multi-view images, using a complex neural network consisting of both encoders and transformers.

Another interesting approach is [29], which uses a hybrid fusion approach where the point cloud is fused with features extracted from the image. A pre-trained object detection model is run on the image, and the corresponding point cloud is then projected onto the image. Each radar point that projects into a bounding box of an object detected in the image is then correlated with the bounding box ID and the class of the detected object. For points that fall in the intersection between two or more bounding boxes, several bounding box ID:s and classes are assigned. When the point cloud is clustered, only points with the same bounding box ID are allowed to be clustered together. Additionally, different clustering parameters are used depending on the camera class of the points. The final step is an iterative cluster merging process, performed to merge clusters that belong to the same camera object.

Lastly, an example of a late fusion approach can be found in [30], which uses the pretrained networks PointPillars [31] and YOLOX [32] for object detection in the point cloud and image respectively. To solve the dimension discrepancy between 3D radar objects and 2D camera objects, a method which uses object pixel height along with statistical height of objects of different types is presented to convert camera objects to 3D. Camera and radar objects are then associated into fused objects, which are in turn associated with historical tracks using a cascaded association structure. When comparing the tracking performance of the suggested solution with other common fusion techniques on VoD, the suggested solution achieves similar or better results.

3

Sensor platform

This chapter gives an overview of the hardware and software setups that have been utilized in this thesis. Section 3.1 explains the data collection platform and the sensors that have been used to collect the data for evaluation. Following this, Section 3.2 outlines the inner workings of the radar perception pipeline.

3.1 Hardware

The data collection platform consists of three parts: a Sensrad Hugin D1 4D radar, an Oak-1 W camera, and a LiDAR from Ouster. With the help of Robot Operating System 2 (ROS2) [33], timestamped data is collected from all three sensors simultaneously. This data is labeled with an internal tool, using image data in combination with the LiDAR. The radar has a footprint of approximately 15x15 cm and can be seen together with the camera in Figure 3.1. The update rates for the radar and the camera are 17 Hz and 60 Hz, respectively. Remaining Hugin radar specifications can be found in appendix A.1.

Figure 3.1: Hugin D1 radar with an Oak-1 W camera mounted on top.

3.2 Software

The current Sensrad perception pipeline only utilizes radar data to detect and track objects in the environment. Figure 3.2 shows a simplified schematic of the pipeline, which has 4 major modules connected in a linear fashion, with one recurrent connection. The input is a raw point cloud, and the result of a run through the pipeline is a set of tracked objects. The modules have the following functions:

- **Preprocessing**: The raw point cloud is preprocessed using proprietary algorithms. An estimation of the ground plane is also calculated. The ground plane estimation and point cloud are sent onward.
- **Screening**: For each point in the point cloud, a likelihood that the point is part of an object is calculated. The likelihood depends on among other things Doppler, radar intensity, distance to ground plane, and location in relation to historical tracks. All points with a likelihood under a threshold are thrown out, and the rest are sent to clustering.
- **Clustering**: To extract objects from the remaining set of points, the DBSCAN algorithm is used. The clustering is done across the range, azimuth, elevation, and Doppler dimensions. Resulting clusters are forwarded to tracking.
- **Tracking**: In the tracking module, a continuous set of tracked objects is maintained. Each iteration, objects detected in the current radar frame are associated with historical tracks or initiated as new tracks. The newly updated set of tracked objects are then both output and fed back to to screening module as a reference to locations where objects are more likely to appear.

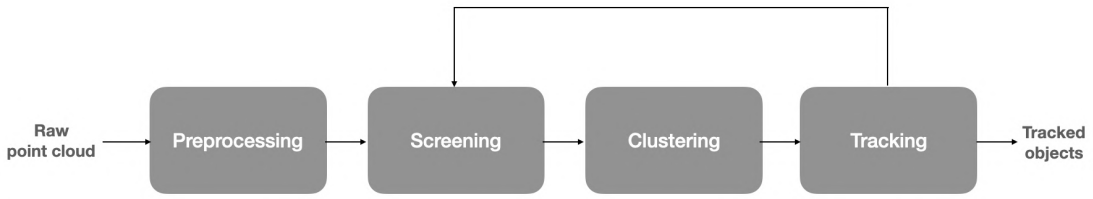


Figure 3.2: Schematic of Sensrad's radar perception pipeline.

4

Methods

This chapter explains the methods implemented during the thesis, and how they have been evaluated. First, the fused tracking pipeline is outlined to give the reader a context of the modifications made. Following that, the changes are explained in more detail. Lastly, the available data and how it has been used in evaluation is presented.

4.1 Fused Pipeline Overview

Considering the radar tracking pipeline described in Section 3.2, Figure 4.1 provides an overview of the new fused tracking pipeline, with differences highlighted. The fused pipeline accepts two inputs: a raw point cloud and a camera image, both depicting the same scene. The fused pipeline introduces two new modules:

- **Image detection:** A camera image is input to the image detection module, and is tasked with extracting information from it. The information is forwarded as bounding boxes or semantic masks.
- **Point cloud annotation:** The information gathered from the image is fused with the point cloud via a point-wise annotation scheme. This new information is used in both the screening and the clustering step.

The changes made in the screening and clustering module are described in more detail in Sections 4.4 and 4.5.

The modified pipeline requires a new pair of unseen inputs each iteration. Since the radar has a slower update frequency, the pipeline is updated every time a new radar point cloud is received. This means that the image may be slightly older than the point cloud. The time discrepancy is on the order of milliseconds, which is considered negligible in the context of the traffic scenarios which are used for evaluation.

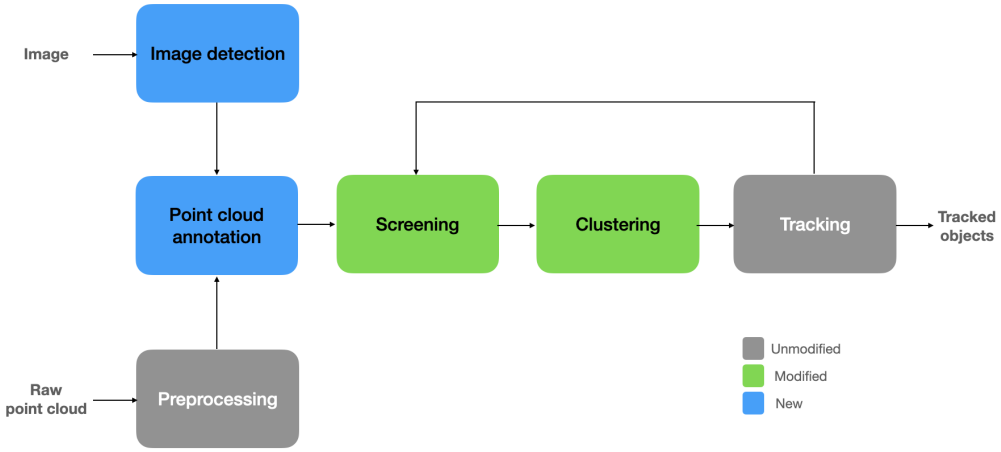


Figure 4.1: Fused tracking pipeline.

4.2 Image Detection

To gather information from the image, a pretrained NN model is utilized. Depending on the point cloud annotation mode, either a model for object detection or a model for instance segmentation is used. Before the image is input to the image model, it is undistorted using the camera matrix and camera distortion coefficients. The output of the image detection module is either a set of bounding boxes or a set of semantic masks, each belonging to a separate object detected in the image.

The chosen image inference model is YOLOv11n, the reason being that it is easily available, has a low inference time, and supports NMS. There is however no specific dependence on any particular image model, meaning that with minor changes it can be substituted.

4.3 Point Cloud Annotation

The point cloud annotation module is where the information gathered from the camera image is fused with the point cloud. This is done on a per-point basis, by assigning two new values to each point in the point cloud:

- **Camera confidence:** A floating point value $0 \leq c \leq 1$ that describes how likely the camera thinks it is that the point is part of an object.
- **Camera cluster index:** An integer $i \geq -1$ indicating which object the camera thinks the point is part of. A camera cluster greater than zero indicates which camera object the point is part of, -1 means that the point could be part of several objects, and 0 means that the point is not part of an object.

Three different ways of assigning the new attributes have been implemented. They differ in what type of camera detections are used, and how the camera confidences

are modeled. The specific implementation differences are found in Sections 4.3.2 and 4.3.3, with the general outline being:

1. **Initialization:** Camera confidences and camera cluster indices are set to zero. A set of camera detections (bounding boxes or segmentation masks) are extracted from the image.

2. **Projection:** Radar points are projected onto the image using the camera matrix and the relative positioning of the radar and the camera.

3. **Association:** Camera detections are iterated and associated with a set of radar points. Optionally, the associated points are filtered based on their range. Remaining points are given a camera confidence and cluster index. The camera cluster indices are set by enumerating the camera detections. Any point associated with several detections is assigned a camera cluster index of -1.

This procedure highlights the importance of agnostic NMS in the image model. Without agnostic NMS, an object with overlapping bounding boxes of different classes would be assigned a camera cluster index of -1. This would prevent the subsequent clustering from utilizing the camera information to its full extent. The camera confidence value is also assigned differently depending on the annotation mode.

4.3.1 Depth Estimation

The camera has a large inherent depth ambiguity. In the annotation methods utilizing bounding boxes, it is not uncommon for the bounding box to include parts of the background along with the located object. Assigning a camera confidence and camera cluster index to such a background point is suboptimal because it leads to feeding irrelevant points to the tracker. The radar has a very good range estimation, which is leveraged here to combat this issue. This is done by filtering all points associated with an image detection.

Under nominal operating conditions, there should be an object of interest inside of each image detection. Reflections from such an object would therefore be closely spaced in the range dimension and, if the target is moving, usually numerous. These observations can be leveraged to filter out background points by identifying dense range clusters and only applying the camera annotations to one of these. The clustering in this case is done only in the range dimension, for all points projected to the same camera object. DBSCAN is used with $\epsilon = 1.5$ m and $minPts = 3$.

The intuitive solution is to extract the largest range cluster, but there are situations where this approach fails. As an example, imagine a pedestrian close to the camera with a large truck further away in the background. In the camera image, the pedestrian is easily detected and partly occludes the truck. When projecting the point cloud, points belonging to the truck may be projected into the pedestrian's bounding box, due to radar or bounding box inaccuracies. If enough "truck points"

are present inside the pedestrian bounding box, the largest cluster would not belong to the pedestrian detection, leading to erroneous annotations.

To amend this, the range clustering approach is adjusted to prioritize closer clusters. In practice, this works by first finding the largest cluster, then searching closer to the camera for clusters of similar size. If such a cluster is found, it is used instead of the largest. If no cluster can be formed, all points inside the image detection are used. Figure 4.2 displays an example of how a set of points would be clustered, and Algorithm 2 formalizes the method.

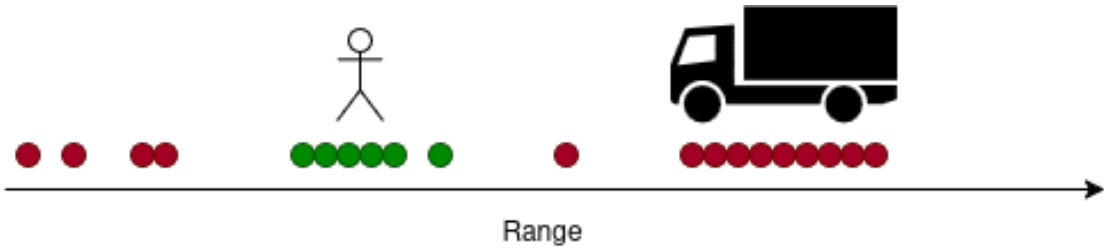


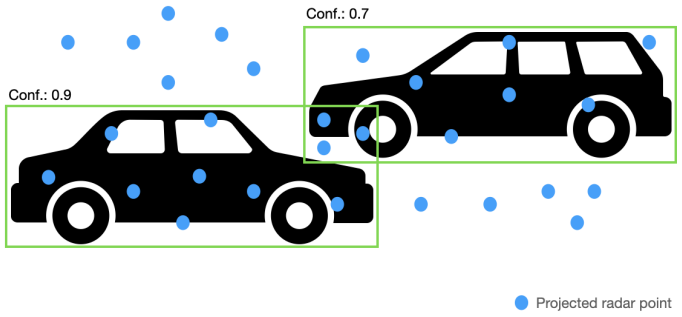
Figure 4.2: Range clustering example for a pedestrian image detection. The green cluster is chosen due to being closer and of similar size to the farthest.

Algorithm 2 Depth estimation inside image detection

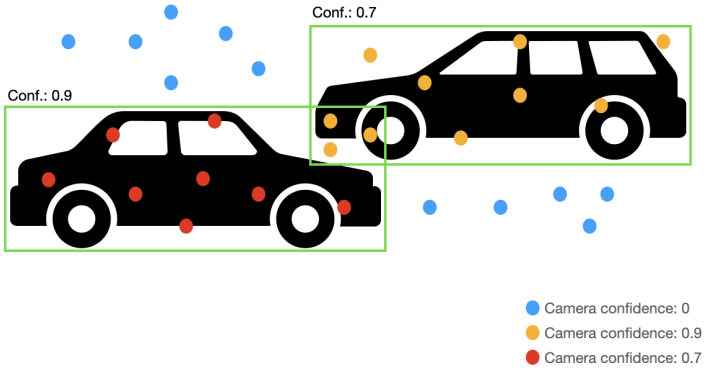
Require: points
Ensure: filtered_points
clusters ← DBSCAN(points.range, eps = 1.5, minPts = 3)
if length(*clusters*) > 0 **then**
    biggest_cluster ← max(clusters by num_points)
    filtered_points ← biggest_cluster.points
    for all cluster **closer than** biggest_cluster **do**
        if cluster.num_points > 0.4 × biggest_cluster.num_points **then**
            filtered_points ← cluster.points
        end if
    end for
else
    filtered_points ← points
end if
return filtered_points

4.3.2 Basic and Segmentation Annotation

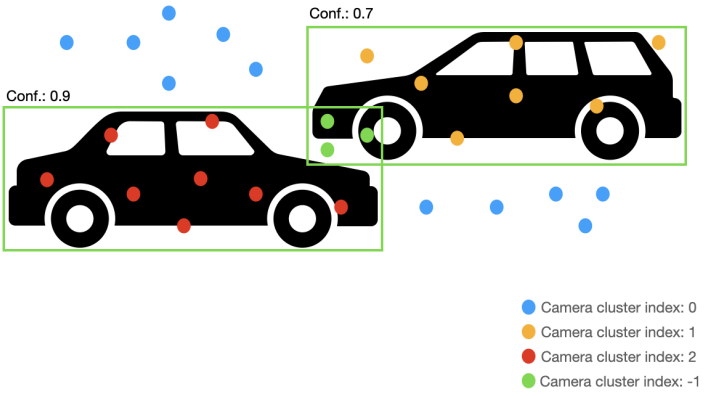
The Basic and Segmentation annotation modes differ in what type of image model is used, and how points are associated with the image detections. The Basic annotation mode uses an object detection model, which means that the image detections are bounding boxes. The association between image detections and radar points is done by associating the bounding box to all radar points that project inside of it. An illustrative example can be seen in Figure 4.3.



(a) Example point cloud projected onto image.



(b) Camera confidences.



(c) Camera cluster indices.

Figure 4.3: Basic annotation mode explained.

The Segmentation annotation modes instead uses and instance segmentation image model, meaning that the image detections are semantic masks. Before association, the masks are preprocessed using a dilation to slightly enlargen the masks, followed by convolution with a Gaussian kernel to smooth out the edges. An example of the preprocessing can be seen in Figure 4.4. The reason for the preprocessing is that radar points sometimes project slightly outside of the camera target, due to radar noise or imperfect extrinsic calibration. This is a bigger problem with semantic masks than bounding boxes, since bounding boxes naturally covers more area than

semantic masks. The association between preprocessed masks and radar points is similarly done by associating masks with points that projects onto them.

(a) Image scene with two people.

(b) Masks from segmentation model.

(c) Masks post dilation and smoothing.

Figure 4.4: Cropped images showing the difference between initial segmentation masks and the enlarged and smoothed ones used for annotation.

The general method for the Basic and Segmentation annotation modes is formalized in Algorithm 3. The camera confidence for each point is chosen as the maximum of the current confidence of the point, the confidence of the image detection, and a threshold. The reason for this is to avoid overwriting previously set camera confidences in case of overlapping image detections. Also note that for both annotation modes, the camera confidence is modeled as uniform over the entire bounding box or semantic mask.

4.3.3 Gaussian Annotation

The Gaussian Annotation mode uses an object detection image model, and models the camera confidence as a multivariate Gaussian surface. For each bounding box, the confidence is modeled as a multivariate Gaussian surface with a maximum value of 1 in the center of the bounding box. The horizontal and vertical sides both have lengths of ± 2 standard deviations from the detection center. All points with a Gaussian confidence value greater than 0.1 are assigned a camera confidence and cluster index. The value 0.1 was chosen empirically and limits the extent of the Gaussian approximation. The full annotation method is shown in Algorithm 4.

Algorithm 3 Pointcloud Annotation (Basic/Segmentation)

Require: image_detections, pointcloud, camera_info
Ensure: pointcloud

```
for each point in pointcloud do
    point.confidence ← 0
    point.camera_cluster_id ← 0
end for
for each detection in image_detections do
    associated_points ← Associate(pointcloud, detection, camera_info)
    filtered_range_points ← DepthEstimation(associated_points)
    for all point in filtered_range_points do
        point.confidence ← max(point.confidence, detection.confidence, threshold)
        if point.camera_cluster_id = 0 then
            point.camera_cluster_id ← detection.id
        else
            point.camera_cluster_id ← -1
        end if
    end for
end for
return pointcloud
```

Algorithm 4 Gaussian Annotation

Require: bounding_boxes, pointcloud, camera_info
Ensure: pointcloud

```
for all point in pointcloud do
    point.confidence ← 0
    point.camera_cluster_id ← 0
end for
for all box in bounding_boxes do
    gaussian ← BBoxTo2DGaussian(box)
    filtered_range_points ← DepthEstimation(points_in_bbox)
    for all point in filtered_range_points do
        gauss_conf ← gaussian(point)
        if gauss_conf > 0.1 then
            point.confidence ← max(point.confidence, gauss_conf)
            if point.camera_cluster_id = 0 then
                point.camera_cluster_id ← box.id
            else
                point.camera_cluster_id ← -1
            end if
        end if
    end for
end for
return pointcloud
```

4.4 Screening

The screening module is used to determine which radar points to pass on to the clustering module. Sending points of high quality (i.e. belonging to an object) to the clustering aids in creating meaningful clusters to forward to the tracker. The pre-existing (radar) screening is a Doppler filter, meaning static points are removed. An object likelihood between 0 and 1 is calculated for each point based on its ego-motion compensated Doppler. The object likelihood is then compared to different noise indicators, for example the point's likelihood of being part of the estimated ground plane. As discussed in Section 4.1, the screening also contains a feedback step from the tracker module. Points in the general area of a predicted, mature track get assigned a higher object likelihood. When choosing which points pass the screening, a static threshold is used, meaning that all points with object likelihood below the threshold are filtered out.

This pre-existing screening is changed to account for the camera confidence point cloud annotations. Two methods are proposed:

- **Max operation screening:** For each point, the object likelihood is calculated through the relative Doppler. Additionally, the assigned camera confidence is extracted and the likelihood of belonging to an object is set to the maximum of the two. This is equivalent to trusting each sensor maximally, and results in forwarding more points to the clustering. The reasoning behind this is that the camera may feed an increased amount of relevant points to the remaining pipeline, while also preserving the original functionality.
- **Weighted screening:** The object likelihood and camera confidence are computed for a point. They are then individually multiplied with a set of predetermined weights and finally added together. Adjusting the weights allows for feeding the pipeline with points predominantly from Doppler or the camera, or a combination of the two. As multipath is a problem stemming from the radar, this approach could help mitigate the ghost track issue. The weights can intuitively be interpreted as a trust factor, signifying how much we trust that the measurements from the different sensors are correct. By for example setting both weights to 0.5, we require points to be marked as interesting both in the camera and the radar to pass the screening module.

4.5 Clustering

In the modified clustering, the suggested camera cluster indices are incorporated. The general idea is that each cluster is only allowed to contain points from one unique, positive camera index. This means a cluster is free to grow using points labeled 0 and -1, as well as points with one unique camera index. The reasoning behind this is that radar noise sometimes makes points project outside of an image detection, but they might still belong to that object. On the same note, the image model sometimes misses objects. By only allowing one unique positive camera index,

it is ensured that distinct objects in the camera domain do not get clustered together, which should help limit object merging. To achieve this, the following adjustments are implemented on a point-level when computing the neighbors in the modified DBSCAN:

- If both points have equal camera indices > 0, ϵ is extended by a factor of 5, empirically chosen. This creates larger clusters and combats object splitting.
- If both points have differing camera indices > 0, ϵ is set to 0 which in practice disallows clustering. This is done to combat object merging.
- If both points have camera indices ≤ 0, meaning that they are considered ambiguous or unannotated by the camera, they are treated as regular radar points and the normal ϵ value is used. This ensures that clusters can be created when there is no information from the camera.

4.6 Data and Evaluation

The methods presented in this chapter have been evaluated on data collected with the data collection platform presented in Section 3.1, using the Hugin D1 radar.

4.6.1 Quantitative Evaluation

In the quantitative evaluation data the ground truth is annotated on an object basis, meaning that each radar frame has a corresponding set of objects that should be detected. The data contains six scenarios, of which an overview is given in Table 4.1. It should be noted that the data does not contain ground truths for objects that remain static throughout the scenarios, and adverse weather conditions are not depicted. To combat the second issue, adverse weather will be simulated by deprecating the performance of the image model. This is done by removing each image detection with a predetermined probability.

Table 4.1: An overview of the scenarios used for evaluation.

Scenario name	Description	Issues present with original pipeline
1m separation	Two people walk side by side 1m from each other, first away from the radar and then they turn back.	Objects merging
2m separation	Two people walk side by side 2m from each other, first away from the radar and then they turn back.	Objects merging
x walk 1	Two people walk in an X-shape.	Lost tracks on temporary occlusion
x walk 2	Two people walk in an X-shape.	Lost tracks on temporary occlusion
straight solo walk 1	One person walks away from the radar. At 50m distance, they stop and stand still for 10s before walking back	None
complex scenario 1	Real world scenario depicting a road and sidewalk.	Multipath, object splitting

The evaluation will be done using the GOSPA metric, both by averaging over an entire scenario and also by inspecting the individual GOSPA constituents over the course of a scenario. This allows to see if there is a general improvement, signaled

by a decreased average GOSPA compared to the original tracking pipeline. It also allows a deeper analysis of why the improvement occurs, by separating GOSPA into its constituents (localization, missed, and false) and visualizing them as a time series.

4.6.2 Qualitative Evaluation

A qualitative evaluation will also be performed. All but one of the scenarios represented in the quantitative evaluation data contain four or fewer dynamic objects. This makes the ground truth easier to label and helps assess the basic functionality. Still, the performance on the quantitative scenarios is not necessarily indicative of the performance in a more cluttered environment. Traffic data showcasing more complex and congested scenarios has been collected by Sensrad, but is missing ground truth labeling. Labeling this data with 3D bounding boxes is out of scope for this thesis. The unlabeled data can still be used in qualitative evaluation however.

The qualitative evaluation focuses on data showcasing the problems studied in the thesis, those being object splitting and merging as well as multipath objects. Two cluttered scenarios are chosen to examine this. The projection of radar tracks onto the image allows for easy visual inspection of the tracks in relation to the image's semantic content. This evaluation will be comparing the radar pipeline to the fusion modes, as well as highlighting pros and cons among some of the fusion implementations.

4.6.3 Runtime Evaluation

To investigate the feasibility of running the implemented solution in a real-time system, an evaluation of the runtime will be done. Data for runtime evaluation will be collected by running a subset of the pipeline implementations on Sensrad's internal server. The timings of these runs will be compared relative to the timings of the radar pipeline, which is able to run in real-time. Specifications of Sensrad's server environment can be seen in Table 4.2. Besides comparing the runtime of the entire pipeline, an analysis of which parts of the fused pipelines contributes the most to the runtime will also be done.

When evaluating runtime, there is no limitation to only use scenarios with annotated ground truth. Thus, a more extensive set of in total 24 scenarios will be included in the timing evaluation. This ensures that a bigger variety of scenarios are represented, as well as allowing for investigating how the complexity of a scenario affects the runtime of the pipeline.

Table 4.2: Specifications of Sensrad's server environment

Component	Description
Operating system	Ubuntu 24.04
CPU	AMD Ryzen 9 5950X 16-core
GPU	NVIDIA GeForce RTX 3090 24 GB
RAM	32 GB

5

Results and Discussion

This chapter presents the results obtained from the study. It begins with a quantitative evaluation of the various implementations, using GOSPA values as the primary metric. This is followed by a qualitative analysis of performance in more complex traffic scenarios. The qualitative analysis is presented in the form of camera snapshots with projected object tracks. Finally, a runtime evaluation is presented.

5.1 Quantitative Evaluation

In the following section, results for different versions of the tracking pipeline are presented and discussed. The results are presented as bar graphs showing percentage change in GOSPA compared to the radar pipeline, calculated as

$$\Delta_{\%} = \frac{GOSPA_{radar} - GOSPA_{fusion}}{GOSPA_{radar}}. \tag{5.1}$$

A positive value means improvement in GOSPA. The right-most values in each figure shows the average percentage difference over all evaluated scenarios. First, a base case is described and presented. This is followed by a series of modifications of the base case, with one or several parameters changed. Unless further specified, chosen parameters match the base case. Tables with absolute GOSPA values can be found in appendix B.1.

5.1.1 Base case

As a reference for further comparison, the base case is defined as using YOLO11n as the image model, with depth estimation, max operation screening, and the improved clustering. The input images are full HD, 1920 × 1080 pixels. The results are presented in Figure 5.1. All three annotation modes perform better than the radar pipeline, with an average improvement in GOSPA of between 12% and 18%. The greatest improvement is seen with Segmentation annotation. The only scenario that shows worse results than the radar pipeline is Basic annotation when evaluated on *complex scenario 1*.

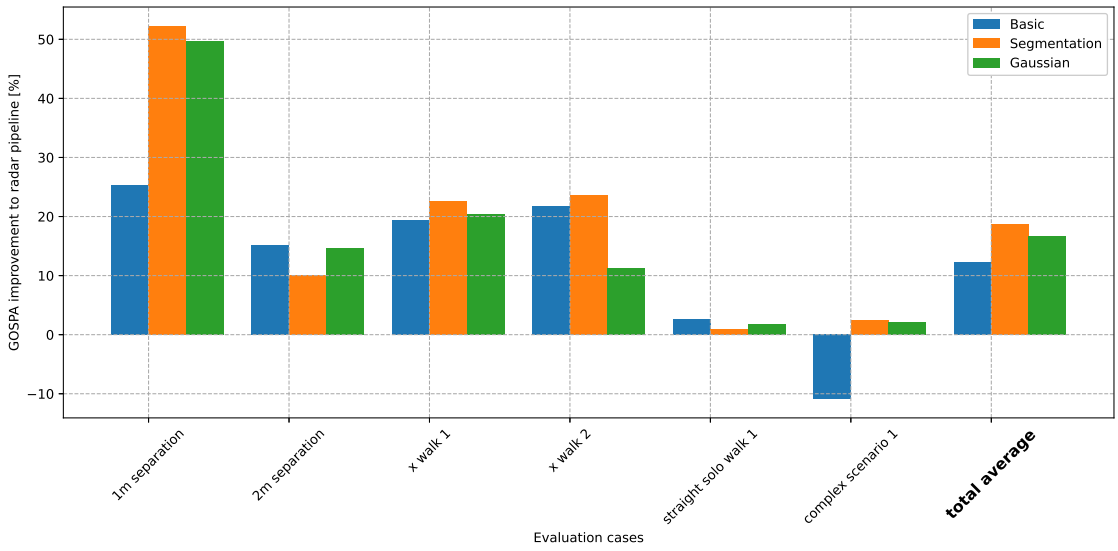


Figure 5.1: Quantitative results for the base case.

Figure 5.2 dives deeper into the scenario *1m separation* for the Segmentation annotation mode, which shows over 50% improvement. The left-most column displays time series of the GOSPA constituents compared to the radar pipeline. Note that GOSPA is a penalizing metric, meaning that a lower score is better.

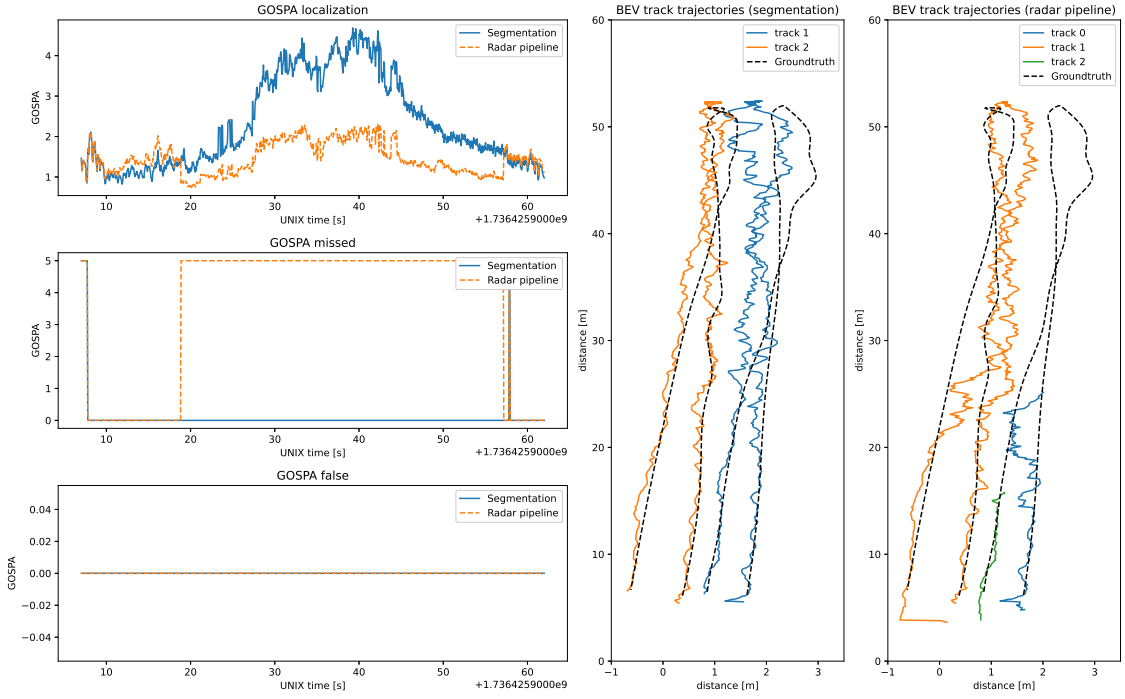


Figure 5.2: GOSPA constituents and BEV track trajectories for base case pipeline with Segmentation annotation compared to radar pipeline.

The improvement is a result of avoiding missed detections in the middle of the

scenario. This also becomes more clear when looking at the two right-most columns, which display a birds-eye view of the track trajectories compared to the ground truth. The fused pipeline manages to keep the two tracks separate for the entirety of the scenario, while the radar pipeline merges the tracks into one when they are further than 40 m from the radar. It should also be noted that while it seems like the localization is better for the radar pipeline, the reason for this is that it only contains one localization error while the fused pipeline gets a localization error from two objects. The increase in localization error for the fused pipeline is compensated by the high penalty for missed objects.

5.1.2 Bigger Image Model

Figure 5.3 displays the results when the small YOLO11n (2.5M parameters) is replaced by the bigger YOLO11x (56.9M parameters) [34]. Interestingly, a comparison with the base case in Figure 5.1 indicates that a bigger image model at best yields the same results as the smaller image model. This is not surprising, since all scenarios except *complex scenario 1* are relatively simple, which means that on these specific scenarios the two image models perform similarly.

What is more interesting is that the performance decreases significantly for the more complex scenario. The reason for this is probably due to sub-optimal ground truth labeling. Due to the limited range of the LiDAR used to create the ground truths, objects are most of the time not annotated unless they are closer than 40 meters from the radar. A better image model is able to detect objects further away than that, meaning that objects will be detected and tracked before they appear in the ground truth, resulting in an incorrect false detection.

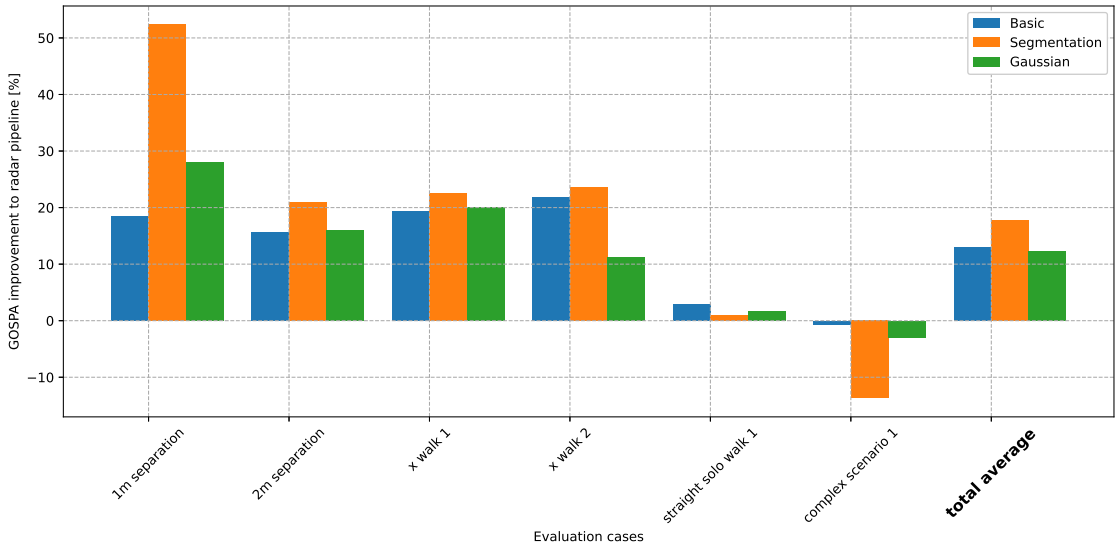


Figure 5.3: Quantitative results for bigger image model.

5.1.3 Lower Image Resolution

Results when decreasing the image size to a maximum of 640×640 and 1280×1280 pixels respectively can be seen in Figure 5.4 and 5.5. Image resizing is done by bilinear interpolation. For both the Segmentation and Gaussian annotation mode, the performance increases with higher image size when compared with the base case in Figure 5.1. This makes intuitive sense, since an image with higher resolution contains more information and thus should yield better results. Surprisingly, for the Basic annotation mode the best performance is seen with an image size of 1280, mostly caused by an increase in performance on *complex scenario 1*.

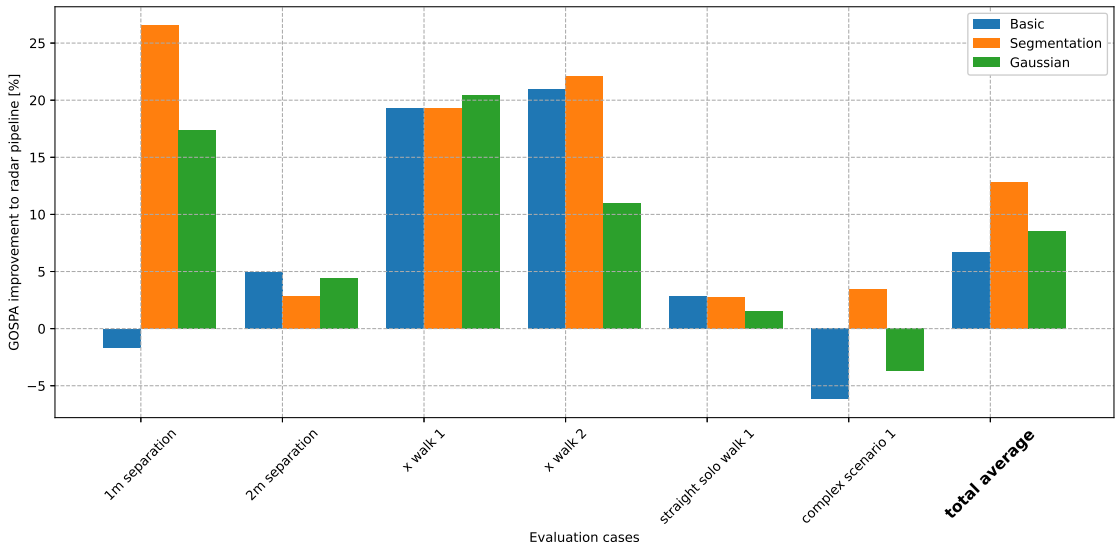


Figure 5.4: Quantitative results for pipeline fed with 640×640 images.

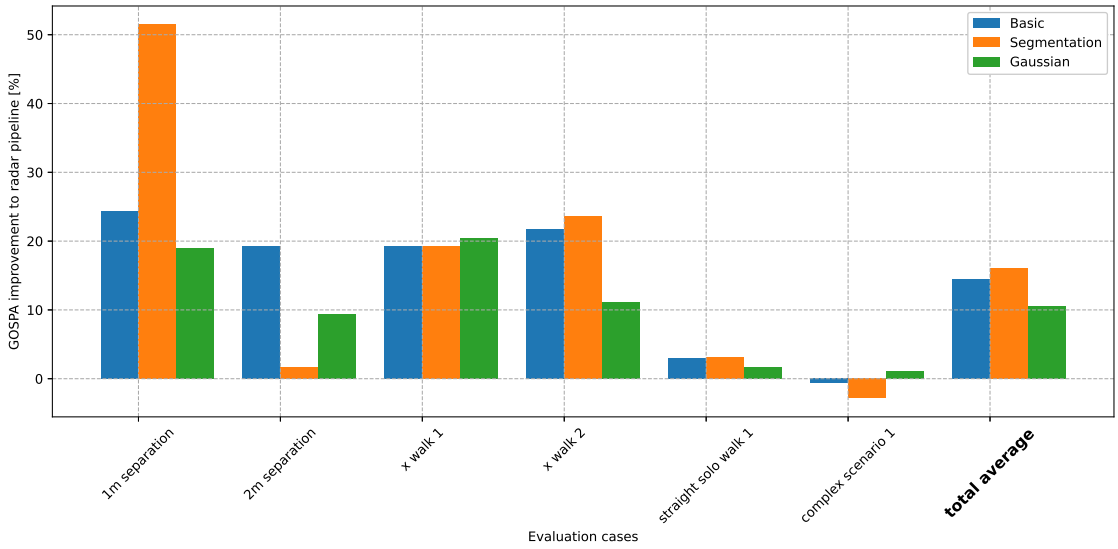


Figure 5.5: Quantitative results for pipeline fed with 1280×1280 images.

5.1.4 Removed Image Detections

To investigate what happens when the image model performance deteriorates, each image detection is discarded with a 50% probability. This simulates what can happen in for example bad weather or lighting conditions. Results can be seen in Figure 5.6. As expected this leads to a performance degradation, although the total average still shows an improvement compared to the radar pipeline. Notably, Segmentation annotation on the scenario *complex scenario 1* sees a significant performance degradation. The reason for this degradation is a false detection, likely initialized from the camera data, that appears among a group of parked vehicles in the scenario. No conclusive reason for the additional false detection has been found, but it likely stems from the difference in annotations when using semantic masks as opposed to bounding boxes.

The general improvement for Basic and Gaussian annotation is however encouraging, since they indicate a robustness towards a loss of information from the camera. This means that if the camera conditions degrade, it is still possible to use only information from the radar with no performance trade-off compared to the radar pipeline.

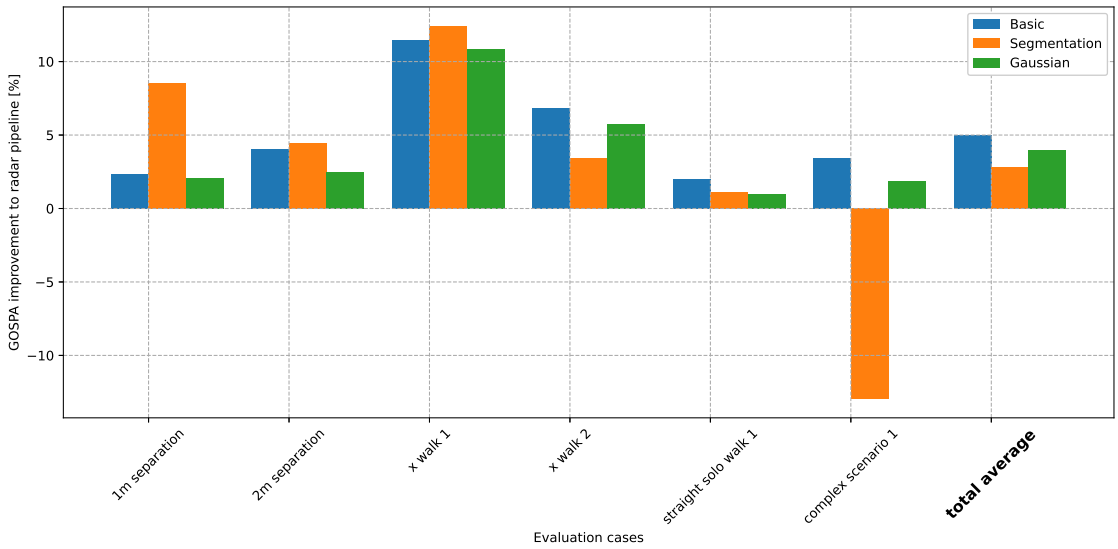


Figure 5.6: Quantitative results for pipeline where each image detection is removed with 50% probability.

5.1.5 Weighted Screening

Figure 5.7 shows the results when max operation screening is replaced with weighted screening. Both the camera and radar weight are set to 0.5, such that each sensor contributes 50% of the likelihood that the point passes the screening module. This increases the performance considerably both when comparing to the radar pipeline and with the base case in Figure 5.1.

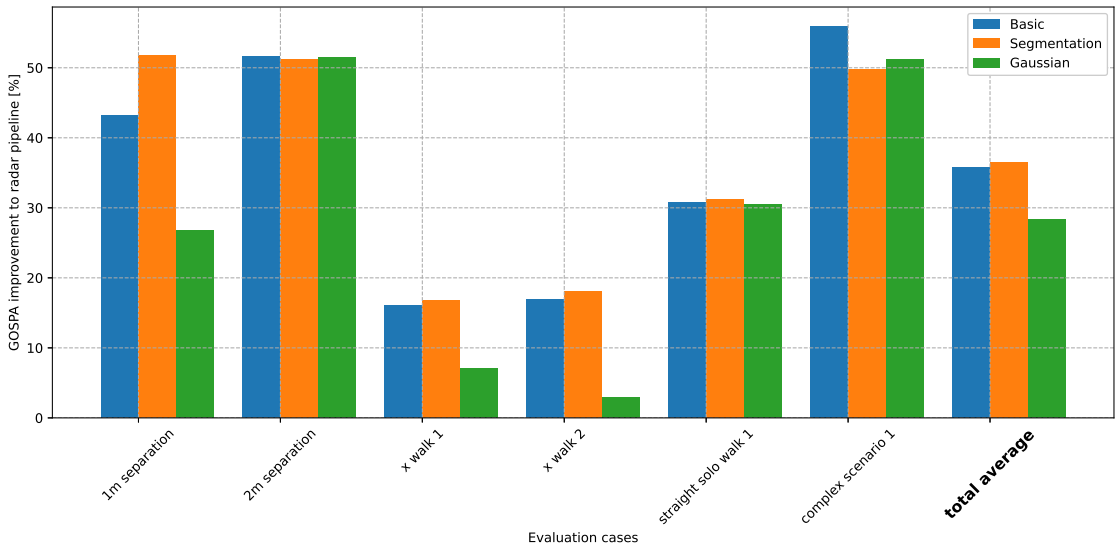


Figure 5.7: Quantitative results with a 50/50 weighted screening.

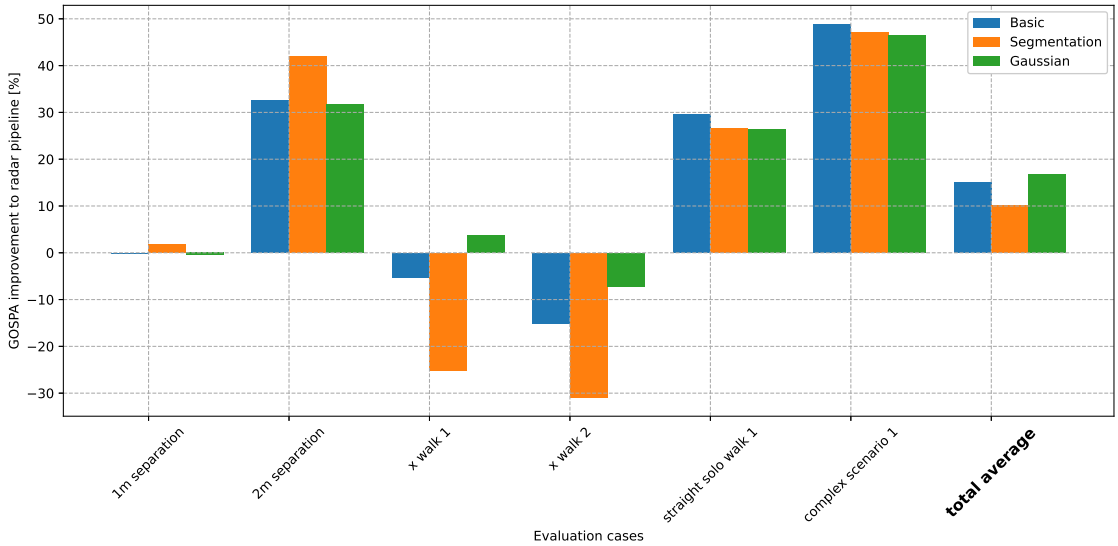


Figure 5.8: Quantitative results with a 50/50 weighted screening and image detections removed with 50% probability.

This also means that the entire system is a lot more reliant on the information from the camera, since this essentially results in that points need to be marked as interesting by both the radar and the camera to pass the screening module. Figure 5.8 shows the results with weighted screening and each image detection removed with a 50% probability. While the averaged results are still show an improvement, we also see that there is a larger variability when compared to the same case but with max operation screening in Figure 5.6. The weighted screening thus seems preferable if both the radar and camera are operating optimally, but if the performance of any of the sensors degrades the results can be worse than with a single sensor setup.

One solution to the unpredictability could be to have an adaptive weighting which is aware of the performance of the sensors, and shift the weights accordingly. For example in very bad lighting conditions, a 90/10 weighting scheme in favor of the radar might be preferable, while the opposite might be optimal when driving in e.g. a well-lit parking garage with a lot of multipath objects.

5.1.6 No Depth Estimation

The performance without depth estimation can be seen in Figure 5.9. It is apparent that the depth estimation is essential for successful tracking when using the Basic and Segmentation projection annotation modes. This makes a lot of sense, since no depth estimation means that all of the points inside a bounding box or semantic mask will be annotated, which will lead to a lot of annotated background points and thus ghost objects. The reason that the Gaussian annotation mode does not suffer to the same degree is that it already is more restrictive with how many points are annotated. The Gaussian distribution inside the bounding boxes gives points closer to its edges lower confidence. These points are more likely to be part of the background, and do not pass the screening due to the annotated confidence.

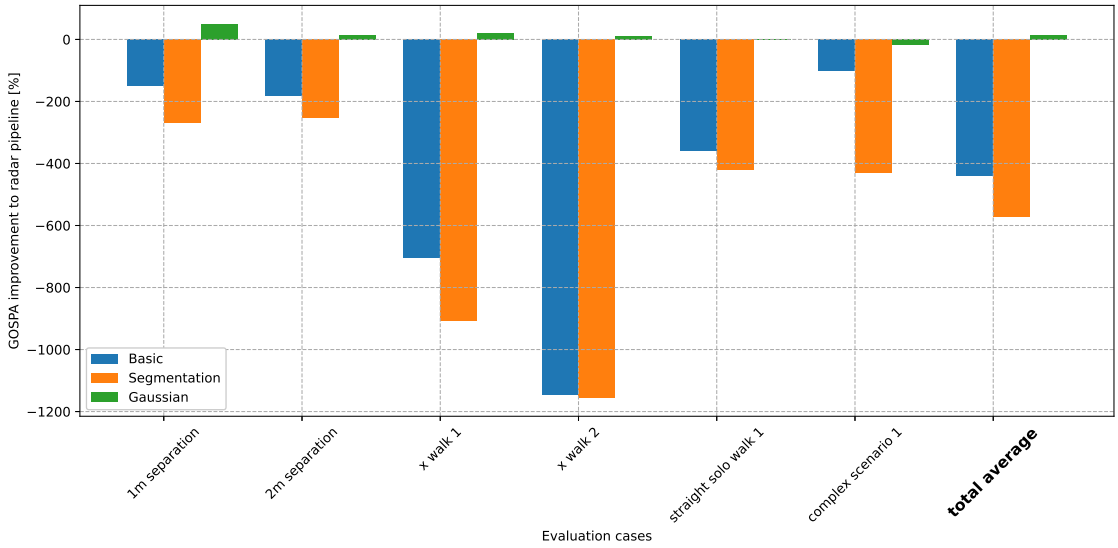


Figure 5.9: Quantitative results without depth estimation.

5.1.7 Quantitative Evaluation Summary

As a summary to the quantitative evaluation, there are a multitude of parameters to take into account when selecting the optimal configuration of the perception system. The most promising annotation modes seems to be Segmentation and Gaussian, with Gaussian being more robust when the camera performance degrades, at a slight cost in tracking performance. While a higher image resolution is preferable, the improved results are not entirely lost when downsizing the images. However, utilizing a larger image model doesn't seem to be worth the extra computational cost. Depth estimation is very important, especially if not using the Gaussian annotation

mode. Lastly, which screening version to use is a matter of deciding how confident one is that the sensors perform consistently well. If camera conditions are uncertain, max operation screening is preferable.

5.2 Qualitative Evaluation

The qualitative evaluation serves as a complement to the quantitative and aims to provide insights on the tracker's performance in more complex, realistic scenarios. By analyzing projected tracks and image detections in the camera domain, an assessment is made of how the fused pipeline handles challenges such as merging and splitting, occlusion, and overlapping objects in the camera. This is meant to help validate and contextualize the quantitative results as well as uncover failure modes in the fused pipeline.

The quantitative results suggested the best performance was acquired using the base case configuration found in 5.1.1 together with the weighted screening. As the qualitative evaluation proceeded, it became evident that this was not entirely true. Since the radar excels at measuring over ranges greater than the camera model can provide detections, it was discovered that fewer objects were being initiated at range. An example of this can be seen in Figure 5.10, depicting a long-range scenario. This was further verified by extracting the range of all tracked objects on a frame-by-frame basis and is presented as histograms in Figure 5.11. In this figure, the radar pipeline and the fusion using max screening have roughly the same shape and size, while the weighted screening has a shorter tail and generally a lower amount of objects. The second-best performing implementation is the base case, which has less camera reliance.

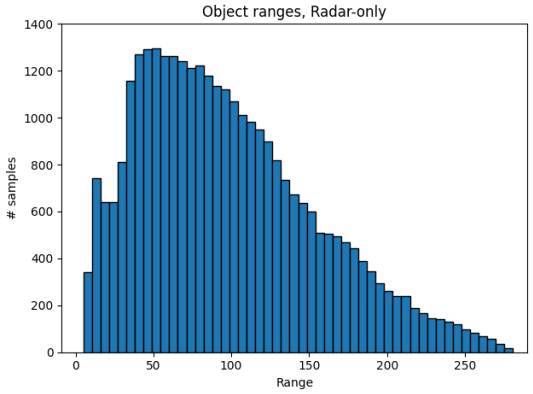
As robustness of the perception system is paramount, and the performance degradation for the Basic and Segmentation annotation modes were larger than for the Gaussian mode, the qualitative evaluation will be performed using the Gaussian annotation. Remaining parameters are set as in the quantitative base case.

(a) Fusion with max screening.

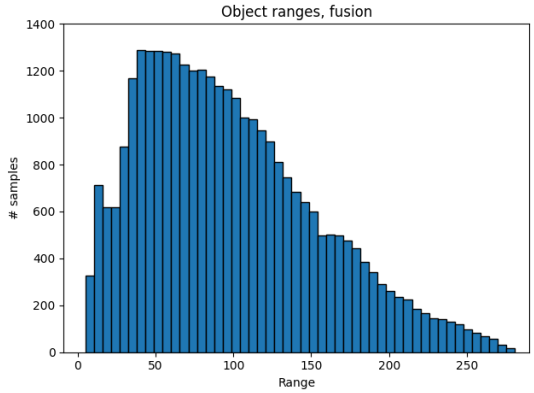
(b) Fusion with weighted screening.

Figure 5.10: Difference in objects found at range for max screening and weighted screening.

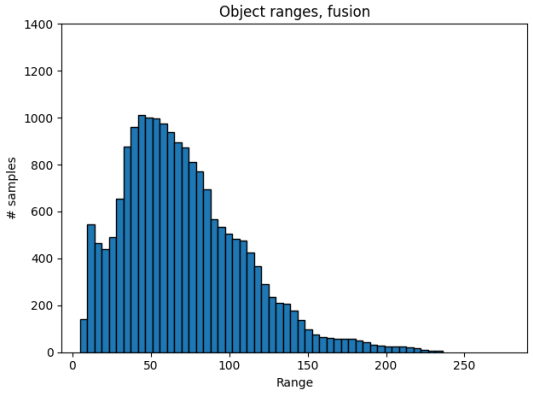
(a) Camera view of scene.



(b) Radar pipeline.



(c) Fusion, max screening.

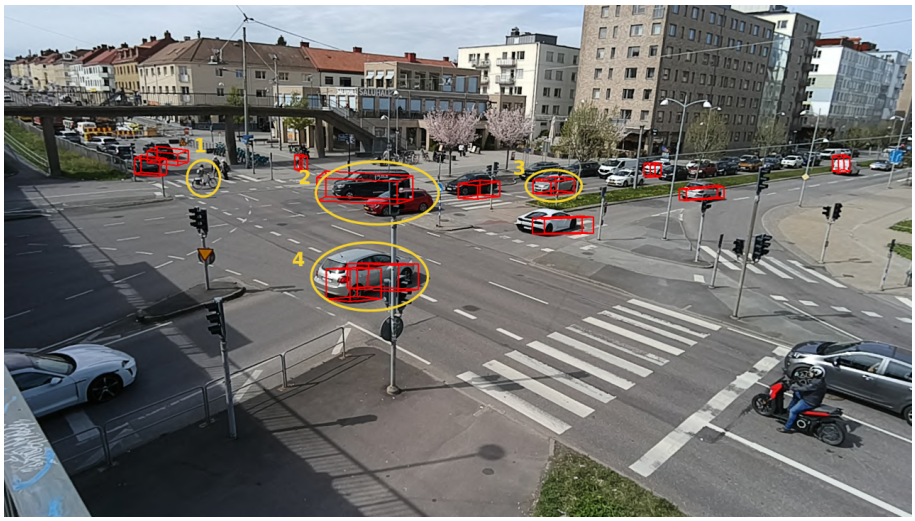


(d) Fusion, weighted screening.

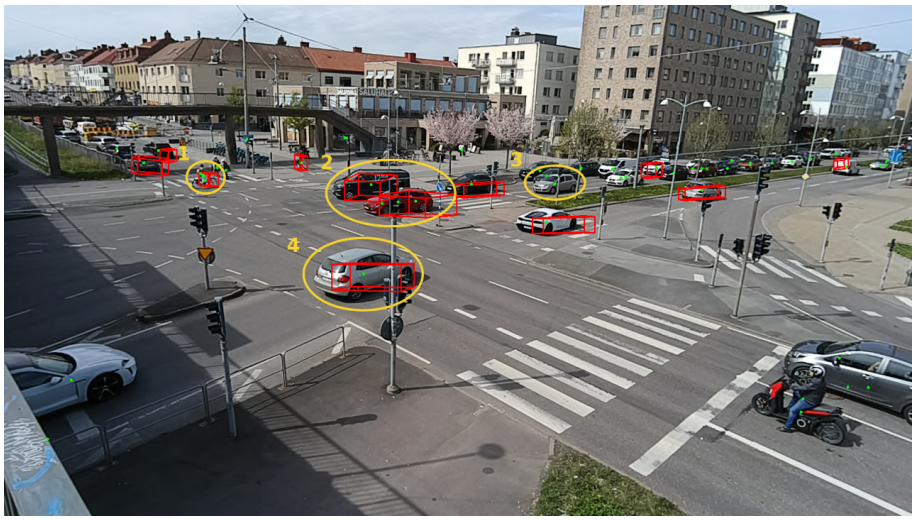
Figure 5.11: Histograms showing ranges of tracked objects with different implementations.

The first qualitative evaluation case is a static radar-camera setup at a crossing in Gothenburg. The scene captures multiple vehicles, pedestrians and cyclists as well as a bus and a large gravel transport vehicle. The red boxes are projected object tracks from the radar, and the green dots in the fused pipeline images signify the center of camera detections. Dots are used for the sake of clarity in the images, in reality the detections are bounding boxes. Regions of interest are highlighted in yellow and numbered.

In Figure 5.12, objects that are merged or split in the radar pipeline are mostly resolved (highlights 2 and 4). This is a commonality across the evaluated cases. When two road users enter the camera scene from the edge or start moving from a stand-still, the radar pipeline quite often fails to separate the tracks. This is likely due to their trajectories being parallel as well as the velocity similarity. A bicyclist is also found in the fused version (highlight 1), while it loses one of the cars that is not yet in the intersection (highlight 3). This is believed to depend on the tracker software, as the car is detected by the image model.



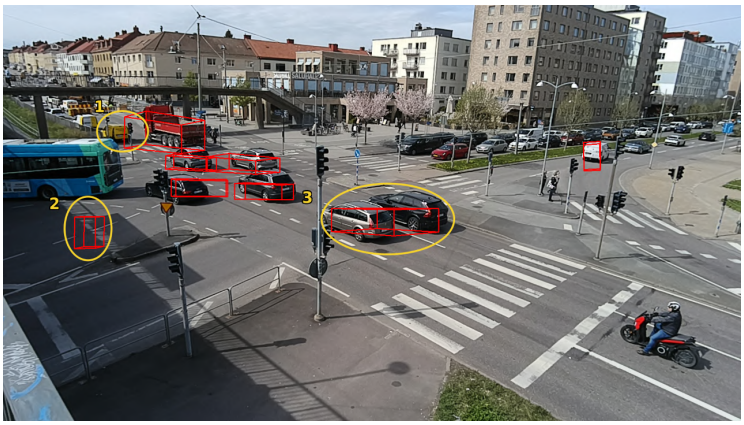
(a) Radar pipeline.



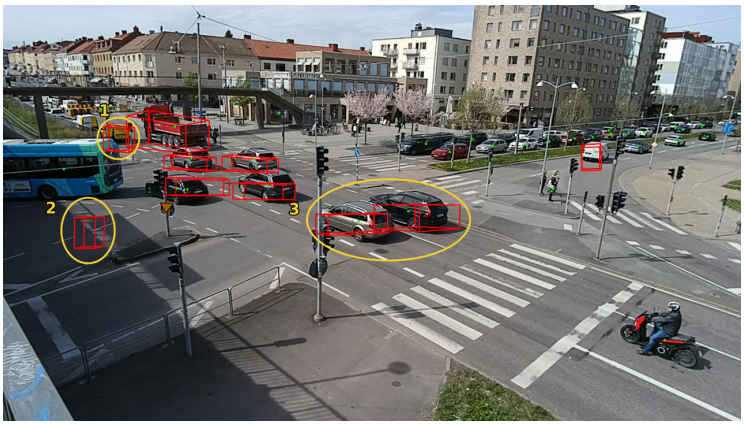
(b) Fusion pipeline, Gaussian mode.

Figure 5.12: Qualitative exhibit 1. The fusion pipeline has better estimations of object extents, and corrects some merging and splitting.

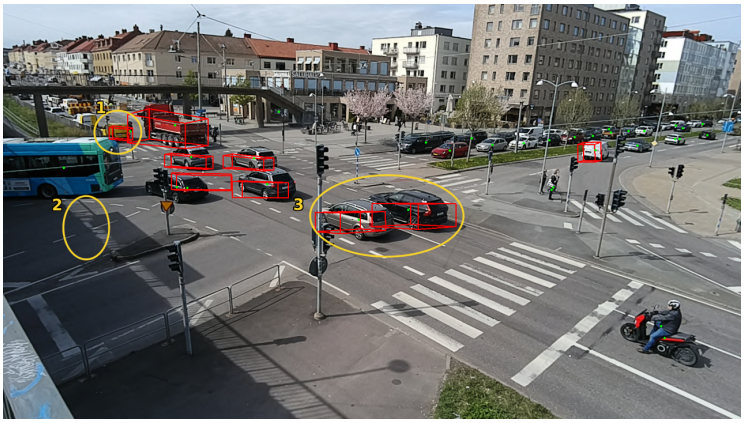
Figure 5.13 shows another snapshot, this time with the radar pipeline along with the two screening modes of the fused pipeline. As expected, the weighted screening removes the ghost track to the left since it is not detected by the camera (5.13c, highlight 2). Both of the fusion modes perform better than the radar-only pipeline with regards to merging and splitting here (highlights 1, 3) but the weighted screening has an advantage since the maximum range in this scenario is relatively short.



(a) Radar pipeline.



(b) Fusion pipeline, Gaussian mode, max screening.



(c) Fusion pipeline, Gaussian mode, weighted screening.

Figure 5.13: Qualitative exhibit 2.

All three of the implementations have issues with recognizing and correctly setting up a track for the bus. For the two fusion modes, this is partly because the image model struggles with finding the bus as a single object, perhaps due to it being articulated in the middle. Figure 5.14 gives a visual example of this. Four separate detections are made, with varying degrees of confidence. With the way the clustering is implemented, this also leads to an inability to cluster points across the areas where there is no detection overlap.

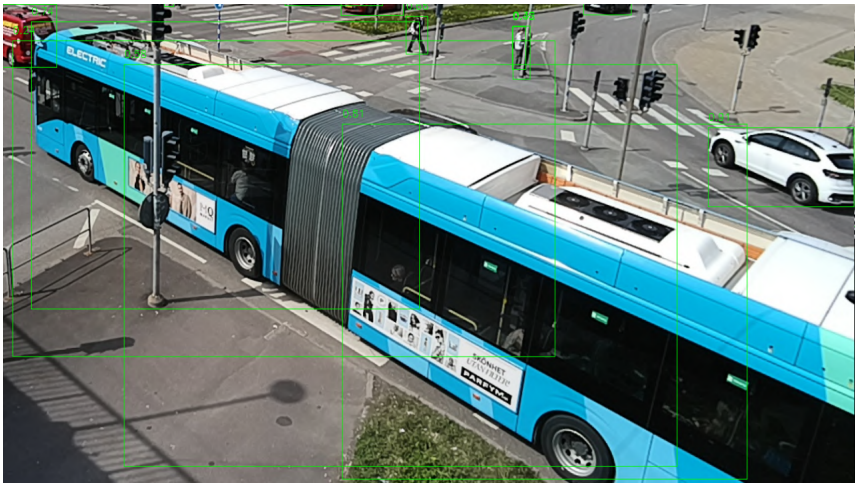
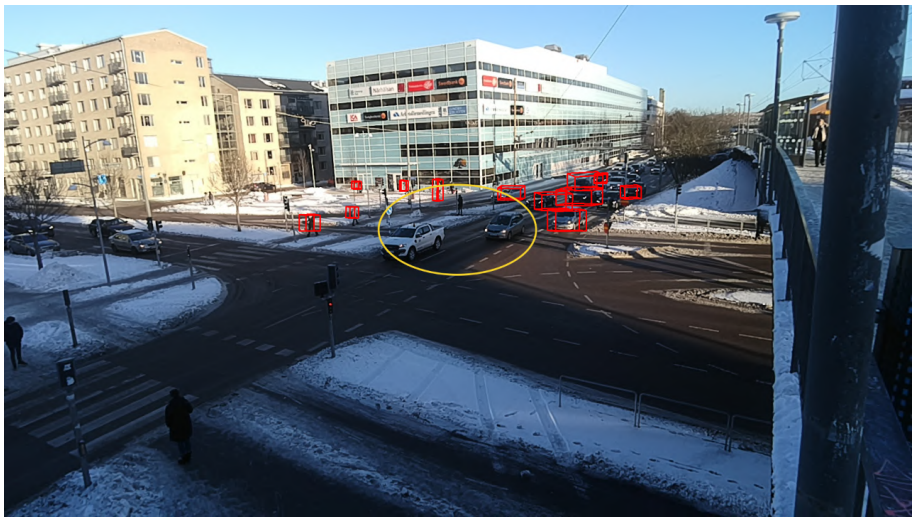


Figure 5.14: Multiple image model detections on a bus.

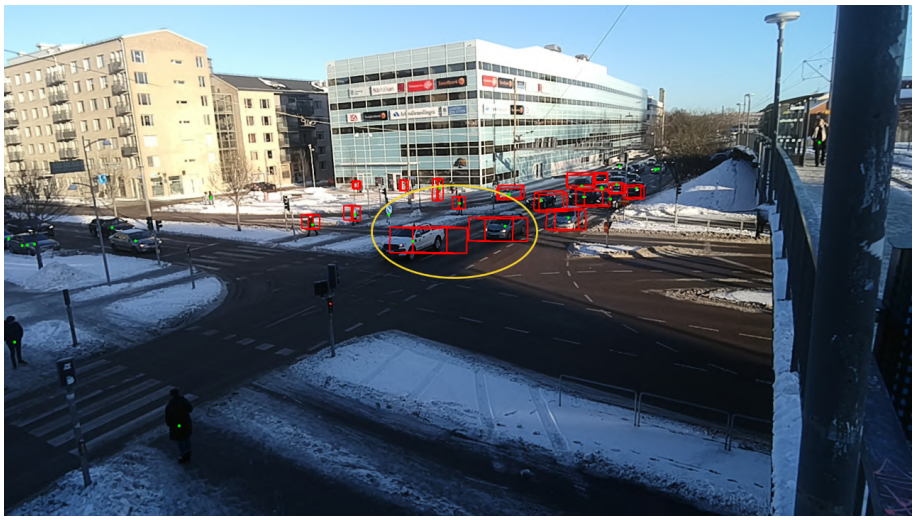
The second qualitative evaluation scenario uses the same hardware setup, but is collected later in the day when lighting conditions are somewhat worse. It depicts a traffic crossing with a varying amount of road users.

Figure 5.15 shows an instance where two cars, already far into the intersection, have been lost by the radar pipeline because it originally created a merged object of the two. As they separated the tracker killed this merged track, leading to the individual tracks being lost for a period of time. This situation is avoided in the fused pipeline thanks to earlier separation of the two targets in the camera domain.

Figure 5.16 shows two pairs of frames, both with a snapshot from the radar pipeline and from the Gaussian fusion mode. In 5.16a and 5.16b, there is a clear improvement in the creation of the truck object. However, twenty radar frames later ($\approx$1.2 seconds) this well-formed object is lost, as depicted in 5.16c and 5.16d. This shows the tracker struggling with a large object, something that the fusion seems to have difficulties with aiding in. This is theorized to be because the image model detections are less consistent for the larger vehicles, perhaps because they have a larger variety in appearance when compared to what is represented in the training data.

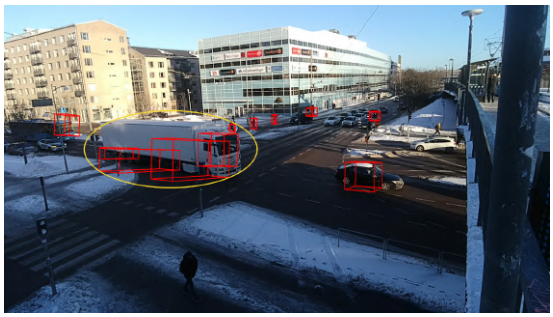


(a) Radar pipeline.

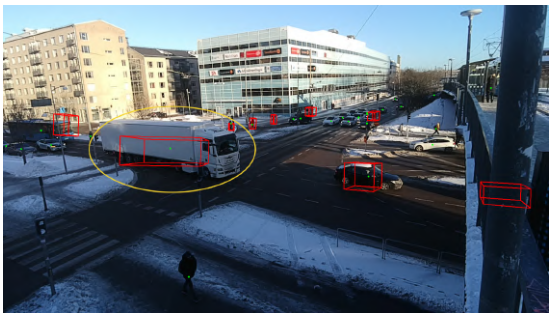


(b) Fusion pipeline, Gaussian mode.

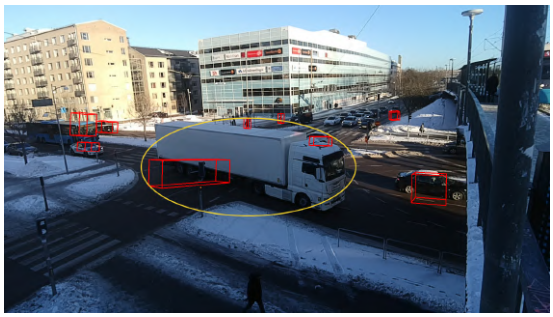
Figure 5.15: Qualitative exhibit 3. The two objects are lost in the radar pipeline because they are initially created as a single merged object.



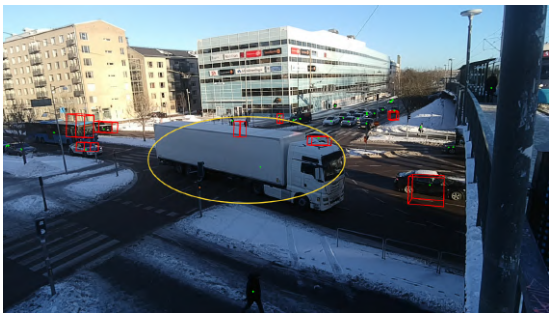
(a) Radar.



(b) Fusion, Gaussian mode.



(c) Radar.



(d) Fusion, Gaussian mode.

Figure 5.16: Qualitative exhibit 4. Subfigures (a) and (b) depict the same frame, as do (c) and (d).

5.2.1 Qualitative Evaluation Summary

The scenes selected for qualitative evaluation showcase situations where the addition of camera data mainly helps the pipeline, however, in many cases the changes are not as apparent. Simultaneously, momentary loss of an object (e.g. in an autonomous driving application) can have serious implications and should not be overlooked.

The qualitative evaluation shows that the proposed methods are able to help with the issues they were developed for, even in more complex scenarios. While the fused pipeline still exhibits some issues, it is challenging to derive which piece of the pipeline an issue stems from. The tracker occasionally exhibits unexpected behavior; however, as it falls outside the scope of this thesis, analysis of its performance is left for future work.

5.3 Runtime Evaluation

Figure 5.17 and Table 5.1 displays the average runtime of one perception iteration of the base case with the Basic annotation mode and varying image resolutions when evaluated on Sensrad's internal server using both CPU and GPU. For real-time deployment, the perception pipeline needs to match the radar's update frequency of 17 Hz, which equates to a runtime of around 59 ms for one perception iteration. This criteria is satisfied inside the area marked green in the figure. It can be seen that real-time performance is only acquired when using GPU and an image size

of 640 × 640 px. However, both GPU with image size 1280 × 1280 px and CPU with image size 640 × 640 px are close, with average runtimes of 72 ms and 73 ms respectively. The radar perception pipeline has an average runtime of 3 ms when benchmarked on the same system, meaning that the fused pipeline increases the runtime with one order of magnitude in the best case.

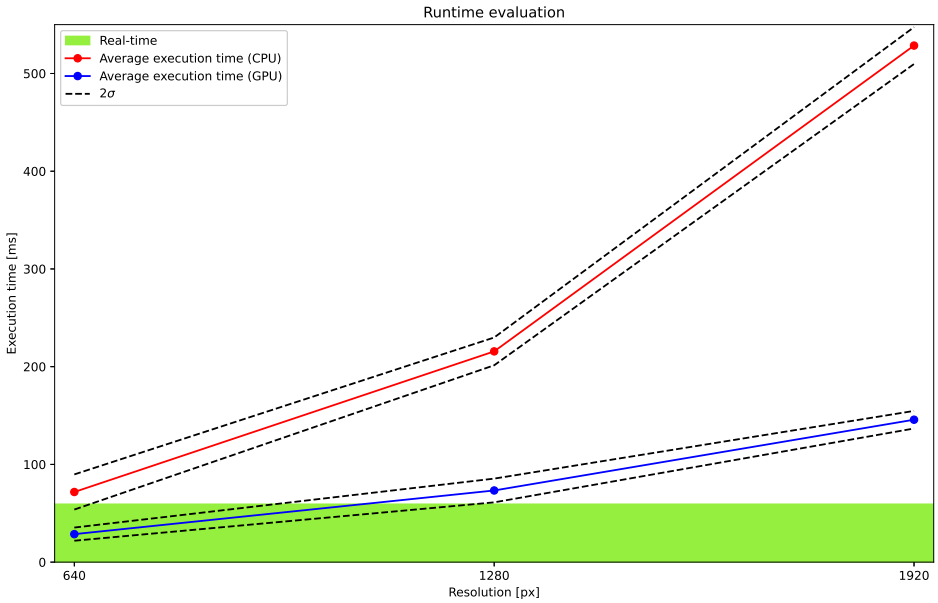


Figure 5.17: Plot of runtime versus image resolution for the base case with basic annotation mode using CPU and GPU.

Table 5.1 distinguishes between runtime on simple scenarios and runtime on complex scenarios. The complex scenarios are collected from real-world traffic situations, and the simple scenarios are simple cases collected to benchmark and test the radars functionality. The simple scenarios typically do not contain more than three objects. The entire dataset contains 8 complex scenarios and 12 simple scenarios. As seen in the table, the runtime difference between the simple and complex scenarios is negligible. This is good, since it indicates that as long as the system can be made to run in real-time in one scenario, it should not be a problem from a runtime perspective to utilize it in other types of scenarios.

Table 5.1: Average runtimes and standard deviations separated by complexity of the scenarios. All measurements in milliseconds.

Run specs / Pipeline impl.	**Radar**	**CPU 640**	**CPU 1280**	**CPU 1920**
Avg. runtime (total)	2.98 +/- 0.7	71.76 ± 9	215.63 ± 7.1	528.56 ± 9.4
Avg. runtime (simple)	2.62 +/- 0.3	71.81 ± 9.4	215.46 ± 7.3	528.59 ± 9.5
Avg. runtime (complex)	3.45 +/- 0.8	71.71 ± 8.5	215.85 ± 8.8	528.5 ± 9.4
Run specs / Pipeline impl.	**Radar**	**GPU 640**	**GPU 1280**	**GPU 1920**
Avg. runtime (total)	2.98 +/- 0.7	28.63 ± 3.4	73.31 ± 6.1	145.73 ± 4.5
Avg. runtime (simple)	2.62 +/- 0.3	28.41 ± 4.0	72.19 ± 7.3	145.20 ± 5.1
Avg. runtime (complex)	3.45 +/- 0.8	28.92 ± 2.4	74.75 ± 3.6	146.41 ± 3.5

To better understand which parts of the fused pipeline contribute the most to the runtime, Table 5.2 shows the runtime for key elements of the fusion. It can be seen that the critical part is image inference, which is a lot slower on CPU compared to GPU, and increases with image size. The image undistortion also contributes to a non-negligible portion of the runtime, with a constant execution time of around 18 ms. This is extra interesting, since there is no strict requirement on undistorting the image. If the camera used has low distortion, the image model should work sufficiently well on the distorted image. Additionally, the projection of the point cloud onto the image can be done with the distortion in mind. Thus, with the right camera, it should also be possible to run the pipeline in real-time with 640×640 px images both CPU and GPU, as well as with 1280×1280 px on GPU.

Table 5.2: Breakdown of runtime contributions, all values in milliseconds. Percentage values are in relation to the entire pipeline's runtime.

Pipeline part / implementation	**CPU 640**	**CPU 1280**	**CPU 1920**	**GPU 640**	**GPU 1280**	**GPU 1920**
Image undistortion	19.8 (27.6%)	18.3 (8.5%)	18.1 (3.4%)	18.1 (63.3%)	17.8 (24.4%)	18.0 (12.3%)
Image inference	48.8 (67.9%)	194.2 (90.1%)	507.3 (96.0%)	7.5 (26.1%)	52.3 (71.5%)	124.7 (85.6%)
Point cloud projection	0.1 (0.1%)	0.1 (0.0%)	0.1 (0.0%)	0.0 (0.2%)	0.0 (0.1%)	0.0 (0.0%)
Point-to-detection association	0.1 (0.1%)	0.1 (0.0%)	0.1 (0.0%)	0.1 (0.2%)	0.1 (0.1%)	0.1 (0.1%)
Depth estimation	0.1 (0.1%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.2%)	0.0 (0.0%)	0.0 (0.0%)
Point cloud annotation	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)

Both the Basic and the Gaussian annotation mode are based on image models for object detection. The results above thus indicate that they both have potential for real-time deployment. To investigate the real-time capabilities of the Segmentation annotation mode, Table 5.3 shows the image inference time of an instance segmentation model on CPU and GPU for various image resolutions. When comparing with Table 5.2, it can be seen that the model for instance segmentation is slower than the one for object detection. The conclusion is that if the Segmentation annotation mode is used, a GPU and images with size 640×640 px is required for real-time development.

Table 5.3: Image inference with instance segmentation model for various image sizes. All values in milliseconds.

Pipeline implementation	**CPU 640**	**CPU 1280**	**GPU 640**	**GPU 1280**
Image inference time	61.8 ± 0.5	242.7 ± 2.3	14.3 ± 1.6	62.8 ± 4.2

6

Conclusion

This thesis has investigated how camera information can be used in a 4D radar tracking application to improve performance, with a focus on the issues of object merging, object splitting, and multipath objects. The suggested methods do not utilize any NNs in the fusion step, but instead projects the radar point cloud onto the image and adds information on a point-by-point basis. Points that project onto objects detected in the image are more likely to pass the screening step in the tracking pipeline. They also get a suggestion as to which object they belong to. This information is then used in the clustering step of the tracking pipeline. Initial results using the GOSPA metric show significant improvements across all scenarios when evaluated against the radar pipeline, with some scenarios seeing up to 50% improvement. The suggested solution is shown to be robust both in regards to a decrease in camera resolution and in simulated adverse camera conditions. When evaluating qualitatively, the results in general show improvements with regards to the presented tracking challenges. The improvements are more distinct closer to the radar-camera setup, something that is caused by a resolution difference between the two sensors.

Since the long term aim is to be able to run in real-time, a subset of the suggested methods were also evaluated with respect to runtime. The results show that when using Basic or Gaussian annotation modes, real-time deployment should be possible by downscaling the images to 640×640 px, using either CPU or GPU. It should also be possible to run images with size 1280×1280 in real-time, but only with a GPU. If Segmentation annotation is used, real-time application is instead limited to only 640×640 px images and requires a GPU.

6.1 Future work

There are several promising directions for extending the work presented in the thesis. While the proposed methods show potential, further investigation is needed to evaluate their performance and applicability in real-world scenarios. The following suggestions outline potential next steps for improving and expanding upon the system.

The first step is to collect and annotate more data, to be able to better validate and analyze the results. With the result of this thesis, additional data is still needed to

help further clarify the performance differences of the proposed methods. Further verification is needed to ascertain the type of impact that the type of scenario, weather conditions, et cetera have. Another point of interest is to investigate how a dynamic weighting of the trust in the two sensors can be done if using the weighted screening. The evaluation shows that weighted screening has a lot of potential with regards to filtering out multipath objects. If it was possible to adjust the weights on the fly, this type of screening could be made to work in a more diverse set of circumstances. To be able to verify this, more labeled data is needed.

It would also be interesting to benchmark the suggested solution against the current state of the art, to see how it compares to NN-heavy approaches to 4D radar-camera fusion. To do this, the pipeline would need to be modified to be able to run with non-Sensrad radar data. The authors have limited insight in how time consuming such a modification would be. Since the parameters of the pipeline are adapted to point clouds from Sensrad's Hugin D1 radar, a new parameter search would probably have to be conducted. It is also probable that some modifications are required for the pipeline to be able to use a point cloud with a potentially different data structure.

As a last suggestion, there are other interesting fusion approaches that have been considered during the thesis but ruled out for different reasons. The current solutions are all based on projecting the point cloud onto a camera image. It would be very interesting to do the opposite, by for example running a depth estimation image model in combination with e.g. instance segmentation to do the opposite projection of image pixels out into the 3D world and handle association there.

Bibliography

[1] Kun Shi, Shibo He, Zhenyu Shi, et al. *Radar and Camera Fusion for Object Detection and Tracking: A Comprehensive Survey*. arXiv:2410.19872 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2410.19872. URL: http://arxiv.org/abs/2410.19872 (visited on 01/14/2025).

[2] Shuai Wang, Luoyu Mei, Ruofeng Liu, et al. "Multi-Modal Fusion Sensing: A Comprehensive Review of Millimeter-Wave Radar and Its Integration With Other Modalities". In: *IEEE Communications Surveys & Tutorials* 27.1 (Feb. 2025), pp. 322–352. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2024.3398004. (Visited on 05/27/2025).

[3] Andras Palffy, Ewoud Pool, Srimannarayana Baratam, et al. "Multi-Class Road User Detection With 3+1D Radar in the View-of-Delft Dataset". In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022), pp. 4961–4968. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2022.3147324. URL: https://ieeexplore.ieee.org/document/9699098/ (visited on 01/14/2025).

[4] Lianqing Zheng, Zhixiong Ma, Xichan Zhu, et al. "TJ4DRadSet: A 4D Radar Dataset for Autonomous Driving". In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. Macau, China: IEEE, Oct. 2022, pp. 493–498. ISBN: 978-1-6654-6880-0. DOI: 10.1109/ITSC55140.2022.9922539. URL: https://ieeexplore.ieee.org/document/9922539/ (visited on 01/14/2025).

[5] Dong-Hee Paek, Seung-Hyun Kong, and Kevin Tirta Wijaya. "K-Radar: 4D Radar Object Detection for Autonomous Driving in Various Weather Conditions". Version 4. In: (2022). DOI: 10.48550/ARXIV.2206.08171. URL: https://arxiv.org/abs/2206.08171 (visited on 04/07/2025).

[6] S.L. Wilson and B.D. Carlson. "Radar Detection in Multipath". In: *IEE Proceedings - Radar, Sonar and Navigation* 146.1 (1999), p. 45. ISSN: 13502395. DOI: 10.1049/ip-rsn:19990264. (Visited on 04/10/2025).

[7] Abu Sajana Rahmathullah, Ángel F. García-Fernández, and Lennart Svensson. "Generalized optimal sub-pattern assignment metric". In: *2017 20th International Conference on Information Fusion (Fusion)*. arXiv:1601.05585 [cs]. July 2017, pp. 1–8. DOI: 10.23919/ICIF.2017.8009645. URL: http://arxiv.org/abs/1601.05585 (visited on 02/26/2025).

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[9] Erich Schubert, Jörg Sander, Martin Ester, et al. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Transactions on Database Systems* 42.3 (Sept. 30, 2017), pp. 1–21. ISSN: 0362-5915, 1557-4644. DOI: 10.1145/3068335. URL: https://dl.acm.org/doi/10.1145/3068335 (visited on 05/15/2025).

[10] Joseph Redmon, Santosh Divvala, Ross Girshick, et al. *You Only Look Once: Unified, Real-Time Object Detection*. Version 5. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640 (visited on 05/16/2025). Pre-published.

[11] Yunjie Tian, Qixiang Ye, and David Doermann. *YOLOv12: Attention-Centric Real-Time Object Detectors*. Version Number: 1. 2025. DOI: 10.48550/ARXIV.2502.12524. URL: https://arxiv.org/abs/2502.12524 (visited on 04/02/2025).

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[13] Ross Girshick. "Fast R-CNN". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, et al. *Feature Pyramid Networks for Object Detection*. 2016. DOI: 10.48550/ARXIV.1612.03144. (Visited on 05/16/2025).

[15] Mujadded Al Rabbani Alif and Muhammad Hussain. *YOLOv1 to YOLOv10: A comprehensive review of YOLO variants and their application in the agricultural domain*. Version Number: 1. 2024. DOI: 10.48550/ARXIV.2406.10139. URL: https://arxiv.org/abs/2406.10139 (visited on 01/23/2025).

[16] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. *BoT-SORT: Robust Associations Multi-Pedestrian Tracking*. Version 2. 2022. DOI: 10.48550/ARXIV.2206.14651. URL: https://arxiv.org/abs/2206.14651 (visited on 04/14/2025). Pre-published.

[17] Yifu Zhang, Peize Sun, Yi Jiang, et al. "ByteTrack: Multi-object Tracking by Associating Every Detection Box". In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, et al. Vol. 13682. Cham: Springer Nature Switzerland, 2022, pp. 1–21. ISBN: 978-3-031-20046-5 978-3-031-20047-2. DOI: 10.1007/978-3-031-20047-2_1. (Visited on 04/14/2025).

[18] Ultralytics. *Non-Maximum Suppression (NMS)*. 2025. URL: https://www.ultralytics.com/glossary/non-maximum-suppression-nms (visited on 04/14/2025).

[19] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[20] William H. Press, Saul A. Teukolsky, William T. Vetterling, et al. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd. New York, NY, USA: Cambridge University Press, 1992. ISBN: 978-0521431088.

[21] Zeyu Han, Jiahao Wang, Zikun Xu, et al. *4D Millimeter-Wave Radar in Autonomous Driving: A Survey*. 2023. DOI: 10.48550/ARXIV.2306.04242. (Visited on 05/06/2025).

[22] James W. Cooley and John W. Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (1965), pp. 297–301. ISSN: 0025-5718, 1088-6842. DOI: 10.1090/S0025-5718-1965-0178586-1. (Visited on 04/10/2025).

[23] Hermann Rohling. “Radar CFAR Thresholding in Clutter and Multiple Target Situations”. In: *IEEE Transactions on Aerospace and Electronic Systems* AES-19.4 (July 1983), pp. 608–621. ISSN: 0018-9251. DOI: 10.1109/TAES.1983.309350. (Visited on 05/06/2025).

[24] Karl Granstrom, Marcus Baum, and Stephan Reuter. “Extended Object Tracking: Introduction, Overview and Applications”. In: (2016). DOI: 10.48550/ARXIV.1604.00970. (Visited on 05/07/2025).

[25] G.W. Pulford. “Taxonomy of Multiple Target Tracking Methods”. In: *IEE Proceedings - Radar, Sonar and Navigation* 152.5 (2005), p. 291. ISSN: 13502395. DOI: 10.1049/ip-rsn:20045064. (Visited on 04/10/2025).

[26] S.S. Blackman. “Multiple Hypothesis Tracking for Multiple Target Tracking”. In: *IEEE Aerospace and Electronic Systems Magazine* 19.1 (Jan. 2004), pp. 5–18. ISSN: 0885-8985. DOI: 10.1109/MAES.2004.1263228. (Visited on 04/10/2025).

[27] R.P.S. Mahler. “Multitarget Bayes Filtering via First-Order Multitarget Moments”. In: *IEEE Transactions on Aerospace and Electronic Systems* 39.4 (Oct. 2003), pp. 1152–1178. ISSN: 0018-9251. DOI: 10.1109/TAES.2003.1261119. (Visited on 04/10/2025).

[28] Lianqing Zheng, Jianan Liu, Runwei Guan, et al. *Doracamom: Joint 3D Detection and Occupancy Prediction with Multi-view 4D Radars and Cameras for Omnidirectional Perception*. Version Number: 1. 2025. DOI: 10.48550/ARXIV.2501.15394. URL: https://arxiv.org/abs/2501.15394 (visited on 01/31/2025).

[29] Jing Zeng, Dipayan Mitra, Ming Chen, et al. “Camera-Assisted Radar Detection Clustering for Extended Target Tracking”. In: *IEEE Transactions on Instrumentation and Measurement* 73 (2024), pp. 1–17. ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2024.3400347. URL: https://ieeexplore.ieee.org/document/10539619/ (visited on 02/03/2025).

[30] Xingbang Tang, Xianghong Cheng, and Ninghui Xu. “A Robust Multiobject Tracking Method Based on 4-D Millimeter-Wave Radar and Monocular Vision Fusion”. In: *IEEE Sensors Journal* 24.22 (Nov. 2024), pp. 37764–37774. ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: 10.1109/JSEN.2024.3465019. URL: https://ieeexplore.ieee.org/document/10696909/ (visited on 01/14/2025).

[31] Alex H. Lang, Sourabh Vora, Holger Caesar, et al. *PointPillars: Fast Encoders for Object Detection from Point Clouds*. Version Number: 2. 2018. DOI: 10.48550/ARXIV.1812.05784. URL: https://arxiv.org/abs/1812.05784 (visited on 01/23/2025).

[32] Zheng Ge, Songtao Liu, Feng Wang, et al. *YOLOX: Exceeding YOLO Series in 2021*. Version Number: 2. 2021. DOI: 10.48550/ARXIV.2107.08430. URL: https://arxiv.org/abs/2107.08430 (visited on 04/07/2025).

[33] Steven Macenski, Tully Foote, Brian Gerkey, et al. “Robot Operating System 2: Design, Architecture, and Uses in the Wild”. In: *Science Robotics* 7.66 (May 25, 2022), eabm6074. ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/10.1126/scirobotics.abm6074 (visited on 05/12/2025).

[34] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

A

Appendix 1

A.1 Technical specifications of Hugin D1 radar

Table A.1: Tehnical specifications of the Hugin D1 radar used in the thesis.

Specification	**Value**
Frequency range	$76 - 81$ GHz
Antennas	48 TX x 48 RX MIMO
Update rate	17 Hz
Power consumption	20 W nom, $9 - 32$ V
Detection range	0.2 m - 450 m (mode dependent)
Range resolution	17 cm - 110 cm (mode dependent)
Azimuth FOV	$100°(\pm 50°)$
Elevation FOV	$30°(\pm 15°)$
Azimuth resolution	$1.2°$
Elevation resolution	$1.5°$
Doppler detection range	-44 m/s to 87 m/s
Doppler resolution	0.08 m/s

B

Appendix 2

B.1 GOSPA values for quantitative evaluation

Absolute GOSPA values and percentage difference to the radar pipeline for all quantitative evaluation cases are presented below. The best performing annotation mode for each scenario and evaluation case are bolded.

Table B.1: Quantitative results for the base case.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	3.7509	25.4	**2.397**	**52.3**	2.5271	49.7
2m separation	4.7241	**4.0099**	**15.1**	4.2476	10.1	4.0295	14.7
x walk 1	1.5711	1.2667	19.4	**1.2155**	**22.6**	1.2519	20.3
x walk 2	1.5787	1.2357	21.7	**1.2066**	**23.6**	1.4017	11.2
straight solo walk 1	1.7228	**1.6772**	**2.7**	1.7076	0.9	1.6936	1.7
complex scenario 1	12.5331	13.9024	-10.9	**12.23**	**2.4**	12.2681	2.1
Average:	4.526	4.3071	12.2	**3.8341**	**18.6**	3.862	16.6

Table B.2: Quantitative results for fused pipeline with better image model.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	4.1003	18.4	**2.395**	**52.3**	3.6194	28
2m separation	4.7241	3.9847	15.7	**3.7356**	**20.9**	3.9682	16
x walk 1	1.5711	1.2665	19.4	**1.2168**	**22.6**	1.2571	20
x walk 2	1.5787	1.2353	21.8	**1.2059**	**23.6**	1.4008	11.3
straight solo walk 1	1.7228	**1.6713**	**3**	1.7059	1	1.6933	1.7
complex scenario 1	**12.5331**	12.6186	-0.7	14.2506	-13.7	12.9069	-3
Average:	4.526	4.1461	12.9	**4.085**	**17.8**	4.141	12.3

Table B.3: Quantitative results for pipeline input with 640x640 images.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	5.1084	-1.6	**3.6917**	**26.5**	4.1521	17.4
2m separation	4.7241	**4.4917**	**4.9**	4.5899	2.8	4.5138	4.5
x walk 1	1.5711	1.2677	19.3	1.2672	19.3	**1.2502**	**20.4**
x walk 2	1.5787	1.2476	21	**1.2297**	**22.1**	1.4054	11
straight solo walk 1	1.7228	**1.6732**	**2.9**	1.6757	2.7	1.6969	1.5
complex scenario 1	12.5331	13.3048	-6.2	**12.1049**	**3.4**	13.002	-3.7
Average:	4.526	4.5156	6.7	**4.0932**	**12.8**	4.3367	8.5

Table B.4: Quantitative results for pipeline input with 1280x1280 images.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	3.8033	24.3	**2.4395**	**51.5**	4.0691	19
2m separation	4.7241	**3.8153**	**19.2**	4.6482	1.6	4.2799	9.4
x walk 1	1.5711	1.269	19.2	1.2688	19.2	**1.2494**	**20.5**
x walk 2	1.5787	1.2367	21.7	**1.2062**	**23.6**	1.4022	11.2
straight solo walk 1	1.7228	1.6708	3	**1.6686**	**3.1**	1.6945	1.6
complex scenario 1	12.5331	12.6115	-0.6	12.888	-2.8	**12.3929**	**1.1**
Average:	4.526	4.0678	14.5	**4.0199**	**16**	4.1813	10.5

Table B.5: Quantitative results for pipeline where each image detection is removed with 50% probability.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	4.9085	2.3	**4.5963**	**8.5**	4.9211	2.1
2m separation	4.7241	4.5332	4	**4.5139**	**4.4**	4.6076	2.5
x walk 1	1.5711	1.3906	11.5	**1.3755**	**12.4**	1.4007	10.8
x walk 2	1.5787	**1.4709**	**6.8**	1.5243	3.4	1.4879	5.7
straight solo walk 1	1.7228	**1.6877**	**2**	1.7039	1.1	1.706	1
complex scenario 1	12.5331	**12.1015**	**3.4**	14.1595	-13	12.3003	1.9
Average:	4.526	**4.3487**	**5**	4.6456	2.8	4.404	4

Table B.6: Quantitative results with a 50/50 weighted screening.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	2.8571	43.2	**2.4214**	**51.8**	3.6803	26.8
2m separation	4.7241	**2.2857**	**51.6**	2.3025	51.3	2.2953	51.4
x walk 1	1.5711	1.3189	16.1	**1.3073**	**16.8**	1.461	7
x walk 2	1.5787	1.3107	17	**1.2945**	**18**	1.5326	2.9
straight solo walk 1	1.7228	1.1918	30.8	**1.1852**	**31.2**	1.1967	30.5
complex scenario 1	12.5331	**5.5292**	**55.9**	6.2923	49.8	6.123	51.1
Average:	4.526	2.4156	35.8	**2.4672**	**36.5**	2.7148	28.3

Table B.7: Quantitative results with 50/50 weighted screening and image detections removed with 50% probability.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	5.0313	-0.1	**4.9385**	**1.7**	5.0494	-0.5
2m separation	4.7241	3.1883	32.5	**2.7386**	**42**	3.2271	31.7
x walk 1	1.5711	1.6542	-5.3	1.9666	-25.2	**1.5133**	**3.7**
x walk 2	**1.5787**	1.818	-15.2	2.0671	-30.9	1.6928	-7.2
straight solo walk 1	1.7228	**1.2113**	**29.7**	1.2654	26.5	1.2697	26.3
complex scenario 1	12.5331	**6.4055**	**48.9**	6.6362	47.1	6.7053	46.5
Average:	4.526	3.2181	15.1	3.2687	10.2	**3.243**	**16.7**

Table B.8: Quantitative results without depth estimation.

Scenario / Annotation mode	Radar ↓	Basic ↓	% diff ↑	Segmentation ↓	% diff ↑	Gaussian ↓	% diff ↑
1m separation	5.0258	12.614	-151	18.5769	-269.6	**2.5338**	**49.6**
2m separation	4.7241	13.2913	-181.3	16.6472	-252.4	**4.0313**	**14.7**
x walk 1	1.5711	12.6494	-705.1	15.8321	-907.7	**1.2537**	**20.2**
x walk 2	1.5787	19.6775	-1146.4	19.8115	-1154.9	**1.4008**	**11.3**
straight solo walk 1	1.7228	7.9102	-359.1	8.9484	-419.4	**1.6934**	**1.7**
complex scenario 1	**12.5331**	25.2119	-101.2	66.422	-430	14.893	-18.8
Average:	4.526	15.2257	-440.7	24.373	-572.3	**4.301**	**13.1**

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY