h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

**Darmstadt University of Applied Sciences**

– Faculty of Computer Science –

# Rad-dar: Evaluation and ranking of cycling infrastructure through the implementation of an IOT device using a 60 GHz radar to measure the distance of overtaking cars.

Submitted in partial fulfilment of the requirements for the degree of

Master of Science (M.Sc.)

by

**Sebastian Schuch**

Matriculation number: 751437

| | | |
|---|---|---|
| First Examiner | : | Prof. Dr. Michael von Rüden |
| Second Examiner | : | Prof. Dr. Ronald Charles Moore |

# DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

*Darmstadt, April 14, 2023*

Sebastian Schuch

# ABSTRACT

In this thesis we use the design science research method to develop an IOT device capable of documenting road safety and the attractiveness of cycling. This is done by automatically mapping the availability and quality of cycling infrastructure using modern 60GHz radar technology. These low-cost, high-performance radars are capable of measuring the distance and speed of passing cars up to 40 times per second. We are undertaking three successive development iterations, improving on the results of the previous evaluation. A key objective is to enable the prototype to operate without user intervention while the cyclist is riding. This requires sophisticated signal processing to detect cars and measure accurate distances at short duty cycles. In addition, the collected data should be transmitted via Bluetooth to the user's mobile phone for temporary storage. After each trip, the collected positions, speeds and distances are sent to a scalable infrastructure that enables collaborative data collection for in-depth analysis. This data can be used to identify safer routes and highlight necessary improvements to cycle routes, supporting urban development by prioritising roads with the best cost-benefit ratio. To do this, we used a clustering algorithm to find roads and locations with a significantly higher incidence of close encounters. All of this was validated through a series of experimental setups, culminating in a two-month deployment of the functional prototype in real-world traffic conditions. In the end, the data highlighted three distinct locations. Despite a lack of urban planning expertise, at least the reason why these locations were identified seems obvious, suggesting that the data and the sensor are promising. Concluding the thesis with a discussion of the limitations of the prototype and the future work that is required to make it a viable product, and to increase the user incentives to actually use it.

# ZUSAMMENFASSUNG

In dieser Arbeit nutzen wir die Forschungsmethode der Design Science, um ein IOT-Gerät zu entwickeln, das die Verkehrssicherheit und die Attraktivität des Radfahrens dokumentieren kann. Dies geschieht durch die automatische Kartierung der Verfügbarkeit und Qualität der Radverkehrsinfrastruktur mithilfe moderner 60-GHz-Radartechnologie. Diese kostengünstigen und leistungsstarken Radargeräte sind in der Lage, den Abstand und die Geschwindigkeit von vorbeifahrenden Autos bis zu 40 Mal pro Sekunde zu messen. Wir führen drei aufeinanderfolgende Entwicklungsschritte durch, um die Ergebnisse der vorangegangenen Bewertung zu verbessern. Ein Hauptziel ist es, den Prototyp so zu gestalten, dass er ohne Benutzereingriff funktioniert, während der Radfahrer fährt. Dies erfordert eine ausgeklügelte Signalverarbeitung, um Autos zu erkennen und genaue Entfernungen bei kurzen Tastverhältnissen zu messen. Außerdem sollen die gesammelten Daten über Bluetooth auf das Mobiltelefon des Benutzers übertragen und dort zwischengespeichert werden. Nach jeder Fahrt werden die gesammelten Positionen, Geschwindigkeiten und Entfernungen an eine skalierbare Infrastruktur gesendet, die eine kollaborative Datenerfassung zur eingehenden Analyse ermöglicht. Diese Daten können genutzt werden, um sicherere Routen zu identifizieren und notwendige Verbesserungen an Radwegen aufzuzeigen, um die Stadtentwicklung zu unterstützen, indem Straßen mit dem besten Kosten-Nutzen-Verhältnis priorisiert werden. Zu diesem Zweck haben wir einen Clustering-Algorithmus verwendet, um Straßen und Orte mit einer signifikant höheren Häufigkeit von engen Begegnungen zu finden. All dies wurde durch eine Reihe von Versuchsanordnungen validiert, die in einem zweimonatigen Einsatz des funktionalen Prototyps unter realen Verkehrsbedingungen gipfelten. Am Ende ergaben die Daten drei verschiedene Standorte. Auch wenn es an städtebaulichem Fachwissen mangelt, scheint zumindest der Grund, warum diese Orte identifiziert wurden, offensichtlich zu sein, was darauf hindeutet, dass die Daten und der Sensor vielversprechend sind. Die Arbeit schließt mit einer Diskussion der Grenzen des Prototyps und der zukünftigen Arbeit, die erforderlich ist, um ihn zu einem brauchbaren Produkt zu machen und die Anreize für die Benutzer zu erhöhen, ihn tatsächlich zu benutzen.

# ACKNOWLEDGMENTS

I’d like to thank everyone who gave me valuable feedback during the writing of this thesis. Special thanks go to Mr Müller for soldering the very tiny circuits when my handicraft skills were insufficient. The 3D printer, provided free of charge by Mr Moeller, made it possible to quickly prototype different ways of mounting the sensor and to evaluate the sensor in it’s intended environment. In memory of my father, who studied radio engineering in Darmstadt.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LISTINGS

# ACRONYMS

| | |
|---|---|
| PCR | Pulsed-Cohrent-Radar |
| FMCW | Frequency-Modulated-Continuous-Wave |
| DSR | Design Science Research |
| TOF | time of flight |
| DFT | Discrete Fourier Transform |
| FFT | Fast Fourier Transform |
| FMCW | Frequency-Modulated-Continuous-Wave |
| FOV | Field of View |
| RCS | Radar Cross Section |
| CFAR | Constant False Alarm Rate |
| CA-CFAR | Cell Averaging Constant False Alarm Rate |
| SNR | Signal to Noise Ratio |
| HWAAS | Hardware Accalerated Average Samples |
| Lidar | light imaging, detection and ranging |
| BLE | Bluetooth Low Energy |
| EIRP | equivalent isotropically radiated power |
| cut | cell under test |
| DBSCAN | density-based spatial clustering of applications with noise |
| OBS | OpenBikeSensor |
| ADFC | Allgemeiner Deutscher Fahrrad-Club |
| BIRCH | Balanced Iterative Reducing and Clustering using Hierarchies |

# Part I

# THESIS

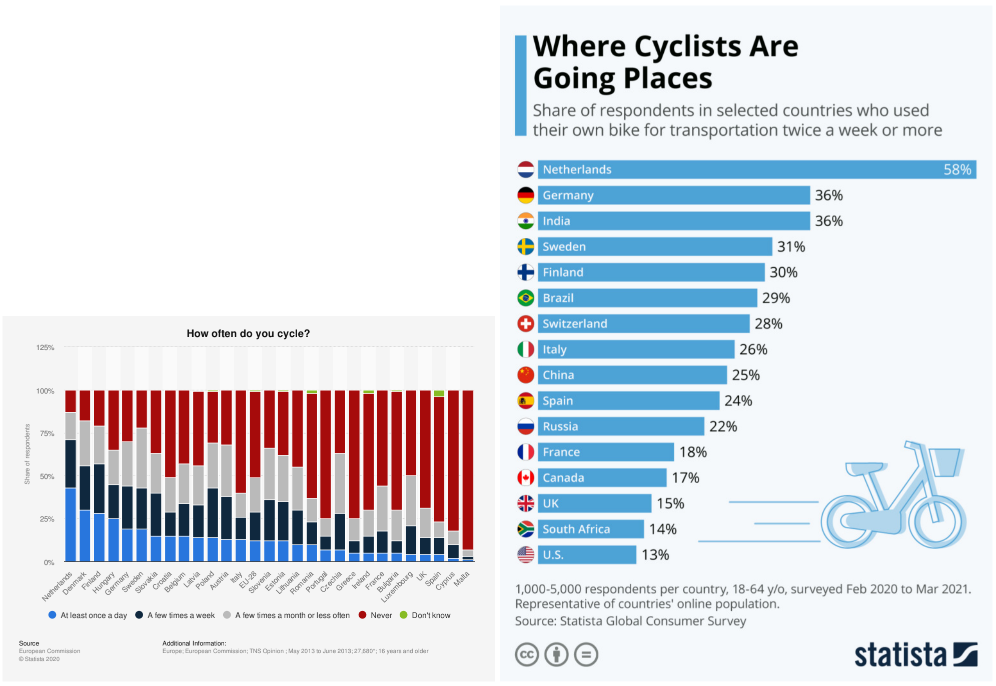
# 1 INTRODUCTION

The bicycle shows great potential to help building environmentally friendly and healthy cities [23]. One prominent factor that affects the usage of bicycles is the provided bicycle infrastructure. Compared to other European countries the german bicycle infrastructure is already in relatively good shape, however countries dedicated to providing a high quality bicycle infrastructure, such as the Netherlands, show a much higher daily bicycle usage. In Figure 1.1(a) about 70% of the people questioned in the netherlands claimed to use the bicycle at least a few times a week. In Germany close to 50% stated to cycle a few times a week. However a more recent survey from 2020 shown in Figure 1.1(b) concluded that only 36% of the german people claim to use the bicycle twice or more per week for transportation. This difference could be due to the slightly different question, or indicate a reduced bicycle usage. In both surveys the Netherlands is the country with the highest percentage of people cycling regularly. This could indicate that the bicycle infrastructure in the Netherlands is of higher quality and the people are more willing to use it. This is supported by the results of other researchers that concluded „The most important approach to making cycling safe and convenient in Dutch, Danish and German cities is the provision of separate cycling facilities along heavily travelled roads and at intersections, combined with extensive traffic calming of residential neighbourhoods" [26, p.523] Additionally, Destatis published in 2021 that in 71,9% of bicycle accidents in Germany, a car was the opponent of the bicyclist [1]. At the same time the bicyclist was responsible in only 25% of those accidents with a car involved. This could indicate that either the availability or visibility of bicycle infrastructure is not sufficient resulting in the bicyclists being overlooked while having to share a lane with cars. Besides the possible induced demand [20], resulting in increased usage of bicycles, there are more desirable side effects of improving the biking infrastructure in cities [22].

Projects such as the SimRaApp [1] and the OpenBikeSensor (OBS) [2] have already shown their potential in 2022 by winning the german bicycle price in the category service and communication [9]. Those projects mapped the bicycle infrastructure by measuring the distance of cars overtaking the bicyclist. Other projects tried to use public data sources to calculate the „bikeability of urban infrastructure" [15]. One of the most important factors determined was the availability of „biking facilities along main streets". This approach offers the option to dedicate the available funds into projects having the biggest impact. By identifying the areas where people already bike regu-

1 https://play.google.com/store/apps/details?id=de.tuberlin.mcc.simra.app
2 https://www.openbikesensor.org/

larly but lack critical infrastructure the impact of the available funds can be maximized.



(a) (b)

Figure 1.1: Survey results from 2013 (a) [11] and 2020 (b) [17]. The Netherlands lead in every survey in regards to regular bike usage. This correlates with the high quality of the bicycle infrastructure in the Netherlands.

## 1.1 OBJECTIVES AND METHOD

The objective of this thesis is to develop a prototype using the Design Science Research (DSR) method [32]. The development process will be iterative, with each cycle evaluated to guide the next iteration and add new knowledge to the corpus. Figure 3.2 in [16, P.27] illustrates a typical DSR iteration from problem identification to conclusion and knowledge extraction.

To ensure the success of the project, it is important to identify relevant metrics before beginning the implementation phase, as emphasized by [32, p. 18]. Furthermore, research questions and experiments that guide the project must be defined, and can be either quantitative (e.g. „What is the range of our sensor?“), exploratory (e.g. „Where are dangerous areas?“), or hypothesis-based (e.g. „During the commute hours, cycling is more dangerous“).

Given the time constraints of a 6-month timeline, the development schedule includes the following segments:

1. Initial research and hardware selection

2. First development iteration

3. Second development iteration
4. Third development iteration
5. Data collection while setting up the backend
6. Data analysis and remaining tasks

The structure of this thesis follows this schedule accordingly.

## 1.2 METRICS

In DSR, it is crucial to define a set of metrics and experiments to evaluate the working prototype developed in each iteration. This ensures that the development is on the right track, potential issues are identified, and knowledge is extracted from the evaluation. In this section, we define the metrics for evaluating the working prototype of the sensor developed in this study.

### 1.2.1 *Distance and Range*

The first metric we consider is the accuracy and range of the distance measurements. The sensor should be able to measure distances up to $3m$, which is the typical street width in Germany where the experiments will be performed. Depending on the speed limit, the street width may be regulated to be between $2.5m$ and $3.5m$. The error in accuracy should be below 5%.

To evaluate this metric, we will use a static experimental setup. The sensor should be placed in front of relevant targets to obtain a reference distance measurement. Then, a series of measurements should be obtained at each distance, starting from $0.5m$ and ending with $3m$, incrementing in $0.5m$ steps. The error should be calculated by comparing the reference measurement with the obtained measurements.

### 1.2.2 *Vehicle Detection*

Since one of the main goals is to automatically detect passing vehicles, instead of having the user manually trigger the measurement, the sensor should be able to detect vehicles automatically. The sensor should be able to detect vehicles with a speed of at least $50km/h$, the common speed limit in cities. The detection should be reliable, i.e. the sensor should not falsely detect a vehicle when there is none. A false negative is acceptable, i.e. the sensor should not detect a vehicle when there is one, but it would be desirable to minimize this error. In order to evaluate this metric an experimental setup should be used, where the sensor is mounted on a bicycle and cars are driven past the sensor, while the timestamps of the measurements are recorded. The detection should be evaluated by comparing the timestamps of the measurements with the timestamps of the cars passing the sensor.

### 1.2.3 *Real-world Environment*

Since the sensor is intended to be used in traffic, we need to evaluate its use in the real world. To achieve this, the sensor should be mounted on a bicycle, and a predefined route should be test-driven multiple times, featuring different cycling infrastructure. A camera should record the trip to obtain a reference for passing cars and evaluate unexpected measurements and identify possible flaws. This experiment evaluates the reliability of the sensor in a real-world environment and additionally allows to show possible advanced analysis methods of the collected data, currently not done by the OBS project.

# 2 RESEARCH

This chapter provides an overview of the different types of sensor technologies available and their principles, including their advantages and drawbacks. Depending on the selected sensor, this chapter will also present the relevant types of signal processing algorithms that may be required to achieve the goal of this project. For a brief overview of the different types of sensor, see [30, Figure 20], which offers spider diagrams for light imaging, detection and ranging (Lidar), Radar, Ultrasonic sensors, and cameras.

## 2.1 TOF-SENSORS

A common method for digitally measuring distance is through time of flight (TOF) technology, which involves measuring the time it takes for a signal to travel from the sensor to an object and back. This is usually done by emitting a signal that is reflected by the object and received by the sensor. The time it takes for the signal to travel from the sensor to the object and back is then used to calculate the distance based on the speed of the signal. The distance can be calculated using the formula $d = \frac{t}{2} \cdot v$, where $t$ is the time measured and $v$ is the propagation speed of the emitted signal.

### 2.1.1 *Ultrasonic sensors*

Ultrasonic sensors are a type of TOF sensor that use sound waves as the emitted signal. They work by emitting a sound wave, which is reflected by an object and received by the sensor. The time it takes for the sound wave to travel from the sensor to the object and back is then used to calculate the distance based on the speed of sound. The distance can be calculated using the formula $d = \frac{t}{2} \cdot v$, where $t$ is the time measured and $v$ is the propagation speed of sound in air. The approximate speed of sound, which is sufficient for distance calculation is $v \approx 340 \frac{m}{s}$. This means that the signal travels approximately $10ms$ for a distance of $1.7m$.

One advantage of ultrasonic sensors is that they are relatively low cost, with prices ranging from a few euros per unit. They are also accurate and precise enough for many use cases, including the presented goal in Chapter 1, with a deviation in distance measurements of around a few millimeters [7, Fig. 3]. However, a major drawback of ultrasonic sensors is that they require a clear line of sight and have limited resolution. Additionally, ultrasonic sensors emit sound waves that cannot be heard by humans, but may be audible to animals such as cats and dogs. This can potentially cause stress or irrational behavior in these animals, leading to potentially dangerous situations.

### 2.1.2 *Laser sensors*

Laser sensors are a type of TOF sensor that utilize laser beams as the emitted signal. They can be divided into two types: simple laser sensors and Lidar sensors. Simple laser sensors emit a single beam in a fixed direction and measure the time until it returns. Lidar sensors, on the other hand, use a rotating mirror to emit beams in multiple directions, allowing them to measure the distance in various directions and angles and produce a representation of the surrounding area. While sophisticated Lidar sensors offer very fine resolution, they tend to be relatively expensive and bulky due to the rotating mechanism. Like ultrasonic sensors, laser sensors also require a clear line of sight.

Relevant signal processing algorithms for laser sensors may include doppler lidar, range calculation, and error correction [31]. Doppler Lidar utilizes the doppler shift of the reflected light to calculate the movement speed of the object along the line of sight, while range calculation involves using the time of flight information to calculate the distance to the object. Error correction algorithms may be used to improve the accuracy of the distance measurements by taking into account factors such as the wavelength of the laser and the refractive index of the medium through which the signal is traveling. For the intended use case an approximation of $c \approx 3 \cdot 10^8 \frac{m}{s}$ is expected to be precise enough. While sophisticated Lidar sensors are relatively large and expensive, simple laser sensors on the other hand, are cheap and could be considered as a viable option.

### 2.1.3 *Radar sensors*

Radar sensors are a type of TOF sensor that use radio waves as the transmitted signal, and were originally patented in 1904. One advantage of radar sensors is that the refractive effect of the Earth's atmosphere can be neglected in the measurements due to its small effect, so that the approximate speed of light $c \approx 3 \cdot 10^8 \frac{m}{s}$ can be used as the value for $v$ in the distance calculation formula $d = \frac{t}{2} \cdot v$ [28, p.3]. However, a more accurate value for $v$ may be required for very high accuracy. Compared to the ultrasonic sensor, the signal propagation speed of radar sensors is much higher, which means that the time it takes for the signal to travel from the sensor to the object and back is much shorter. For example, the propagation time for a distance of $1.5m$ is now only $10ns$. Depending on the type of radar, the transmitted signal may be reflected by the object before the original transmission is complete. For pulsed coherent radars, this means that there is a direct scattering loss that results in a blind spot near the sensor, depending on the pulse length. A longer pulse length is required for better Signal to Noise Ratio (SNR) and for measurements over longer distances. As a general rule for Pulsed-Cohrent-Radar (PCR), better SNR is associated with lower resolution and lower measurement frequency. A Frequency-Modulated-Continuous-Wave (FMCW) radar, on the other hand, continuously modulates the frequency and is able to receive the reflected signal even if the transmission is not complete. This is achieved by filtering the received signal, which is based on the subtraction of the transmitted signal from the received signal. This is possible because the received signal has a different frequency than the signal just transmitted.

Radar sensors are available in a wide range of wavelengths, allowing an optimal approach for the particular application. The wavelength of the emitted signal can affect both the measurement accuracy and the power consumption of the sensor. The energy of a photon is directly proportional to its frequency, with the relationship $E = h \cdot f$, where $h$ is Planck's constant. Given the fixed propagation speed of electromagnetic waves, the frequency of the signal is directly related to the wavelength $\lambda = \frac{c}{f}$. A longer wavelength requires less energy but has a lower intrinsic precision [27].

There are several methods for detecting targets using radar, as described in [28, chap. 6]. All of these methods assume that the reflection has a reasonably high SNR. The idea is to compare the received reflection with a fixed or relative threshold. The expected reflection depends on the material, distance, size and shape of the target. While the relative dielectric constant of materials, and thus the reflectivity, depends on the frequency of the signal, metal can be considered absolutely reflective for all commercially available frequencies. Furthermore, the effective radar cross-section of simple metal objects, such as the side of a car, can be approximated by $\sigma = \frac{4\pi A^2}{\lambda^2}$, where $A$ is the area of the metal side when hit near 90 deg. With this reflection property, the expected received signal $P_r$ can be calculated using the radar equation $P_r = \frac{P_t \cdot G_t \cdot \sigma \cdot A_r}{(4\pi)^2 \cdot R^4} \cdot \frac{1}{L_t \cdot L_{atm}}$, where $P_t$ is the transmitter power, $G_t$ is the gain of the transmitter and $R$ is the distance to the target. $L_t$ and $L_{atm}$ are constant losses due to the antenna $L_t$ and $L_{atm}$ [21]. This received signal can then be compared to either a fixed threshold or a threshold calculated or measured based on the noise level. Since the signal propagates in a three-dimensional space, the expected reflectance decreases with distance according to the inverse square law represented by the term $R^4$ in the equation. The distance of the object can be taken into account as a factor when calculating the threshold. Since the radar equation is affected by several radar properties, there is no general guideline for the configuration of the radar. All parameters, such as gain, transmit power and others, must be considered and evaluated individually for each application. Advantages and disadvantages must be carefully weighed to achieve the desired results. For example, while high transmit power significantly improves the SNR, it increases the pulse length while keeping the system's power consumption constant. This means a larger blind spot for PCR and fewer measurements in a given time. In Section 2.2.1, the use of Cell Averaging Constant False Alarm Rate (CA-CFAR) and special considerations for our use case that had to be taken into account are explained in more detail.

In addition, some systems allow discrimination between different targets and background/clutter by calculating the Doppler shift of the received signal.

Similar to Lidar sensors, it is possible to use the Doppler shift to determine the radial velocity of the target. This can be used to determine whether a target is moving towards or away from the sensor, which in turn can be used to determine whether the target is part of the static background. However, calculating the Doppler shift requires a series of measurements, and determining the radial velocity requires relatively expensive signal processing in the form of Discrete Fourier Transform (DFT) [21, p. 35].

### 2.1.4 *Camera sensors*

Cameras are widely used for computer vision tasks because of their ability to capture images of the environment, which can be processed to determine the presence of objects. For instance, in the field of transportation, cameras are used for traffic management and control, where they can detect vehicles, measure their speed, and monitor traffic flow [6].

In addition, stereo cameras have been employed to calculate the distance to objects by comparing the images captured by two cameras, making them suitable for autonomous vehicles [33]. This technology has already been implemented in advanced driver-assistance systems (ADAS), where stereo cameras are used for lane departure warning, adaptive cruise control, and collision avoidance [5].

Despite the potential benefits of using cameras for various applications, there are several factors that make it a less attractive option. Firstly, the relatively high cost of cameras make it a less attractive option compared to the OBS. Additionally, legal restrictions regarding privacy concerns and data protection can limit the use of cameras in certain situations, such as in public spaces or areas where individuals have a reasonable expectation of privacy. Furthermore, the setup required for cameras can be cumbersome, as they require a fixed position and a clear view of the environment. This basically restricts the possible positions of the camera to the bike rack.

In conclusion, cameras are a well-established and versatile technology for computer vision tasks, but their applicability may depend on various factors such as cost, legal considerations, and setup requirements. Nonetheless, with ongoing advancements in camera technology and image processing techniques, cameras are expected to remain a possible solution for the challenges of determining the quality and availability of biking infrastructure.

## 2.2 ALGORITHMS

Due to the advantages and disadvantages mentioned above, we chose radar technology as the sensor of choice for our prototype. Table 3.1 compares different radar platforms and Chapter 3 explains in detail why the XM122 IOT module was chosen and how it was used. For this section we'll only discuss the information relevant to the A111 60Ghz radar, a PCR, as not all algorithms are suitable for this type of radar.

### 2.2.1 *Presence Detection*

In the field of radar, there are several signal processing algorithms that are used to detect targets. The choice of algorithm depends on the type of radar, the targets to be detected and the computing resources available. [28] provides an overview of different algorithms, their advantages and disadvantages. Since we have chosen a PCR for our application, we will focus on the algorithms suitable for PCR systems.

One of the simpler algorithms for performing detection tasks is the fixed threshold algorithm. In this method, a predetermined or boot measured fixed value is used to check whether a detection has occurred. The advantage of this algorithm is that it has a fixed running time. Since the radar returns $n$ reflection samples, each must be compared to the threshold exactly once, resulting in a running time of $\mathcal{O}(n)$. However, the fixed threshold algorithm has several drawbacks. The threshold must be chosen carefully and may only work under the initial conditions for which it was chosen. If the threshold is too high, the algorithm will not detect any targets, and if the threshold is too low, the algorithm will detect targets that are not there. This is particularly problematic when the radar is used in changing conditions. Changes in environmental conditions can cause the sensor to make false detections. In addition, a fixed threshold is tied to the sensor's settings. Changing the pulse length, for example, will result in a different amplitude for an otherwise unchanged situation.

Instead of using a fixed threshold, it is possible to use a Constant False Alarm Rate (CFAR) algorithm. These algorithms dynamically recalculate the threshold to achieve a CFAR, hence the name. A very prominent type of CFAR algorithm is the CA-CFAR [28, p. 337]. This algorithm calculates a new threshold for each cell under test (CUT). This is done by summing all the other cells and using that sum as the threshold. This has the advantage of increasing the threshold as the noise level increases. This works particularly well when the noise is homogeneous. However, it has the disadvantage of being much more computationally expensive than the fixed threshold algorithm. Since the sum must be recalculated for each cell to be tested, the runtime for a signal with $n$ cells effectively changes to $T(n \cdot (n-1)) \in \mathcal{O}(n^2)$ for all useful measurement series. There are more sophisticated variations, but the idea of determining the threshold remains the same and is only slightly improved by applying different types of filters. For example, calculating individual thresholds for the left and right sides of the CUT, ignoring a number of cells next to the CUT called the CFAR guard, or even a combination of several CFAR algorithms [21, ch. 9].

### 2.2.2 *Distance measurement*

Distance measurement is implemented using the calculation described in Section 2.1. Due to the properties of metal, the flat metallic side of a car should have a very strong Radar Cross Section (RCS). The strongest peak should be close to or at the shortest distance from the sensor. This strong peak should allow a very accurate distance measurement as the RCS should be relatively narrow. As described in Section 2.1.3, calculating the distance to one or more targets is relatively straightforward as the returned data series are in the time domain. Once a detection has been made, the distance correlates with the propagation time of the signal. The numerical value can be calculated using $d = d_0 + i \cdot \Delta d$, where $d_0$ is the offset to the first cell for a PCR, mostly determined by the blind spot caused by the pulse length, $\Delta d$ is the distance between two sample points, also known as the resolution, and $i$ is the index of the sample point where the detection occurred.

### 2.2.3 *Fast Fourier Transform*

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT) of a sequence of data points. The DFT is a mathematical transformation that decomposes a sequence of data points into their individual frequency components, which can be used to represent the data in the frequency domain. In our case, these are the discretely sampled amplitudes of the reflection [28, ch. 5.3]. The FFT is a faster and more efficient version of the DFT and is widely used in signal processing applications such as audio and image processing.

The FFT algorithm works by recursively dividing the input data into smaller and smaller pieces. „The FFT algorithm achieves its efficiency by replacing the computation of one large DFT with that of several smaller DFTs. “ [24, p.86]

The FFT algorithm has a time complexity of $O(n \log n)$, which makes it much faster than other methods of computing the DFT, such as brute force, which has a time complexity of $O(n^2)$. This makes the FFT algorithm particularly useful for analysing large data sets or for applications that require real-time data processing. For radar, this allows a so-called range-doppler map to be calculated from multiple radar pulses. Each pulse provides information about the distance to the target, and by transforming the individual sweeps/pulses into the frequency domain, we can calculate the frequency difference between the reflected and received pulse. This frequency change is caused by the radial velocity of the target. The radial velocity $v$ can be calculated using $v = 0.5 \cdot c \cdot \frac{f_D}{f_t}$, where $f_D$ is the frequency difference between the transmitted and received pulse, $f_t$ is the frequency of the transmitted pulse, and $c$ is the speed of light [21, ch. 2-6]. This calculation only works as long as the frequency of each pulse is twice the frequency difference. This means that in the case of pulsed radar, the pulse length must be long to cover greater distances and have a better signal-to-noise ratio, but the pulse length must be short enough to measure the expected radial velocities.

One possible implementation is the recursive depth-first approach proposed in [19, Listing 1.] and commonly known as the Cooley-Turkey FFT algorithm. The algorithm achieves its efficiency by recursively splitting the input into smaller and smaller pieces and then combining the results of the smaller pieces. This is done by calculating the DFT of the even and odd indices of the input data and then combining the results. However, the implementation presented does not seem suitable for the embedded XM122 due to the new memory allocation for each recursion. It may be possible to investigate an implementation that rearranges the calculations to avoid recursion and memory allocation.

### 2.2.4 *Clustering Algorithms*

Since the goal of this project is to identify dangerous areas or roads, a logical approach would be to use some form of clustering algorithm to identify these areas. Clustering is a common technique for explorative data analysis. This would allow us to identify areas where there are a lot of close encounters. Since we will be working with geographic data, the research will focus on clustering algorithms that are better suited to handle geographic data. [14, ch. 07, p.149] gives a very good overview of different clustering algorithms and their use cases.

#### 2.2.4.1 *k-means*

k-means is probably one of the most popular clustering algorithms. k-means divides the given data points into exactly $k$ clusters. This is done by selecting $k$ random data points as initial cluster centres, and then assigning the data points to the nearest cluster centre. This makes k-means a distance based clustering algorithm. The new data may affect the cluster centre, which has to be recalculated. This is repeated until the clusters no longer change. K-means is interesting because it is suitable for large datasets, which is what we expect to end up with in production, as k-means is reasonably scalable and an efficient algorithm. The big problem with this algorithm is that it requires the user to specify the number of clusters $k$. One approach is to use the elbow method to determine the correct number of clusters, but this also means that the algorithm has to be run for each $k$ that needs to be tested. Furthermore, the algorithm is susceptible to outliers because they affect the mean values of the clusters [14, p.157]. Additionally k-means will always assign all points to a cluster. This is in contrast to the intended use case of identifying areas with a higher measurement density than other areas.

#### 2.2.4.2 *DBSCAN*

Since we do not know the number of clusters to expect, one approach would be to use density based clustering algorithms. One possible algorithm would be density-based spatial clustering of applications with noise (DBSCAN).

DBSCAN will determine the number of clusters by creating a cluster for each point that has at least a certain number of neighbours *MinPts* within a certain distance $\epsilon$. These points are called core points. A non-core point within $\epsilon$ of a core point is assigned to the same cluster as the core point and is considered directly reachable. A point within $\epsilon$ of a directly accessible point is considered to be densely accessible. With this set of rules, the initially identified clusters will naturally grow and merge until all points are either assigned or considered outliers [14, p. 171]. The big advantage for our use case is that we do not need to specify the number of clusters, the algorithm will identify them automatically. Furthermore, the parameters for this algorithm can be naturally translated into the real world application. For example, $\epsilon$ can be translated to the real world distance between two points, e.g. in km. Additionally, *MinPts* can be chosen to correlate with a certain percentage of all overtaking maneuvers that were considered dangerous. This allows sensible parameters to be selected for the algorithm without knowledge of the total collected data that may be available at the time of algorithm selection. To reduce the overall runtime it might be possible to run the algorithm twice, once with a high $\epsilon$ and *MinPts* to identify one cluster per city, allowing a partitioning of the data into smaller subsets. Then cluster each subset with a much smaller $\epsilon$ to perform the final clustering highlighting the danger zones within each city. We would expect this to reduce the runtime of the algorithm because the high $\epsilon$ would allow a fast assigning of points to a cluster in the large dataset, and the low $\epsilon$ on the smaller dataset would not impact the runtime as much as operating on the entire dataset because there are less distances to compare to. This might not even be necessary because DBSCAN is generally already suited for large scale datasets.

#### 2.2.4.3 *BIRCH*

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) separates it from the previous two algorithm in the aspect that it does reduce the amount of data to be processed by initially creating a summary. Furthermore the fact that BIRCH initially summarizes the datasets into subclusters before reducing the information to retain of this cluster [34]. This makes it suitable for large datasets because it summarizes dense areas. A common use case for BIRCH was the separation and compression of images by identifying pixel clusters. BIRCH can however be adapted to use the clustering approach of other algorithms by using the BIRCH summary as input for the clustering algorithm. Since we already presented the idea of separating the data into subsets, one per city each, BIRCH might be the ideal algorithm for this goal, especially because it can process large datasets even with limited memory. Just as k-means however it is necessary to specify the expected amount of clusters beforehand. For the separation of the data into subsets this should be fine however, as the amount of participating cities can just be seen as an upper limit to avoid splitting one city into multiple subsets. Having two cities in the same subset does not matter as the final clustering will still be able to identify the danger zones within each city.

## 2.3 RELATED WORK

A number of projects have been carried out in the field of bicycle accessories and research. This section gives an overview of the most relevant projects.

### 2.3.1 *Open Bike Sensor*

The Open Bike Sensor is a project based on the „Project Radmesser“[1] launched in 2018. These projects use a TOF sensor, currently either an HC-SR04 or JSN-SR04T ultrasonic sensor, to measure the distance to cars. However, the user has to trigger each measurement individually. With the OpenBike sensor, this is done by pressing a button on the bike's handlebars. Placing the burden of ensuring high data quality on the user would require a significant change in the attractiveness of the bike, and the user would need to be aware of the system. This could distract the user and lead to accidents. In addition, the OpenBike sensor has to be purchased, assembled and programmed manually, with instructions requiring basic soldering skills and some computer affinity[2]. This may be one of the reasons why the OBS isn't as well known as it could be. As citizen science depends on the participation of the general public, this is a significant drawback. As the project has been running since 2020, they have already been able to demonstrate the value of the data in a project in Berlin. The data was used to compile various statistics and create maps that provide insights into the situation of cyclists in Berlin.

### 2.3.2 *Garmin Rear Radar*

Garmin has a product called „Varia“[3]. This product uses radar to detect approaching cars and warn the cyclist. While this device can automatically detect cars, it is not able to measure the lateral distance to passing cars. While it can tell the speed and distance of an approaching car, it does not really give an indication of the cyclist's safety, as a fast car could pass the cyclist at more than the required safety distance. In addition, the price of these devices is €200 or more, which is a significant barrier to entry.

### 2.3.3 *Street condition classification*

In related work, a pulsed coherent radar has been used to perform material and state detection to identify road conditions. This was done by modelling different material properties with the reflection and scattering behaviour as well as the dielectric properties of the materials. In this way they were able to use the amplitude to distinguish between wet, snowy and dry asphalt [18, p.40]. At the same time they claim in their conclusion that the A111 radar is limited in it's range in a fast moving environment.

1 https://interaktiv.tagesspiegel.de/radmesser/kapitel7.html
2 https://www.openbikesensor.org/docs/hardware/v00.03.12/build-instructions/
3 https://www.garmin.com/de-DE/p/601468

### 2.3.4 *Stationary radar for bicycle safety*

Another project is a stationary professional 24GHz FMCW radar that can be used to map the situation at hand. While this is a very nice technique to digitise and analyse already known problem spots, the device is too big to be used on a bicycle. [13]

# 3 THE ARTIFACT

This chapter describes the artifact we created, its components, and why those components were chosen. It also describes the implementation of the artifact and the challenges encountered during the three implementation iterations. The first implementation was based on the distance detector provided by the SDK, the second implementation was based on the envelope service provided by the SDK, and the third implementation was based on the power bin service. Each implementation has been improved based on the evaluation of the previous one. All implementations can be flashed and tested following the instructions in [3, 2.1 and 2.2].

Figure 6.2 shows an abstraction of the setup, starting with the radar, transmitting the measurements via BLE to the Android app, adding the location information and storing the data. At the end of the tour, the data is uploaded to the backend where it is pre-processed and stored in a dynamodb. A node.js web application is used to visualise the data, and additional lambdas are used to simplify database access. This chapter focuses on the implementation of the radar signal processing.

## 3.1 HARDWARE SELECTION

Based on the advantages and disadvantages presented in Section 2.1, we came to the conclusion that pulsed coherent radar is a promising technology without violating people's privacy. Despite the drawbacks presented in the paper referenced in Section 2.3.3, we decided to go with this type of technology. Research has shown that sophisticated signal processing should be able to compensate for the poorer SNR when working in a long range, rapidly changing environment. Due to the optimal dielectric properties of the intended target and the near optimal 90° orientation, we still expect the radar to perform reasonably well. Assuming that most cars are made of metal, combined with a large and relatively uniform surface, it is expected that there will be a significant difference in reflection strength compared to hedges, pedestrians, wooden walls and more.

There are several possible radar chips available for purchase. Table 3.1 summarises the information available in a tabular format. We have chosen to use the XM122 iot 60Ghz PCR radar chip developed by acconeer [25]. Mainly because this chip has a very competitive price of only $\leq$ 35€ each, contains an nrf52840 cpu and a Bluetooth antenna in a very space efficient circular layout. A front and rear view of the sensor is shown in Figure 6.4. This package offers almost everything you need at a price close to or even below that of the OBS. 60Ghz corresponds to a wavelength of $\approx 5mm$, which allows for very energy efficient radar pulses at short ranges, as discussed in Chapter 2.

Table 3.1: Manufacturer specifications of possible 60GHz radar platforms, as far as they are available.

| | XM122 | DEMO BGT60LTR11AIP | XE121 | AWR6843AOPEVM |
|---|---|---|---|---|
| **Frequency** | 60GHz | 60GHz | 60GHz | 60GHz |
| **Radar Type** | PCR | FMCW | PCR | FMCW |
| **Range** | up to 7m | up to 14m | up to 20m | 12-15m for people detection |
| **Resolution** | 0.5mm | limited by measurable frequency difference | ? | ? |
| **Bluetooth** | Yes | No | No | No |
| **CPU** | nrf52840 | external (usb) | ? | ? |
| **Clock Speed** | 64MHz | - | ? | 600Mhz |
| **RAM** | 256kB SRAM | - | ? | 768KB |
| **Flash** | 1MB | - | ? | 1.4MB |
| **Price** | ≈ 35€ | ≈ 324$ | ≈ 191€ | ≈ 135$ |
| **Size** | 33mm circular | 64 mm x 25.4 mm size | ? | ? |

While a FMCW radar might be more accurate and faster with a better SNR, it comes at a much higher price. A possible chip of similar size to the XM122 costs $\approx$ 130€ [2].

Both sensors have the advantage that they can be used without restrictions as long as the equivalent isotropically radiated power (EIRP) is below $0.1W$ on average, as regulated by the FCC in 2021 [10].

For these reasons we have chosen the XM122 for our prototype, but other radar chips may be equally or more suitable. Once the platform was selected, the first step was to evaluate its basic functionality and characteristics. An important aspect outlined in Chapter 2 was the fact that the amplitude of the reflections would be used to differentiate between cars and other objects. To see how well this worked and at what distance, we connected the sensor to a PC to stream the data, starting with the default settings for each mode supported by the sensor.

At the time of writing, the SDK for the A111 has been split into 4 different types of analysis (services).

- Power Bins Service - This service provides a histogram of the power of the received signal correlated with the distance to the target.
- The Envelope Service - This is a finer grained version of the Power Bins Service. A sequence of radar pulses is transmitted, sampling the received reflections every $0.5mm$, averaging and time smoothing the result to produce a stable signal.
- IQ Service - This service is not used in this project as it is used to detect small movements.
- Sparse Service - The raw signal is sampled every $\approx 6cm$. This mode produces a series of arrays, each representing a sweep. The arrays contain the raw data from the radar chip and can be used to implement custom algorithms such as FFT to extract the Doppler shift. Due to the large number of sweeps, this introduces a significant error in distance measurements.

These services differ in computational complexity, pulse length, sampling rate and other performance metrics. In particular, pulse length is an important factor in radar performance. The longer the pulse, the more energy is consumed and the lower the resolution of the range measurement. However, the additional energy allows a longer sweep length by improving the SNR, but also increases the blind spot near the sensor due to possible direct leakage. Using the Power Bin service, we positioned the sensor in front of various targets and recorded the amplitude of the reflections. As expected, the amplitudes vary with material and distance. In general, we observed in the raw data that a human at 1m produces a reflection that is as strong as a car at 2.5m. A car at the same distance as a person produces a reflection about 3-4 times as strong. While a brick or wooden wall will produce a stronger reflection than a human, it is still only half as strong as a metal object. Furthermore, at a distance of around 1.5m, reflections from non-metallic objects become so weak that they cannot be reliably distinguished from background noise. Figure 3.6 illustrates this observation by plotting the average amplitude measured per distance and target.

## 3.2 FIRST IMPLEMENTATION

For the chosen hardware, the first implementation was based on the distance detector provided by the SDK. This abstraction is implemented on top of the envelope service. The code can be referenced in Listing 6.2. With the SDK implementation, the code is quite simple. As long as a target is detected, send the measured distance. If no target is detected, send a distance of 0.

This is the highest level of abstraction provided by the SDK and the easiest to use. To use it, you need to set a threshold and choose how to sort the detected objects. Valid sorting strategies are sorting by occurrence, reflectivity or relative reflectivity as a function of distance. The maximum number of detections returned must also be set. Finally, the radar-specific settings such as start point, sweep length, gain and other settings must be set according to the requirements defined in Chapter 1.

However, the resulting radar performance was suboptimal for our use case. Although the distance measurements appeared to be very good, it did not take much evaluation to conclude that the distance detector was not suitable for our use case. This was mainly due to the low frame rate of less than $6Hz$. This is mainly due to the fact that the Distance Detector uses the Envelope Service, which combines a series of successive frames to produce a stable and accurate signal optimised for distance measurements. To ensure reliable detection of fast moving objects, the sampling rate must be high enough. According to the sampling theorem, the sampling rate must be at least twice the highest frequency component of the signal. Assuming that cyclists pedal at $25\frac{km}{h} \approx 7\frac{m}{s}$ and cars overtake them in a city at a speed of less than $50\frac{km}{h} \approx 14\frac{m}{s}$, it can be assumed that the speed difference is usually less than $7\frac{m}{s}$. Considering the measurement angle of the selected sensor and assuming a symmetrical setup, the possible measurement duration $t$ can be described using $h$ as the height of the spanning triangle shown in Figure 3.1,

$\Delta v$ as the velocity difference and $\alpha$ as the measurement angle, as shown in Figure 3.1, the following relationships can be derived:

$$s = 2 \cdot h \cdot \tan(\frac{\alpha}{2}) \tag{3.1}$$

$$t = \frac{s}{\Delta v} \tag{3.2}$$

$$d = \frac{h}{cos(\frac{\alpha}{2})} \tag{3.3}$$

These functions are relevant to the detection algorithm because they describe the assumptions made previously. For example, the effect of a car approaching and then moving away can now be compared to the expected values calculated with Equation 3.2. Let's say a minimum distance of 1m was measured, then we can calculate the expected maximum distance when entering the Field of View (FOV). $d = \frac{1}{cos(40)} \approx 1.3m$. It is also very useful to be able to calculate the expected duration of a car in the FOV using Equation 3.2. Assuming a minimum distance of $0.5m$ and a maximum $\Delta v$ of $7\frac{m}{s}$, we can calculate an expected lower bound for the duration $t_{(0.5)} = \frac{2 \cdot 0.5m \cdot \tan(40)}{7\frac{m}{s}} \approx 0.12s$. This information is relevant because it means that our implementation must achieve a measurement rate of at least $16Hz$ if we are to be able to reliably measure a passing vehicle.

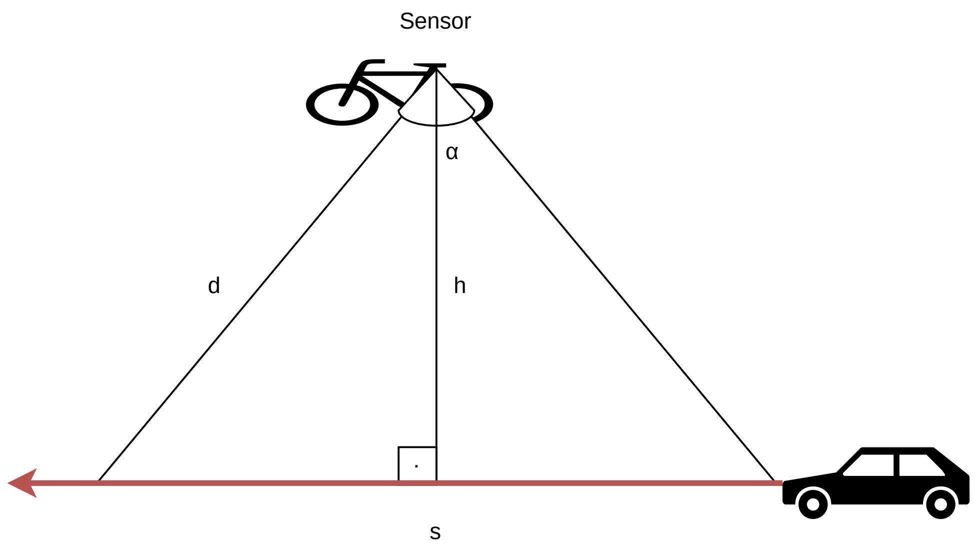


Figure 3.1: Sketch of the measurement angle of the sensor and it's correlation to the measurement duration.

Another minor aspect is that the implementation does not allow a distance dependent threshold. This means that the threshold is the same for all distances. This is not optimal because non-metallic targets can produce a reflection at close range that is similar in strength to a metallic object at distance. Therefore, the distance detector would detect non-metallic objects at close range. This also leads to random background noise spikes exceeding the threshold and triggering a false positive measurement. This problem could be solved by requiring a minimum number of measurements to be considered a valid measurement. However, this would require a much higher measurement rate and would not solve the problem of non-metallic objects. In addition, increasing the measurement rate will require a significant sacrifice in either sweep length or pulse length. Both changes would result in a reduction in range. The reduced pulse length would also worsen the SNR.

## 3.3 SECOND IMPLEMENTATION

Based on the observations made in the previous section, we decided to implement the distance detector in a different way, hoping to achieve a higher frame rate. This implementation was also based on the envelope service. This allowed much finer grained access to the radar settings. In addition to sweep range and pulse mode, running the envelope service directly allowed settings such as Hardware Accalerated Average Samples (HWAAS) and other settings to be adjusted. Reducing the HWAAS to 1 resulted in a less stable but much more sensitive measurement, as objects that were only briefly in the FOV were no longer averaged. Furthermore, instead of using a fixed threshold, we have now implemented a CA-CFAR variant that takes the current distance into account. This implementation can be referenced in Listing 6.3. This implementation barely achieves the required update rate. Still a massive improvement over the previous implementation. We did not manage to get above $12Hz$, mainly due to the fact that the required sweep length is sampled every $0.5mm$ in the envelope service. This means that a sweep length of $2700mm$ results in 5400 individually sampled points. As a result, the distance measurement is expected to be very accurate. In addition, the doubled frame rate can be set to 2 and is still no slower than the previous implementation.

### 3.3.1 *Distance evaluation*

To evaluate the accuracy, precision and range of this implementation, we performed the first experiment defined in Chapter 1.

Figure 3.2 shows the individual measurements and their amplitude, again confirming the initial observations that the amplitude is significantly different for metal objects and therefore suitable for differentiation using distance and amplitude dependent thresholds. The lower amplitude near the trailer was caused by the setup error. The bicycle was not standing upright, so the sensor had a significant part of the FOV occupied by the tyres, resulting in a lower overall amplitude. The graphs in Figure 3.3 show the absolute and relative measurement errors. After examining the data, we have concluded that this error is caused by our experimental setup. Therefore, we can't infer the real accuracy of the envelope implementation. But even if the setup was perfect, the largest error was $10cm$, which would still be acceptable.

Another indicator supporting this explanation, is that an analysis of the data showed, that the spread between the minimum and maximum distance measurement, in a single measurement series, was $6cm$. The aggregated results can be seen in the Table 6.1. Even when using the worst case in our measured data as the measurement error, the implementation was able to measure the distance of metal objects up to $3m$ with the required accuracy. The measurement series for non target objects, e.g. the wall shows a significant measurement error because the reflection produced was not strong enough above 1.5m to be reliably detected, resulting in random locations being identified as the highest amplitude, explaining the measurements at random distances with an amplitude at threshold. This is expected behavior as the assumption that non metal objects, e.g. a wall, will produce significantly weaker reflections allowing to filter them out in the final implementation.

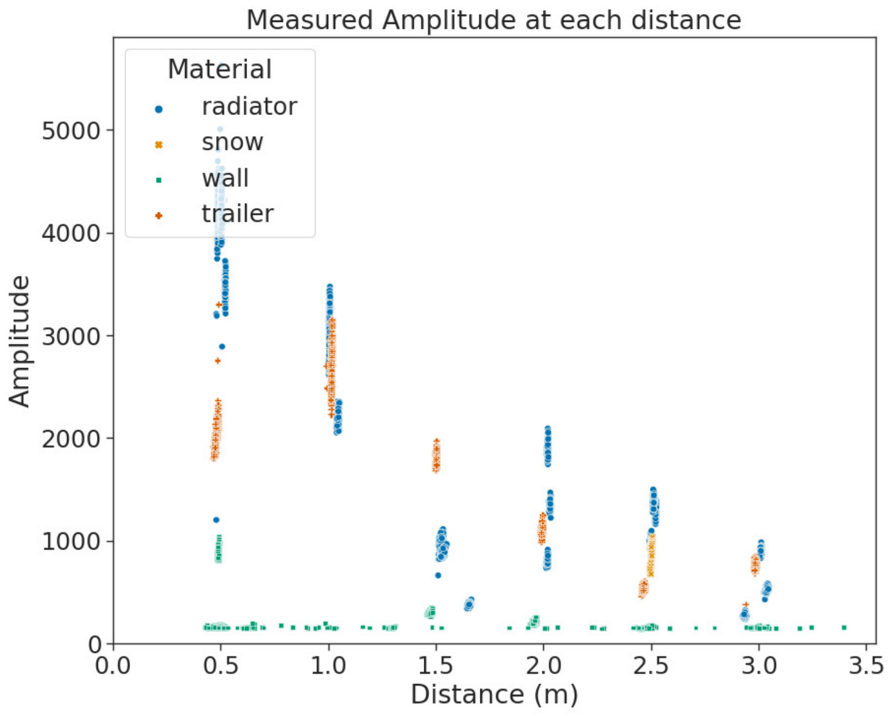
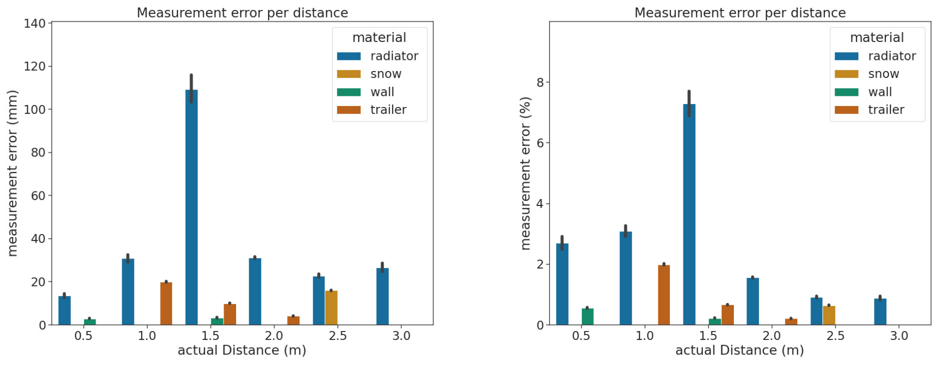


Figure 3.2: shows a scatterplot of all measured amplitudes from the Envelope service. Interestingly, the radiator amplitude at $1.5m$ has a lower amplitude compared to the following distances, which is unexplained. As we did not have a fixed measurement threshold, objects with too weak a reflection at a given distance will produce random distance measurements due to background noise.

### 3.3.2 *Vehicle detection evaluation*

The next step was to evaluate the reliability of the sensor during target detection. Some of the singular measurement data obtained can be referenced in Figure 3.4. For readability reasons, only an extract can be shown, but no additional information can be derived from the rest of the data, as the general pattern is the same. In this figure you can see the dashed lines where a car passed the sensor and was noted by a human. The blue dots represent distance measurements taken by the radar sensor. While the detection of cars was very reliable, with very no false positives and almost no false negatives during the test, one very important flaw was discovered. A car passing next to the sensor will not produce a single measurement, but will result in multiple measurements. The implementation on the sensor should therefore



(a) (b)

Figure 3.3: shows the measurement error, absolute (a) and relative (b), per set up experiment with the 95% confidence interval. The measurement error is most likely caused by the experimental setup and not by the measurements themselves, however even the recorded error would still be within acceptable limits.

be improved in the next iteration to produce a single measurement for each passing car. This could be achieved by further increasing the frame rate and adding some sort of timeout before a detection is considered complete. However this timeout should be selected carefully to avoid closely passing cars into one. Overall, the results seem very promising and the implementation is already very reliable, considering that no false positives occurred during testing in a stationary setup.

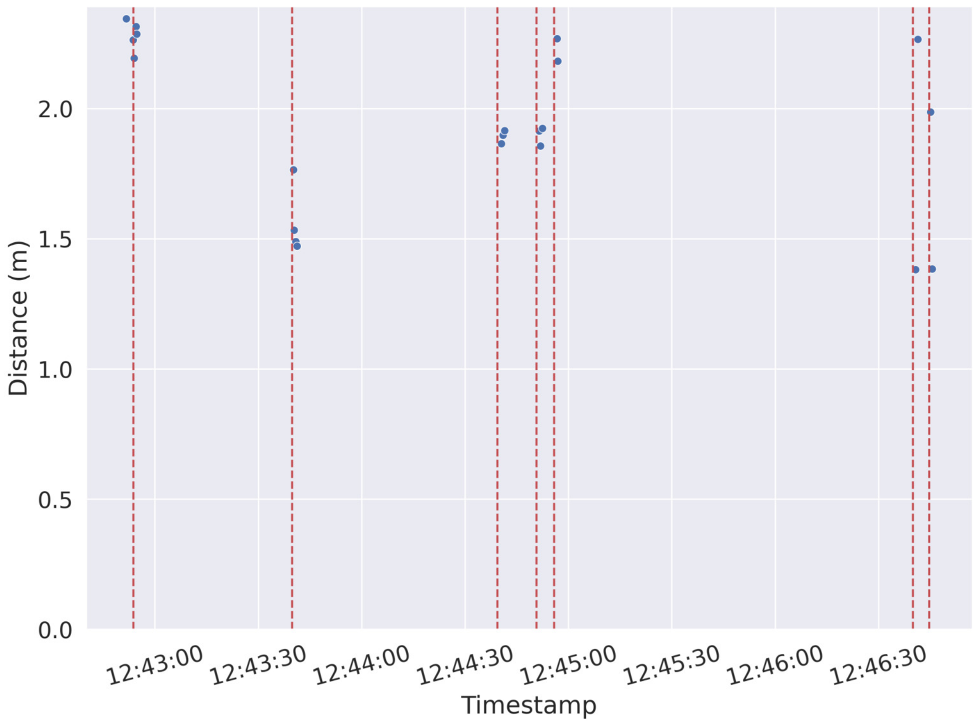


Figure 3.4: This graph shows a short section of the timeline of cars passing the sensor and the measurements taken using the envelope service. Only a snippet is shown to keep the graph readable. The rest of the data gives no additional information.

## 3.4 THIRD IMPLEMENTATION

For the third iteration, we started with a different mode of operation. Instead of the envelope service, this implementation was based on the power bin service. The main difference between the envelope service and the power bin service is that the power bin service samples the reflection every $0.5mm$, but groups them so that the resulting data is composed of bins with a width set by the user. This allows a higher frame rate because less data has to be moved between the radar and the microcontroller. The disadvantage is that the resolution is lower and the data is not as accurate. Using a bin width of $2cm$, and thus reducing the sample points to 135, the frame rate was increased to up to $40HZ$, but to ensure a stable frame rate we locked it at $30Hz$ to have spare time for additional tasks such as sending the data when necessary, and to reduce power consumption.

The CA-CFAR implementation of the second iteration was reused and further improved by adding in-frame signal smoothing. This means that instead of testing each cell individually, the algorithm uses a pre-defined number of neighbouring cells to determine the actual expected value of the current cell. Currently, all adjacent cells are used equally, but a further iteration could instead weight the cells based on their distance from the CUT. The reason for this change is that we have found that random fluctuations in the amplitude, caused by interfering background noise, can sometimes cancel out the reflection. This random noise artefact can mess up individual cells, but is very unlikely to affect several adjacent cells at the same time. This should additionally reduce the measurement jitter seen in the second iteration as it moves the measured peak closer to the centre of the overall peak. It also further reduces the likelihood of false positives by averaging out random cells above the threshold.

Based on the amplitude observations from the previous iteration, we modified the CA-CFAR algorithm to take into account the distance of the target when comparing to the threshold. Furthermore, the new algorithm measured the distance not only to the strongest reflection, but also to the first reflection above the threshold. This was done to avoid detection of close non-metallic targets. The required threshold is now additionally influenced by the distance to the sensor. A closer object requires a multiple of the threshold, while a further object only requires to be slightly above the threshold to be considered a detection. This is due to the fact that wooden objects or humans produce a reflection below the noise when they are beyond a distance of $\approx 1.5m$ with the selected sensor configuration.

Another important change was the addition of a timeout of $\approx 75ms$ before a detection is considered over. This correlates to 3 consecutive measurements without detection in our current duty cycle. This was done to solve the problem of a single car being separated in multiple detections discovered in the second iteration. In addition to the minimum required amount of measurements this resulted in very promising results in regards to the vehicle detection. In order for a metal car to not be detected, it has to pass the sensor in less than $100ms$. In reference to Figure 3.1 this means that the car needs to pass the cyclist with a $\delta v > 50\frac{km}{h}$. This would be a possible, but very very unlikely scenario, simply due to the danger of such a high speed at such a high distance. If the car passes at $2m$ the required $\delta v$ would be $\approx 95km/h$. It is therefore save to assume that for all practical purposes the minimum amount of required measurements won't cause any false negatives, but basically reduces the amount of false positives to a minimum by avoiding the detection of small metal objects such as Lamp posts, since they produce usually only a single reflection above the threshold. This is mostly due to their RCS reflecting to the sensor only when at the center of the pulse.

### 3.4.1 *Distance evaluation*

The distance evaluation was carried out in much the same way as in the second iteration. The two main differences were that in this setup we made sure that the bike was standing upright, as if it were in motion, and not on the kickstand. The second difference was that instead of measuring the distance from a single point, we now set up the bike so that the front and back of the bike were the same distance from the target, resulting in a much more parallel and repeatable setup, reducing the error caused by it. The results are again largely as expected and are shown in Figure 3.6. As we can see from the changes in the required amplitude compared to the threshold, in Figure 3.5 the measurements at random distances when using a wall as a target above 1.5m no longer occur. Looking at the measurement errors in Figure 3.8 and Figure 3.7 we can see that the human error has been significantly reduced with the new setup. The new sampling with the new implementation shows an expected measurement resolution of $2cm$. As we do not really see this in the results, we can conclude that the setup is still not precise enough to measure the absolute error of the sensor. However, we can see that the error is still well below the initially defined acceptable level of $10cm$. Table 6.2 shows a maximum spread of $4cm$ at 2.5m. This seems reasonable, especially considering the expected resolution, which would mean that if the target is at the $n_{th}$ measurement point, then by pure chance noise has caused the adjacent measurements to be the strongest reflection. Considering the fact that we have multiplied the measurement rate, this error is still acceptable, especially considering that this is the worst case of the measurements obtained.

### 3.4.2 *Vehicle detection evaluation*

For this experiment, no changes to the setup were made and it was repeated exactly as described above.

The data plot shown in Figure 3.10 shows that the changes to the implementation had the desired effect. The reference moments are shown in Figure 6.8 and Figure 6.9. A single car produces only one measurement as it passes the sensor, while the duration of $75ms$ is short enough to separate the

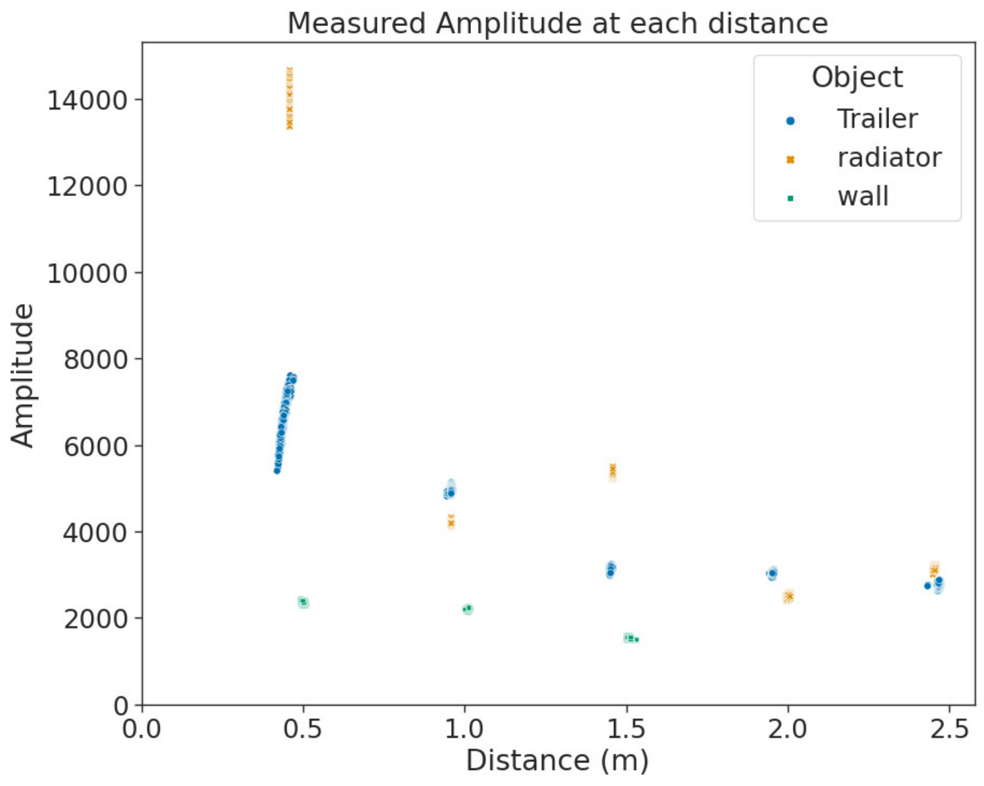


Figure 3.5: shows a scatterplot of all the amplitudes measured by the Power Bin service.

cars properly. Furthermore there was no false positive measurement, and no car was missed by the sensor. The selected section also shows the remaining problem. The only car without distance measurement, was in fact not missed by the radar, but the measurement got lost while transmitting to the phone. This needs to be addressed in the future. Secondly, cars in the opposite direction are still being detected, but this is to be expected and is less an implementation problem than a definition problem. To address this, expert knowledge from people experienced in road design is needed. These experts should advise on how to deal with this data, as it is technically a false positive, but if a car is heading towards a cyclist and passes the cyclist at close range, this would still be a dangerous situation and could therefore be valuable information. One example illustrating this can be referenced in Figure 6.10.

As we have been running in favourable conditions, we need to consider what would happen if the conditions were not so favourable. For example, assuming rain as an evenly distributed noise, we can conclude that this would simply result in a higher calculated threshold. Therefore we can safely assume that conditions such as rain, snow, high humidity or other weather conditions would have little or no effect on the results. The only way the weather would affect the readings would be through the sensitivity of the sensor, which would not result in false readings, only missed cars. This

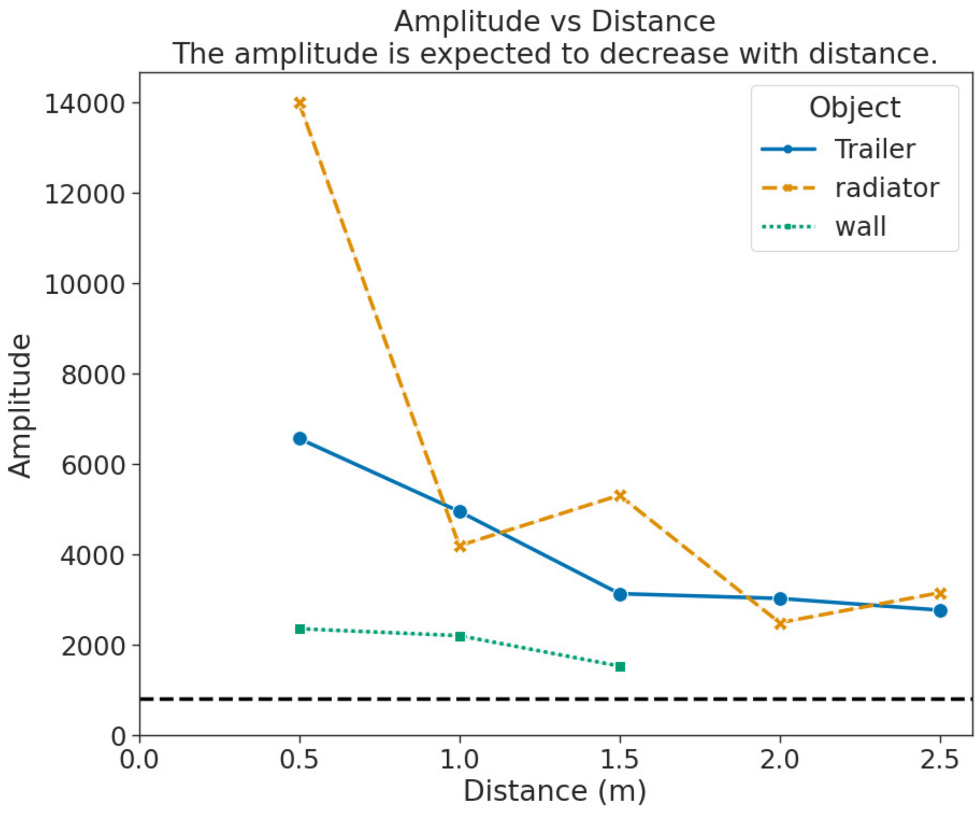
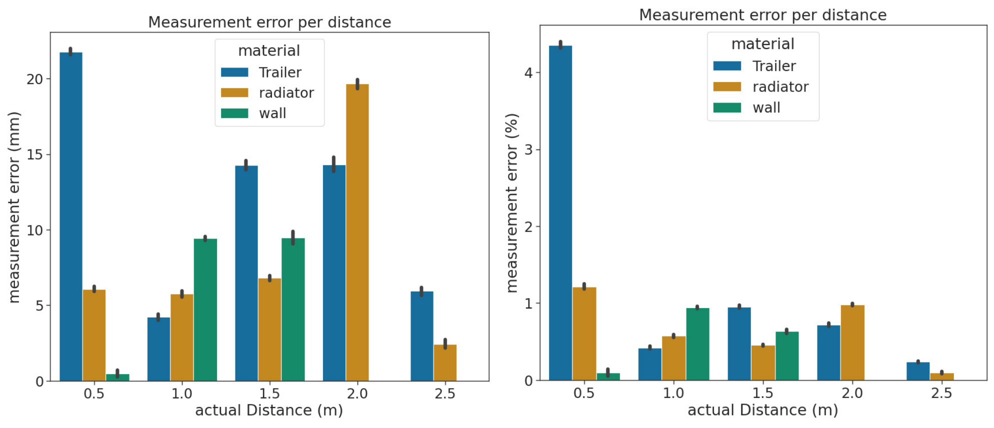


Figure 3.6: shows the average amplitude as a function of distance. The dashed line shows the background noise.

should be fine, as it will only minimally affect the analysis and the data will still be usable.



(a) (b)

Figure 3.7: Measurement error in (a) absolute and (b) relative values for the power bin implementation. The resolution is now 2cm, but the error is still in the range of 6cm.

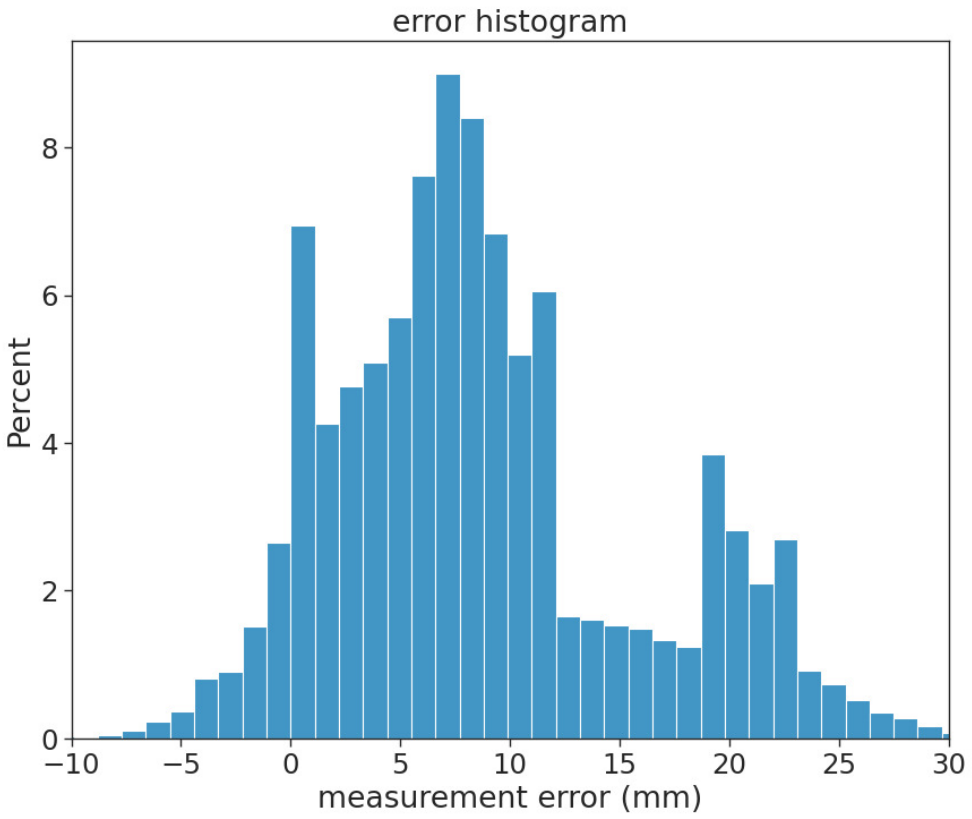


Figure 3.8: shows the histogram of all measurement errors of the power bin implementation. Since the power bin implementation has a resolution of $2cm$, the error would be limited to these increments if the setup were perfect.

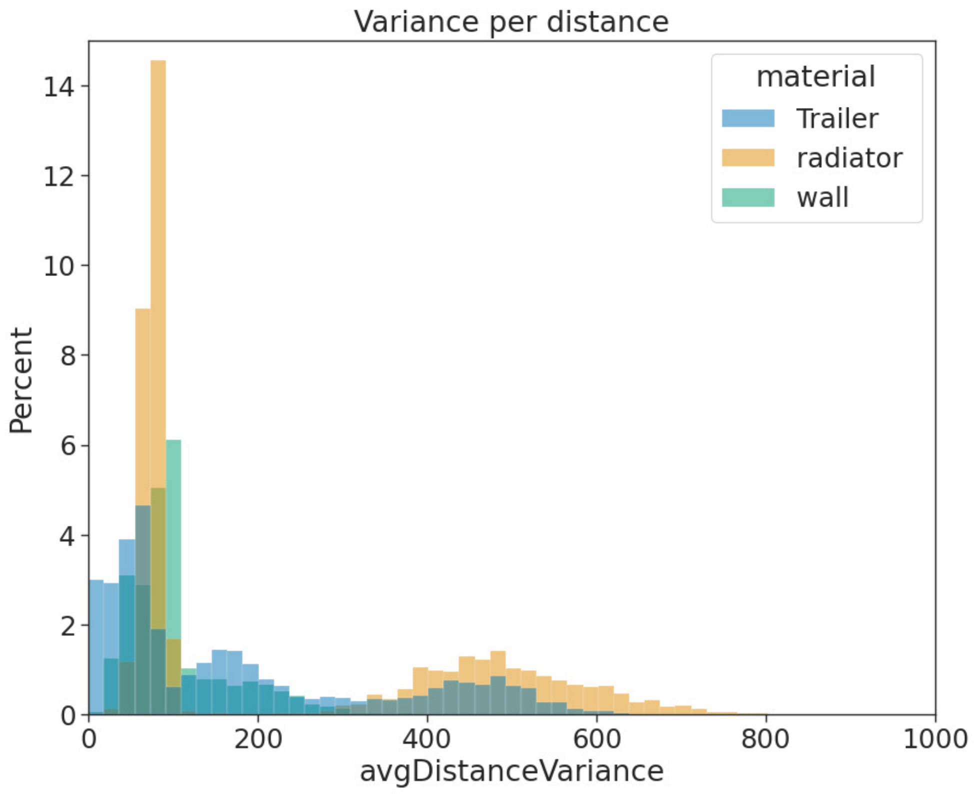


Figure 3.9: shows a histogram of the variance of the measurements. As expected most measurements show very little variance. The maximum recorded variance would still allow a relatively accurate measurement, but also show that individual measurements might not be accurate enough, to for example, draw conclusion about the minimum distance during an encounter.

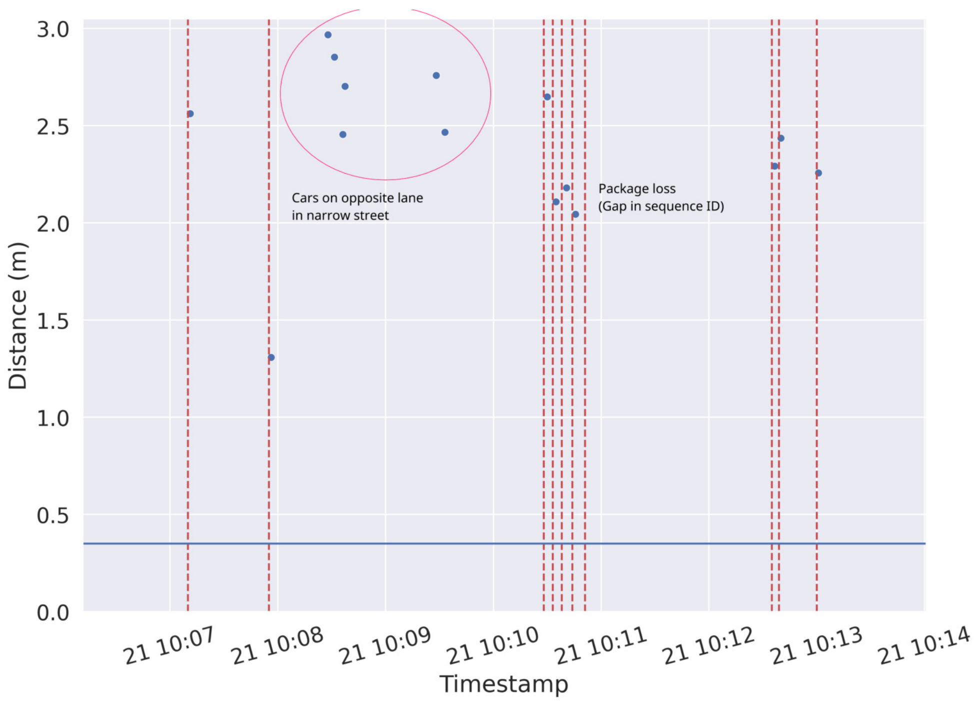


Figure 3.10: shows an interesting section of the data obtained. The dots are measured distances and the dashed lines are cars passing the cyclist, obtained from the reference video. This section is particularly interesting because it shows that the changes made in the third iteration allow a single car to produce only one measurement, but at the same time it highlights that cars in the opposite direction are detected when they come closer than $3m$. This graph also shows a case where a packet loss caused a missed measurement. This can be seen in the appendix.

# 4 ANALYSIS

In this chapter we will present different analysis techniques to evaluate the collected data. We start with a sample measurement to show how the data can be interpreted. This is followed by a simple tabular analysis and then more complex techniques such as aggregation and clustering methods. To obtain the data, the code produced during the third development iteration was used.

To do this, we chose a test route that starts from a common point, a supermarket, and makes a round trip along a main road. This road provides some infrastructure in the form of an „additional sign 1022-10" allowing cyclists to use the pavement. This infrastructure is not continuous, allowing a direct comparison between different conditions. The route also includes a section through a park where trees will be less than 2m to the left of the cyclist, providing an opportunity to test the detection algorithm when passing unwanted objects within the FOV and range of the sensor.

We also want to quickly review the general rules of the road for cyclists over the age of ten. In general, a cyclist must use the road. They must ride on the right-hand side of the road. In doing so, they should keep a sufficient distance from the kerb and parked cars [LG Berlin, Az. 24 O 466/95]. This is usually interpreted as the width of a door to a parked car. However, it is also required that if there is a cycle lane with the sign „traffic sign 237 cycle lane" you must use it. However, „traffic sign 1022-10 bicycle lane free" does not require cyclists to leave the road, and if they decide to do so, they must cycle at walking speed as defined by the StVO. With this in mind, we can conclude that cyclists are allowed to use the road along the entire length of the experimental route. In some sections, the use of the footpath is allowed if desired.

## 4.1 EXAMPLE MEASUREMENT

Before starting the general analysis of all the data collected, it is important to gain an understanding of the data collected. To do this, we will present a single measurement and explain how the different values can be used to interpret the situation and draw conclusions about the reliability of the measurement. Figure 4.1 shows the picture that belongs to the data in Table 4.1. Going through the rows in the table, we can see that each row with 0 or 1 measurements was a timeout transmission, and can be attributed to the lack of a sufficiently strong reflector. Furthermore, the second row can be correlated to the red car that passed at a considerable distance. As expected, the distance to the strongest and the first reflector are very close. Furthermore, the variance of both measurements is small enough to conclude that the measurement is reliable and that the car moved in a relatively straight line. The second detection can be seen in the fourth row. This detection belongs to the image shown in Figure 4.1. As we can see, both the distance to the strongest reflector and the distance to the first reflector are well below the required safety distance. Interestingly, the measurements aren't as close together as they are for the car. However, looking at the picture of the vehicle, a valid explanation would be that something further inside the vehicle is a stronger reflector than the outer frame of the truck. This is particularly interesting as it would mean that the strongest reflector would always produce measurements of a defensive nature, while the first reflector would produce a measurement that could be considered pessimistic. If we consider the sample variance of both measurement variants, we can estimate the confidence interval, assuming the measurement error is normally distributed, with $d \pm \sqrt{\sigma^2}$, giving a 95% interval of $[1.12m, 1.32m]$ for the strongest reflector measurements and $[0.83m, 1.28m]$ for the first detection measurements. With this in mind, we will use the measurement at the first reflector in the following analysis, unless otherwise stated. Furthermore, from this measurement it can be concluded without doubt that this was indeed a dangerous situation. Both measurements show a similar variance, because the variance is not caused by inaccurate measurements, but because the distance initially decreases, reaches a minimum, remains at this level until the next cyclist passes, and then increases again when the rear of the car passes the sensor. If the actual minimum distance to the car is desired, it would be a viable option to use the average of the first quartile as a minimum estimation. Using a quartile instead of a fixed number of measurements has the advantage that we do not need to know how many measurements make up the „close " part of the situation. We did not do this, simply because that for the intended use case it is more interesting to see how the overall situation unfolded. The reason for this can be illustrated by comparing two different situations. Situation A is where a car passes a cyclist in a straight line at an exemplary minimum distance of 1.2m. Situation B is where a car initially overtakes the cyclist at more than the required safety distance, without a clear view, and suddenly has to reduce the safety distance to 1.3m to avoid oncoming traffic. While situation A produced an overall lower measurement with a small confidence interval, situation B is arguably the more dangerous situation, with a naturally larger variance in its measurements. In addition, situation A may give a bad impression, while situation B may actually be scarier. This leads to another aspect that the measurements can't accurately capture. Since the aim is to make cycling more attractive, the perception of safety is also important. And a large and noisy vehicle passing at a certain distance may give a very different impression to a smaller vehicle passing at the same distance.

The first approach to analysing the collected data was simply to generate a table. Table 6.3 shows the first ten data points, ordered by the average distance of the location in ascending order. This allows you to identify the locations with the lowest average distance, and therefore the most dangerous locations. Additionally, filtering out locations with a high standard deviation

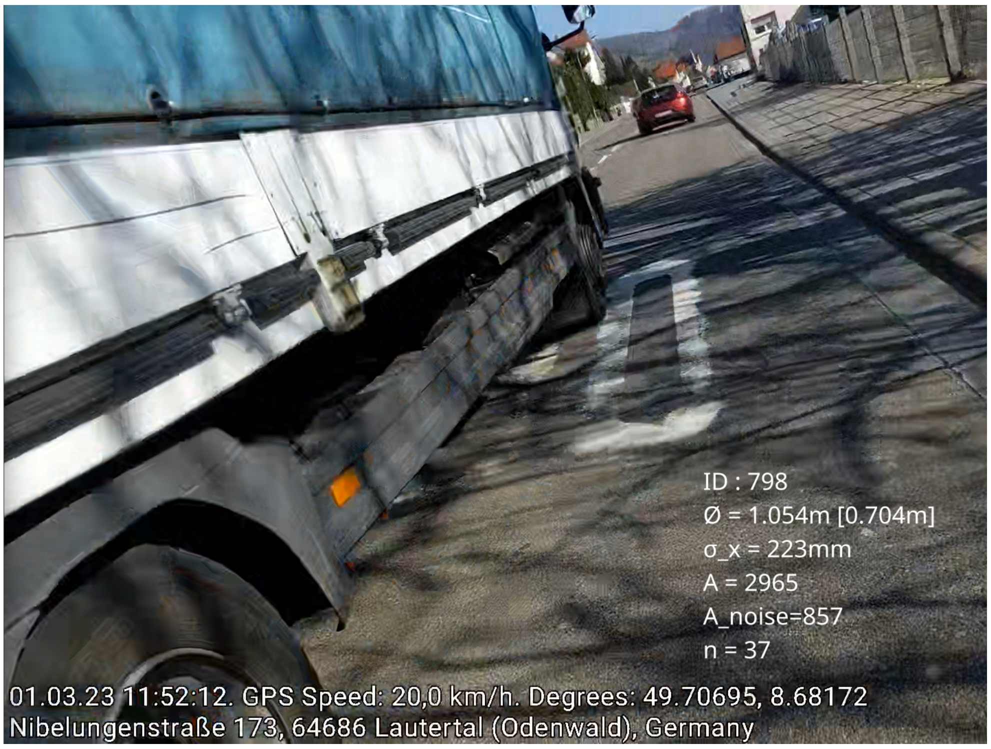

ID : 798
Ø = 1.054m [0.704m]
σ_x = 223mm
A = 2965
A_noise=857
n = 37
01.03.23 11:52:12. GPS Speed: 20,0 km/h. Degrees: 49.70695, 8.68172
Nibelungenstraße 173, 64686 Lautertal (Odenwald), Germany

Figure 4.1: shows the screenshot displaying a typical encounter. Notably is that the correlation to Table 4.1 is perfect. The red car was measured, then a timeout occurred, and then the truck was measured. Furthermore it is an interesting example because the red car produced a measurement where the first and strongest reflector basically is the same distance, while the truck produced a measurement where the first and strongest reflector are significantly different. This is the reason why we will use the first reflector in the following analysis. Since the measured distance of the first reflector correlated better with the intended use case.

Table 4.1: measurements from 2023-03-01 10:52:00 to 2023-03-01 10:52:30 between 0.0m and 3.1m

| timestamp (GMT) | $\overline{d}(m)$ | $\sigma_d^2$ | $\overline{h}(m)$ | $\sigma_h^2$ | v | N |
|---|---|---|---|---|---|---|
| 2023-03-01 10:52:06.604 | 3.00 | 0 | 3.000 | 0 | 6.08 | 0 |
| 2023-03-01 10:52:08.891 | 2.24 | 3729 | 2.230 | 4009 | 5.98 | 12 |
| 2023-03-01 10:52:11.269 | 3.00 | 0 | 3.000 | 0 | 6.17 | 0 |
| 2023-03-01 10:52:13.028 | 1.22 | 10047 | 1.054 | 50075 | 5.57 | 37 |
| 2023-03-01 10:52:16.968 | 3.00 | 0 | 3.000 | 0 | 5.83 | 1 |
| 2023-03-01 10:52:19.802 | 3.00 | 0 | 3.000 | 0 | 5.83 | 1 |

helps to filter out locations where the data is not yet conclusive. We did not filter at this point as we wanted to see the full picture.

However, a possible filter would be your own speed of movement. If the cyclist is stationary, the data collected may not be representative of the actual situation, for example if the bike is parked on the pavement. Another filter could be the number of measurements. If the number of measurements is too low, the data is not conclusive enough to draw conclusions. The same goes for the standard deviation. Using the standard deviation of a detection, we can argue about what range the car must have been in with some confidence. A lower bound for the width of this interval has already been presented in Figure 3.1. Another indicator of the reliability of the data is the difference between the strongest and the first reflector. This value will diverge because the first reflector will usually be the shortest distance to the car, while the strongest reflector should be the distance to the largest area with the best reflective properties such as angle, etc. If the difference is too large, this may indicate that the measurement is not reliable.

## 4.2 MAP VIEW

The next step was to visualise the data on a map to make the data more visually appealing. The maps were obtained from openStreetMaps. Due to the terms of use, it is not allowed to automatically query map renderings. For this reason, we currently output the download URL when generating the map. This allows the user to manually download the map background without violating the openStreetMaps terms of use. Figure 4.2 shows the map with the locations marked, including the timeout broadcasts where no car was detected. While this map is difficult to read in terms of interesting locations, it does serve the purpose of visualising coverage. If we only transmitted when a car was detected, the system would have no way of distinguishing between areas where no data was available because no one was cycling there, and areas where no data was available because no dangerous situation occurred. To identify dangerous locations, the data must first be filtered. One logical step is to use only measurements where the measured distance was less than the required safety distance. The required safety distance actually depends on the bicycle used, as the required 1.5m in cities starts at the handlebars. So the handlebars must be added to the required distance. Individual measurements can be linked to a user by their phone's UUID. This is why we had to include non-anonymous data in the data collected. In addition, we are usually only interested in measurements where the cyclist is moving. Therefore we added a filter to remove data where the measured speed is below $1\frac{m}{s}$. This speed is chosen arbitrarily and is simply based on the idea of removing as little data as possible from the following analysis. For the same reason, we have not added filters based on the variance of the measured distance, the accuracy of the location provided, or other items that are theoretically a viable filter to maintain a high quality standard. Filtering based on these values becomes most interesting when analysing crowdsourced data to draw conclusions. At this point, we also want to keep questionable data points to be able to identify possible problems with the sensor.

With this subset of the collected data, we generated a map by simply grouping based on rounding the latitude and longitude to achieve grouping within $10m$ bins. This is *only possible near the experimental site* due to the localised data. The result can be seen in Figure 4.3. While this has worked reasonably well, it has the disadvantage of using a simple proximity approach. This means that a road is not considered as a whole if it is divided into several sections. While this isn't really a problem, a long dangerous road would not be as prominent as a short dangerous section. At the same time, a long section would be even more interesting to road planners, as infrastructure improvements to such sections would have a greater impact. To achieve this we used the DBSCAN algorithm presented in Section 2.2.4.2. We chose this algorithm because it's parameters are logically translatable to the real world. The neighbourhood parameter $\epsilon$ can be expressed in metres and describes the maximum distance for two detections to be considered „next to each other ". We decided to evaluate this parameter empirically by generating maps with an epsilon for 10m, 100m and 1000m. This time, instead of simply rounding the location, we transformed the geological data into radians. The second parameter is the number of measurements required to be considered a hazardous location. Since we collected the data over a period of 60 days and made a total of $\approx 70$ trips, we chose a minimum number of 10 detections. This effectively means that 15% of the time someone cycled there, they were overtaken below the safety distance". Applying this clustering algorithm to the data with a $\epsilon = 100m$ resulted in the map shown in Figure 4.4. The identified location belonging to cluster 1 and 2, can be seen at https://sebastian-schuch.de/cluster/1 and https://sebastian-schuch.de/cluster/2. The cluster with ID 0 did only emerge shortly before finishing this thesis, therefore we were not able to acquire a snapshot in time. The required neighbourhood size can be further reduced as the data coverage increases, as the clusters can grow organically. This will also avoid adding noise points to the identified clusters. Additionally, during the slow data collection, we observed that the cluster with ID 0 and 1 slowly moved towards each other, suggesting that these two clusters are likely to merge with increasing data coverage, indicating a long strip without sufficient cycling infrastructure. The current state of the collected data and identified clusters can be seen in Table 6.4. Based on this data we could now rank the identified locations, by their absolute or relative amount of hazardous situations. It is also possible to rank these clusters, for example by average distance. The first would prioritise locations where a lot of users encounter uncomfortable situations. The advantage here is that it would maximise the overall benefit by ensuring that priority is given to places where a lot of cyclists benefit from the changes. However, it runs the risk of unfairly favouring areas where there are simply a high number of cyclists. The second approach would take this into account, but is much

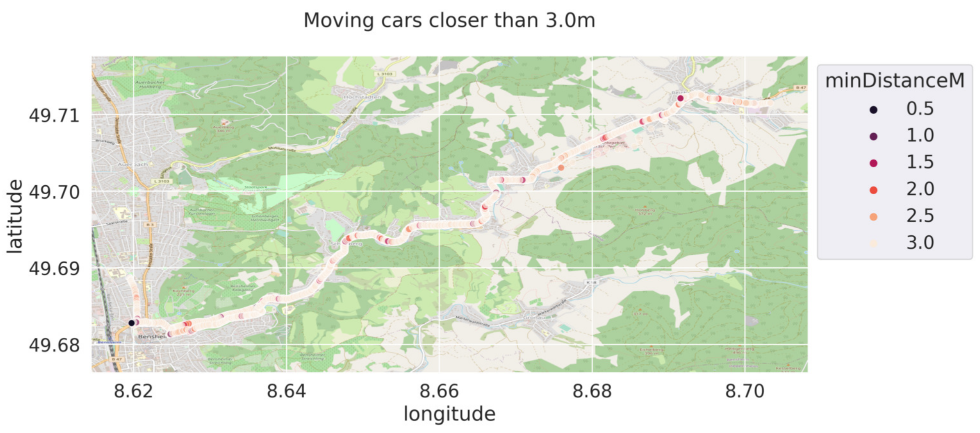


Figure 4.2: shows the map of locations with the measured distance marked. This includes the data points when no cars were present.

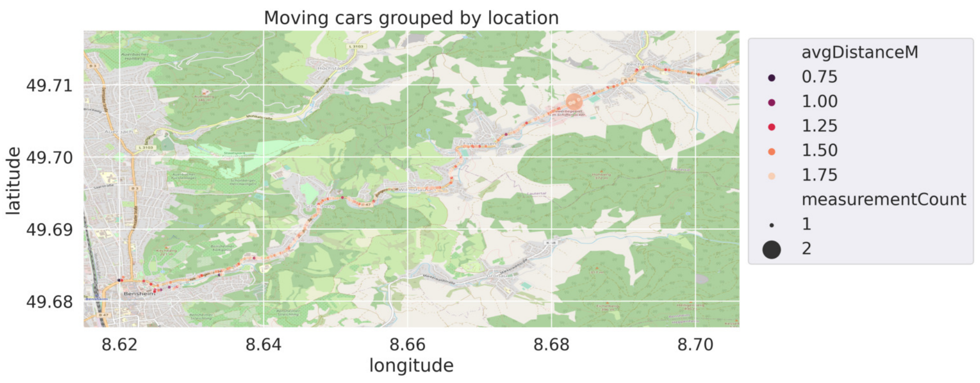


Figure 4.3: shows the map of locations where the distance was less than 1.85m or 1.5m safety distance, points are grouped by proximity and scaled by frequency. This map is too cluttered to be useful, since it does not highlight locations with a notable accumulation of dangerous encounters, and only allows the visual identification of areas with a lack of dangerous encounters.

more susceptible to false conclusions based on chance alone. Areas where few people cycle, but where there is a randomly occurring dangerous encounter every time, would be ranked high, even though the overall benefit of improving the infrastructure elsewhere would be greater. However, these considerations should be made by experts, as the interpretation of the data should be done by the people responsible for the infrastructure. The data is simply a tool to help them make better decisions, and it seems capable of doing so.

Although we did not collect enough data due to time constraints, the clusters identified do indeed support the first hypothesis. It is true that areas without any infrastructure are not as prominent as areas with unusable "cycling free„ , for example because of parked cars. While this may be a coincidence, it is interesting to note and should be kept in mind for further analysis. Additionally, the location referenced in Figure 4.5 shows a particularly bad example of infrastructure. The cycle lane ends abruptly after

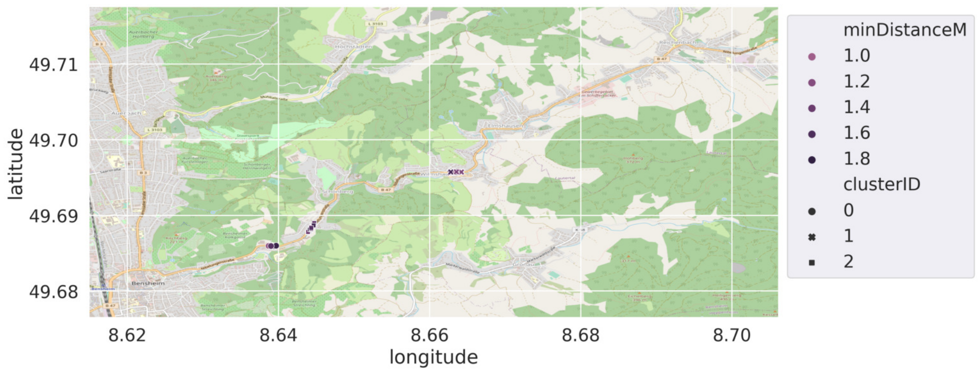


Figure 4.4: shows the map with the cluster. As $\epsilon$ we have empirically determined $50m$ and 11 as the minimum number of points that correlate with $\approx$ 33% of the times cycled there, due to the overall 33 trips performed. Due to the very low range epsilon, the clusters are currently limited in size. However it is expected that the clusters will grow organically as the data coverage increases. It might be advantageous to actually further decrease epsilon, since 50m, could result in two parallel streets being assigned to the same cluster, despite never actually meeting. For this to be successful, a much higher data coverage would be required.

a bend, only to begin again a few metres further on. This forces the cyclist out into the road for no apparent reason, causing drivers to overtake closer than they would if the cyclist was already on the road. As the developed radar collects the same type of data as the openBikeSensor, the map `https://obs.adfc-darmstadt.de/` can be consulted for more insight into the possible conclusions that can be drawn from sufficiently large data sets.

## 4.3 DUTY CYCLE & PACKAGE LOSS

Figure 4.6 shows the measurement duration. The expected duration would be at around $33ms$, and sometimes above when individual measurements have to be discarded due to a bad SNR ratio. However the fact that the histogram shows other accumulations at multiple of the expected measurement duration, needed to be investigated. Investigating the received data and logging the IOT output via UART, revealed that the problem was not part of the embedded code. As can be concluded by gaps in the sequenceID in the received data, the problem was caused by the Bluetooth Low Energy (BLE) scan. While package loss is to be expected in a connection less protocol, the amount of packages lost was too high.

We improved the amount of packages lost by changing the BLE scanner settings to `SCAN_MODE_LOW_LATENCY`, `CALLBACK_TYPE_ALL_MATCHES` and `MATCH_MODE_AGGRESSIVE`. However this only worked for the first 30min of a cycling tour, after which the package loss significantly worsened. This was caused by the BLE scan being downgraded automatically after $30min$ to the „opportunistic "mode. This was revealed when Investigating the implemen-

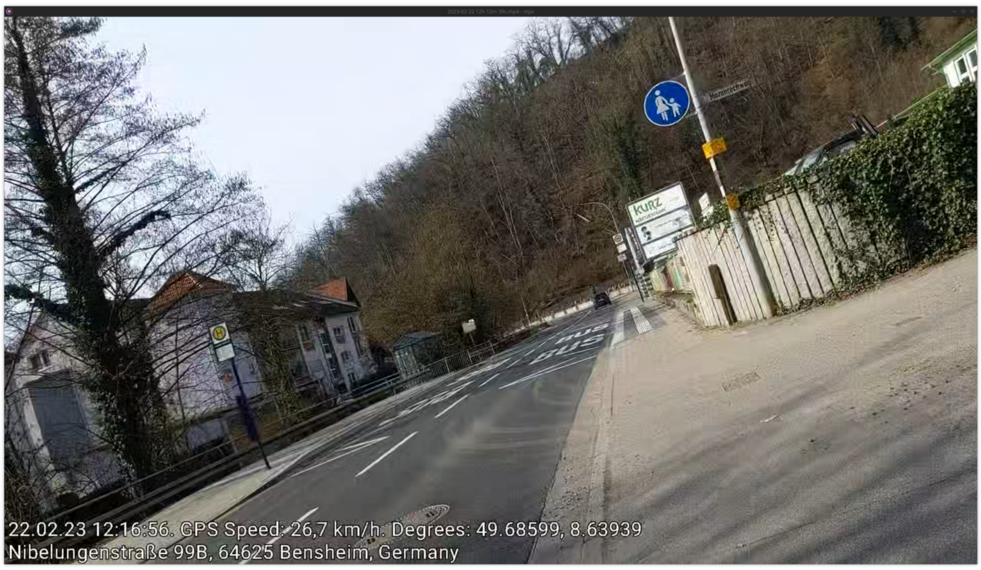

22.02.23 12:16:56, GPS Speed: 26,7 km/h, Degrees: 49.68599, 8.63939
Nibelungenstraße 99B, 64625 Bensheim, Germany
BUS

Figure 4.5: shows one of the locations identified by DBSCAN with $\epsilon = 100m$ and 7 as the minimum number of points. This location shows the end of one cycle lane and the immediate start of another.

tation of the AOSP scanner code. At the time of writing this thesis this was not specified in the BluetoothLeScanner documentation[1].

```
// Maximum msec before scan gets downgraded to opportunistic mode
 private static final int SCAN_TIMEOUT_MS = 30 * 60 * 1000;
```

.

We decided to circumvent this limitation by automatically stopping and restarting the scan as soon as no package was received for more than $10s$. Since the sensor should at least broadcast every $2s$ this means that the scan will be restarted as soon as at least 5 consecutive packages are lost. At the same time scan will then not be restarted until at least one package is received. This means that if no package is received for more than $30min$ after a restart, android will again downgrade the scanner to opportunistic mode to save power.

This solution more or less reduced the amount of package loss to a minimum while preserving the intended power save feature. If the user then starts a new trip, eventually even in opportunistic mode a package will be received, and the scanner mode will then be upgraded automatically again.

This solution provides a tradeoff between energy consumption and package loss, considering the fact that it allows to be used without any form of user intervention.

1 https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner

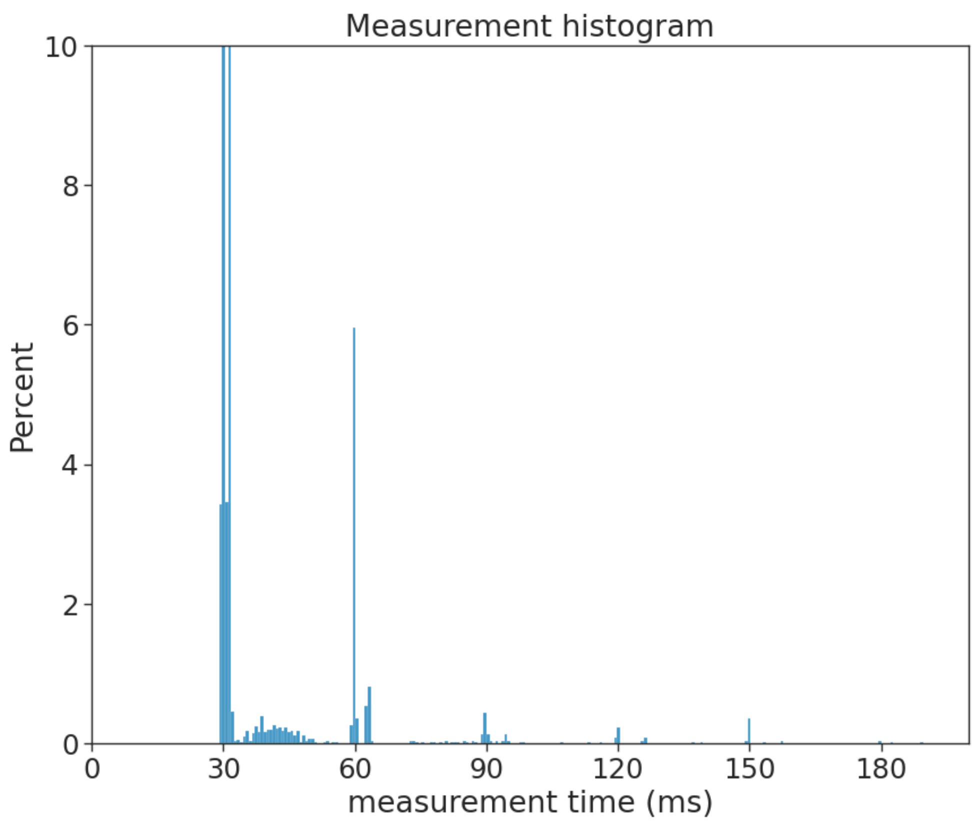


Figure 4.6: Shows the average measurement time histogram. As we configured the sensor to measure every 30*ms*, we were not expecting the other data points. Examining the data showed that the multiples of 30*ms* are caused by lost measurements.

# 5 CONCLUSION

Over the course of 6 months, we've researched, designed and evaluated a prototype using the DSR method. This prototype performs the same measurements as the OBS, but does not require any user interaction. We've implemented 3 different versions, evaluating each implementation at the end of a DSR cycle, using the knowledge gained as a basis for the next iteration, iteratively improving the sensor until we've achieved an accuracy and precision of a few centimetres. This required a fine balance of tuning the radar configuration to achieve the optimal tradeoff between duty cycle, resolution and sensitivity. Analog to the CAP theorem improving one of these aspects always means sacrificing performance in the others. Furthermore, the final prototype not only measures the distance of passing cars, but also includes the quality of the given measurement. This allows for very sophisticated analysis and ensures a high level of data integrity. With the data collected during the third experiment, we were able to conclude that we were able to apply different analysis methods to extract knowledge about the cycling infrastructure and prove the reliability of the data collected. Unfortunately, some valuable data was lost due to various reasons, such as the loss of BLE packets, as well as hardware issues. Once these remaining issues are resolved, we would argue that the prototype is ready for a larger scale citizen science project.

By using DBSCAN we were able to identify the areas in our test road where the safety distance was remarkably often not maintained. This is a very promising result and exceeds the currently deployed analysis methods of the OBS infrastructure of the ADFC Darmstadt. Therefore even if the radar does not make it into the project, the new analysis methods could be used to improve the current infrastructure running with the manually operated OBS sensors.

However the device has some inherent problems. During the course of writing the thesis we have encountered numerous situation that pose a significant threat, but are not considered by the initial premise. Such as suddenly pulling out cars from a side street, cargo sticking out to the side and many more. The problem is that adding enough radars to achieve a 360 degree FOV would cause a significant cost overhead simply to cover these edge cases.

## 5.1 INTERVIEW WITH THE ADFC DARMSTADT

The Allgemeiner Deutscher Fahrrad-Club (ADFC) Darmstadt was kind enough to give us an interview about their experiences with the OBS and their opinion of the new prototype. The interview was conducted with *Mr Görgen* on the 14th of March, 2023. During this interview we presented the prototype and the collected data and asked for their opinion. In particular, we discussed the need for the handlebar button when using the radar to distinguish between cars and other objects. Mr Görgen explained that the button will most likely remain a necessary part, as the overall goal of the project is to transparently communicate the safety of the infrastructure to city officials. Only allowing data that has been manually marked as valid is a reliable way to convince non-technical users of the legitimacy of the data being collected. The button also plays a key role in ensuring that the data collected is only collected on the street. As part of the project requirements, no data can be collected on a path separated by a kerb, as this is not part of the road infrastructure according to the legal definition, and therefore there is no required safety distance. The same logic applies to vehicles approaching cyclists on narrow roads. As *Mr Görgen* explained, situations such as one-way streets where cyclists are allowed to ride in both directions need to be assessed in the full context, where the distance alone does not provide enough information. This led to a brief discussion about possible improvements to the analysis methods. He mentioned that the website provides a very useful filtering mechanism, but at the time of the interview, the website did not provide a way to analyse the data for hotspots where the safety distance is not regularly maintained, indicating that the presented application of the DBSCAN is a new approach. Finally, we discussed the cost of an OBS sensor. Although there is no definitive answer to this question, as the price depends on many external factors, he estimated that the price would be between €50 and €100. This could indicate that the price of the XM122 radar is below the cost of the OBS, especially considering that the XM122 only requires the addition of a power source, reducing the amount of assembly required.

## 5.2 FUTURE WORK

During the course of this thesis we've identified some aspects that could be improved in future work. This section gives an overview of these aspects.

### 5.2.1 *Refining the data*

As we have presented several situations in this thesis that produce measurements that are not technically cars overtaking the cyclist, the measured distance is still correct and belongs to an interesting situation. Therefore, it is crucial to consult an expert in the field of urban planning to decide how to handle this data. We suggest that the data be treated as follows:

- Cars in the opposite lane - This data should just be kept as the measurements are usually long distance, but if a car is coming towards the cyclists and still falls below the safety distance, this should still be highlighted and investigated to see if changes to the infrastructure can resolve this.

- Stationary objects to the right - This data should also be collected by adding a radar to the right side, as the free space to the right is a relevant factor in interpreting the distance to the left.
- Stationary objects to the left - This data should be recorded, but it is crucial to identify whether an object is stationary by checking the Doppler shift. This can be done by occasionally taking a measurement optimised to measure the radial velocity of the target. This means sacrificing some distance measurements, but it is crucial to identify stationary cars as they are less relevant for analysis.

### 5.2.2 *Streaming*

Instead of using BLE broadcasts, it is important to switch to a lossless protocol. The packet loss is too great to be acceptable and causes too much data loss. This is a problem that other researchers are already investigating, as it is a known problem for low power wearables [29]. The reason we did not move away from the current implementation, despite the fact that it caused us some data loss, was the ease of use. Instead of having to pair your phone and make sure a connection was established, the broadcast approach we chose allowed you to just start cycling, as the sensor is automatically turned on with the bike, and listening to the BLE broadcasts was always possible due to the almost non-existent battery consumption. This convenience must be maintained in order to maximise the adoption of the sensor. Therefore, we need to find a way to maintain the comfort of the current implementation while still being able to stream the data losslessly.

### 5.2.3 *Hardware*

At the time of writing, the available close range radars changed substantially with the announcement of the A121 chip [4]. It is a successor to the A111 and also has a 60Ghz radar. In particular, the SDK has been simplified by combining all the services into one, which greatly improves performance for our use case by solving the problems described in Section 3.1. This is done by providing a much more fine-grained configuration, allowing step length, number of sweeps and increased buffer sizes. Therefore, the experiments performed in this thesis should be repeated with an A121 radar chip for a direct comparison. At the same time, the experimental setups need to be repeated with the XM122 chip in order to obtain a more accurate measurement of the radar chip's accuracy by eliminating the deficiencies of the setup presented in Chapter 4 Furthermore, when starting a new development iteration with a new prototype, consider replacing the USB-C power supply with another connector that has a locking mechanism to avoid the problem of lost data due to a loose connection.

### 5.2.4 *Measuring the distance to the right*

One improvement to the experimental setup that was discovered but could not be implemented in time is the ability to measure the distance to the right of the cyclist. It is a legal requirement to keep a safety distance from parked cars to avoid opening doors. It would only be necessary to mount the sensor on the other side as well, since the detection algorithm already works reasonably well in distinguishing between cars and walls, for example. This would also make it possible to check whether cars are overtaking closer, if the cyclist keeps the required distance from parked cars. See Figure 5.1 for an example of a situation where the safety distance was not maintained due to parked cars. The use of a second sensor to determine the available distance from parked cars on the right would further improve the quality and reliability of the data, as it would be possible to assess whether the cyclist is able to maintain the required distance from parked cars.

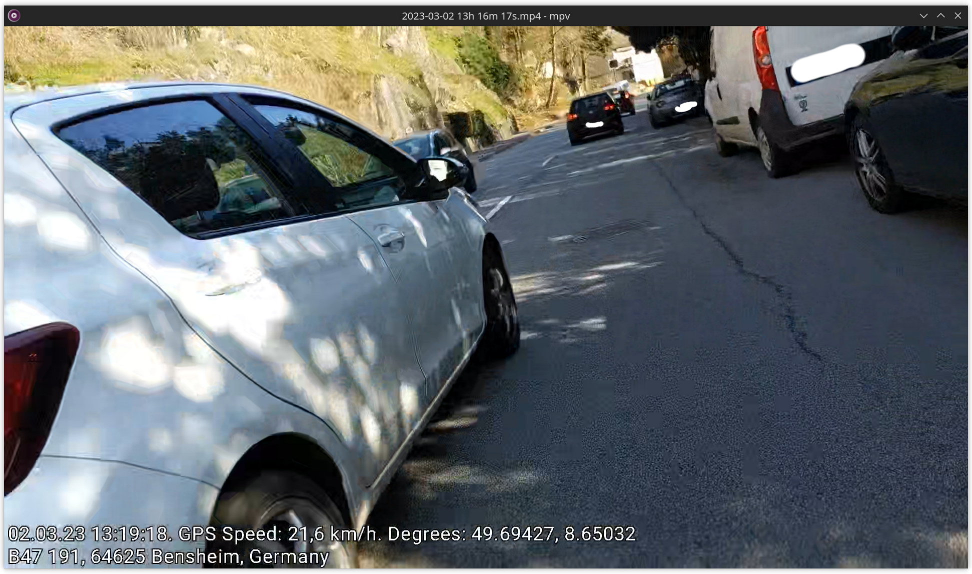

2023-03-02 13h 16m 17s.mp4 - mpv
02.03.23 13:19:18. GPS Speed: 21,6 km/h. Degrees: 49.69427, 8.65032
B47 191, 64625 Bensheim, Germany

Figure 5.1: shows a typical situation where the safety distance has not been maintained due to parked cars. Using a second sensor to calculate the available distance from parked cars on the right would further improve data quality and reliability.

### 5.2.5 *Data Integrity and Privacy*

At this stage, no measurements have been taken to ensure data integrity. This is because the only filtering implemented in the application is by MAC address. Therefore, a malicious actor could sniff the packets sent by the radar, copy the MAC address to spoof it's own, and inject synthetic data into the system. One way to prevent this would be to implement a signature on the packets that could be verified by the application. A user friendly way to implement this would be to package the radar with a QR code containing the MAC and public key associated with the device. If this cannot be implemented due to BLE limitations, a pairing process as specified by BL [8] is required. The data sent is highly sensitive as the GPS data allows the user to be tracked. Until enough users submit data, a de-anonymisation attack is possible. Regardless of the number of users, the app should be extended to allow the addition of zones where no measurements are taken, to avoid the disclosure of important locations such as home or work. Furthermore, the GDPR requires that the user has the possibility to delete their data, which is currently not possible as the data is not personalised.

### 5.2.6 *User Engagement*

Once enough data has been collected from users, work can begin on a routing algorithm. The algorithm should be able to take the data into account and calculate a route that minimises the risk of close encounters. The algorithm should be able to optimise for different trade-offs, minimising additional distance while minimising time spent in dangerous areas.

In addition to the routing algorithm, the application should be extended to increase user incentives, possibly using a gamification approach. The current prototype does not provide any incentives for the user to use the app, except for purely altruistic reasons. The app could be extended to include an interesting dashboard showing the user's statistics, such as the number of close encounters, the distance travelled, a heat map of their own data. This would also allow the user to see the impact of their actions by showing them their own data as a heatmap.

### 5.2.7 *Continuous Analysis*

At the time of writing, the serverless backend was only deployed locally and analysis was triggered manually. This was sufficient due to the limited amount of data collected during the evaluation phase. However, when the system is deployed in a production environment, the backend should be deployed in such a way that each new batch of data submitted is automatically used to update the currently found clusters. In theory, this should be efficient due to the way the DBSCAN algorithm works. Before inserting the data into the database, it is easy to check if it is in the neighbourhood of an existing cluster, or if it will lead to the formation of a new cluster. We expect that the biggest edge case to be aware of is that new data may cause previously separated clusters to merge, resulting in the possible need to update a large amount of data. In addition, the total number of clusters will increase massively as the reach increases, which would massively increase the runtime of the analysis if the analysis is restarted from scratch on each new batch of data. Furthermore, it may be possible to use simple location-based grouping to first separate the data into different regions, such as cities, and then do the clustering on a per-region basis. Since we do not expect to find clusters that span multiple regions, this should be sufficient to reduce the runtime of the analysis by allowing independent and parallel processing of the data.

## 5.3 LESSONS LEARNED

During the development iterations we encountered a strange problem regarding the power supply soldered to the chip. Debugging the power supply with a ammeter showed that it was missing a $5.1k\Omega$ resistor specified in [12, p. 232] in Table 4-25. If this isn't done most USB-C chargers won't deliver any power at all, and most power banks will shut down, since the communication with the power sink fails. This resulted in the strange problem that the radar sensor could only be powered with USB-A chargers. With this experience we recommend to redesign the power supply and not buy cheap.

The second lesson learned is that performing the experiments and acquiring sufficient data took way longer than initially planned. This is mostly due to the intensive labor requirement of the experiments that can not be automated in a meaningful way. Collecting data for approximately one third of this thesis was just too much of a time sink to be worth it. As for the next iteration, we would recommend acquiring volunteers beforehand, and distributing the devices to them. In combination with an update to the app, allowing to update the device firmware over bluetooth, this would allow the acquisition of data to be much faster.

Part II

# APPENDIX

# BIBLIOGRAPHY

[1] [Online; accessed 12. Sep. 2022]. Dec. 2021. URL: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Verkehrsunfaelle/Publikationen/Downloads-Verkehrsunfaelle/unfaelle-zweirad-5462408207004.pdf?__blob=publicationFile.

[2] Infineon Technologies A. G. *BGT60LTR11AIP | XENSIV™ 60GHz first completely autonomous radar sensor for motion sensing - Infineon Technologies*. [Online; accessed 25. Nov. 2022]. Nov. 2022. URL: https://www.infineon.com/cms/en/product/sensor/radar-sensors/radar-sensors-for-iot/60ghz-radar/bgt60ltr11aip.

[3] Acconeer AB. "XM122 Software Development Guide, User Guide." In: a111-v2.12.0 (2022).

[4] Anna Aleryd. "EVK for A121 available now - get yours now to try out our new radar sensor - Acconeer." In: *Acconeer* (July 2022). URL: https://www.acconeer.com/news/evk-for-a121-available-now-get-yours-now-to-try-out-our-new-radar-sensor.

[5] Otmane Amimi, Anass Mansouri, Saad Dose Bennani, and Yassine Ruichek. "Stereo vision based advanced driver assistance system." In: *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*. 2017, pp. 1–5. DOI: 10.1109/WITS.2017.7934605.

[6] Norbert Buch, Sergio A. Velastin, and James Orwell. "A Review of Computer Vision Techniques for the Analysis of Urban Traffic." In: *IEEE Trans. Intell. Transp. Syst.* 12.3 (Mar. 2011), pp. 920–939. ISSN: 1558-0016. DOI: 10.1109/TITS.2011.2119372.

[7] A. Carullo and M. Parvis. "An ultrasonic sensor for distance measurement in automotive applications." In: *IEEE Sens. J.* 1.2 (Aug. 2001), p. 143. ISSN: 1558-1748. DOI: 10.1109/JSEN.2001.936931.

[8] Matthias Cäsar, Tobias Pawelke, Jan Steffan, and Gabriel Terhorst. "A survey on Bluetooth Low Energy security and privacy." In: *Comput. Networks* 205 (Mar. 2022), p. 108712. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2021.108712.

[9] *Die Siegerprojekte 2022 – Deutscher Fahrradpreis*. [Online; accessed 8. Nov. 2022]. Nov. 2022. URL: https://www.der-deutsche-fahrradpreis.de/siegerprojekte-2022.

[10] *FCC Seeks to Enable State-of-the-Art Radar Sensors in 60 GHz Band*. July 2021. URL: https://docs.fcc.gov/public/attachments/FCC-21-83A1_Rcd.pdf.

[11] *Frequency of people cycling in Europe 2013 | Statista*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://www.statista.com/statistics/820612/frequency-european-cycling.

[12] USB 3.0 Promoter Group. *Universal Serial Bus Type-C Cable and Connector Specification*. 2019. URL: https://www.usb.org/sites/default/files/USB%20Type-C%20Spec%20R2.0%20-%20August%202019.pdf.

[13] Manfred Hägelen, Rainer Jetten, Jürgen Kassner, and Reinhard Kulke. “Safety and Comfort Enhancement with Radar for a Bicycle Assistance System.” In: *2019 20th International Radar Symposium (IRS)*. IEEE, June 2019, pp. 1–7. DOI: 10.23919/IRS.2019.8768109.

[14] Jiawei Han, Jae-Gil Lee, and Micheline Kamber. “An overview of clustering methods in geographic data analysis.” In: *Geographic data mining and knowledge discovery* 2 (2009), pp. 149–170.

[15] Michael Hardinghaus, Simon Nieland, Marius Lehne, and Jan Weschke. “More than Bike Lanes—A Multifactorial Index of Urban Bikeability.” In: *Sustainability* 13.21 (Oct. 2021), p. 11584. ISSN: 2071-1050. DOI: 10.3390/su132111584.

[16] Alan Hevner and Samir Chatterjee. “Design Science Research in Information Systems.” In: *Design Research in Information Systems*. Boston, MA, USA: Springer, Boston, MA, Mar. 2010, pp. 9–22. DOI: 10.1007/978-1-4419-5653-8_2.

[17] *Infographic: Where Cyclists Are Going Places*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://www.statista.com/chart/25156/share-using-bike-for-transportation-regularly.

[18] Anton Ingemansson, Carl Folkesson, Jonathan Jonsson, Kevin Wingård Olsson, Max Johansson, and Rebecka Bergström. “Classifying road conditions with radar and supervised learning.” In: (2020).

[19] Steven G Johnson and Matteo Frigo. “Implementing FFTs in practice.” In: *Fast Fourier Transforms* 15 (2008).

[20] Douglass B. Lee, Lisa A. Klein, and Gregorio Camus. “Induced Traffic and Induced Demand.” In: *Transp. Res. Rec.* 1659.1 (Jan. 1999), pp. 68–75. ISSN: 0361-1981. DOI: 10.3141/1659-09.

[21] Albrecht K Ludloff. *Praxiswissen radar und radarsignalverarbeitung*. Springer-Verlag, 2013.

[22] Vladimir Mrkajic, Djordje Vukelic, and Andjelka Mihajlov. “Reduction of CO2 emission and non-environmental co-benefits of bicycle infrastructure provision: the case of the University of Novi Sad, Serbia.” In: *Renewable Sustainable Energy Rev.* 49 (Sept. 2015), pp. 232–242. ISSN: 1364-0321. DOI: 10.1016/j.rser.2015.04.100.

[23] Mark J. Nieuwenhuijsen. “Urban and transport planning pathways to carbon neutral, liveable and healthy cities; A review of the current evidence.” In: *Environ. Int.* 140 (July 2020), p. 105661. ISSN: 0160-4120. DOI: 10.1016/j.envint.2020.105661.

[24] Henri J. Nussbaumer. “The Fast Fourier Transform.” In: *Fast Fourier Transform and Convolution Algorithms*. Berlin, Germany: Springer, 1981, pp. 80–111. DOI: 10.1007/978-3-662-00551-4_4.

[25] *Products - Acconeer*. [Online; accessed 25. Nov. 2022]. Oct. 2022. URL: https://www.acconeer.com/products.

[26] John Pucher and Ralph Buehler. "Making Cycling Irresistible: Lessons from The Netherlands, Denmark and Germany." In: *Transport Reviews* 28.4 (July 2008), pp. 495–528. ISSN: 0144-1647. DOI: 10.1080/01441640701806612.

[27] R. Keith Raney. "Maximizing the Intrinsic Precision of Radar Altimetric Measurements." In: *IEEE Geosci. Remote Sens. Lett.* 10.5 (Feb. 2013), pp. 1171–1174. DOI: 10.1109/LGRS.2012.2235138.

[28] Mark A Richards. *Fundamentals of radar signal processing*. McGraw-Hill Education, 2014. ISBN: 9781260468717. URL: https://hds.hebis.de/ulbda/Record/HEB339792248.

[29] Vishal Varun Tipparaju, Kyle R. Mallires, Di Wang, Francis Tsow, and Xiaojun Xian. "Mitigation of Data Packet Loss in Bluetooth Low Energy-Based Wearable Healthcare Ecosystem." In: *Biosensors* 11.10 (Oct. 2021). DOI: 10.3390/bios11100350.

[30] Jorge Vargas, Suleiman Alsweiss, Onur Toker, Rahul Razdan, and Joshua Santos. "An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions." In: *Sensors* 21.16 (Aug. 2021), p. 5397. ISSN: 1424-8220. DOI: 10.3390/s21165397.

[31] Ulla Wandinger. "Introduction to Lidar." In: *Lidar*. New York, NY, USA: Springer, New York, NY, 2005, pp. 1–18. DOI: 10.1007/0-387-25101-4_1.

[32] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Germany: Springer. ISBN: 978-3-662-43839-8.

[33] Abdelmoghit Zaarane, Ibtissam Slimani, Wahban Al Okaishi, Issam Atouf, and Abdellatif Hamdoun. "Distance measurement system for autonomous vehicles using stereo camera." In: *Array* 5 (Mar. 2020), p. 100016. ISSN: 2590-0056. DOI: 10.1016/j.array.2020.100016.

[34] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: A New Data Clustering Algorithm and Its Applications." In: *Data Mining and Knowledge Discovery* 1.2 (June 1997), pp. 141–182. ISSN: 1573-756X. DOI: 10.1023/A:1009783824328.

# 6 SOURCE CODE

Listing 6.1: CFAR peak detection implementation.

```
struct detection
{
    uint16_t peak_bin;
    uint16_t peak_value;
    double peak_distance;
    double threshold;
};
#define detection_t struct detection

detection_t find_peak(uint16_t *data, acc_service_power_bins_metadata_t
    power_bins_metadata)
{
        detection_t detection = {0};

        for (uint16_t i = 0; i < power_bins_metadata.bin_count; i++)
        {
                if (detection.peak_value < data[i]) // don't bother
                    calculating cfar theshold if it's below the current
                    peak anyway
                {
                        // calculate cfar theshold
                        int sum = 0;
                        int count = 0;
                        double distance = power_bins_metadata.start_m +
                            (power_bins_metadata.length_m /
                            power_bins_metadata.bin_count) * i;
                        for (uint16_t j = 0; j < power_bins_metadata.
                            bin_count; j++)
                        {
                                if (j < i - CFAR_GUARD_WIDTH_LEFT || j >
                                     i + CFAR_GUARD_WIDTH_RIGHT) // if
                                    we are outside the CFAR area
                                {
                                        sum += data[j];
                                        count++;
                                }
                        }
                        int cfar_threshold = sum / count;
                        sum = 0;
                        // check if we have a peak at all required bins
                        bool is_peak = true;
                        for (uint16_t j = i - REQUIRED_ADJACIENT_BINS; j
                             <= i + REQUIRED_ADJACIENT_BINS; j++)
                        {
                                int start = (j - (CUT_WIDTH - 1) / 2 < 0
                                     ? 0 : j - (CUT_WIDTH - 1) / 2);
                                int end = (j + (CUT_WIDTH - 1) / 2 >
                                    power_bins_metadata.bin_count ?
                                    power_bins_metadata.bin_count - 1 :
                                    j + (CUT_WIDTH - 1) / 2);
                                for (uint16_t k = start; k <= end; k++)
                                {
                                        sum += data[k];
                                }
                                sum = sum / (end - start + 1); //
                                    calculate average
                                // if weighted amplitude is below
                                    threshold
                                if (sum < ((CFAR_FACTOR * cfar_threshold
                                    ) + (cfar_threshold / distance) *
                                    DISTANCE_FACTOR))
                                {
                                        is_peak = false;
                                }
                        }

                        if (is_peak)
                        {
                                // we have a peak
                                detection.peak_bin = i;
                                detection.threshold = cfar_threshold;
                                detection.peak_value = sum;
                                detection.peak_distance = distance;
                                // if it's the first peak
                                if (detection.first_bin == 0)
                                {
                                        detection.first_bin = i;
                                        detection.first_distance =
                                            distance;
                                        detection.first_value = sum;
                                }
                        }
                }
        }
        return detection;
}
```

Listing 6.2: Distance detector implementation.

```
while (true)
{
    success = acc_detector_distance_get_next(distance_handle, result,
        number_of_peaks, &result_info);

    if (!success)
    {
        printf("acc_detector_distance_get_next() failed\n");
        break;
    }
    if (result_info.number_of_peaks == 0 && broadcast_data.
        measurement_counts > 0) // no detection, with previous data
    {
        if (broadcast_data.measurement_counts > min_count)
        {
            update_beacon();
        }
        else
        {
        }
    }
    else if (result_info.number_of_peaks > 0) // detection
    {
        broadcast_data.measurement_counts++;
        if (broadcast_data.measurement_counts == 1) // first detection
        {
            broadcast_data.avg_distance_mm = (uint16_t)(int16_t)(result
                [0].distance_m * 1000);
            broadcast_data.min_distance_mm = broadcast_data.
                avg_distance_mm;
            broadcast_data.min_distance_amplitude = (uint16_t)(int16_t)(
                result[0].amplitude);
            broadcast_data.background_noise_amplitude = (uint16_t)(
                int16_t)(threshold_sensitivity * 1000);
        }
        else if (broadcast_data.measurement_counts < max_count) // not
            first detection
        {
            broadcast_data.avg_distance_mm = (uint16_t)(int16_t)((
                broadcast_data.avg_distance_mm * (broadcast_data.
                measurement_counts - 1) + result[0].distance_m * 1000) /
                 (broadcast_data.measurement_counts));
            if (broadcast_data.min_distance_mm > result[0].distance_m *
                1000)
            {
                broadcast_data.min_distance_mm = (uint16_t)(int16_t)(
                    result[0].distance_m * 1000);
                broadcast_data.min_distance_amplitude = (uint16_t)(
                    int16_t)(result[0].amplitude);
            }
        }
        else
        {
            update_beacon();
        }
    }
}
```

Listing 6.3: Envelope implementation.

```
while (true)
{
    if (!acc_service_envelope_get_next_by_reference(envelope_handle, &
        envelope_data, &envelope_result_info))
    {
        acc_service_deactivate(envelope_handle);
        success = false;
        printf("acc_service_envelope_get_next_by_reference() failed\n");
        break;
    }
    // print_(envelope_data, envelope_metadata.data_length);
    //  Analyze the data to extract distance, and if the measured object
         has a different velocity than the background.
    //  See https://docs.acconeer.com/en/latest/handbook/a111/services/
        sparse.html for general information on sparse services.
    detection_t detection = ca_cfar_peak_detection(envelope_data,
        envelope_metadata.data_length, &envelope_metadata);
    // printf("Distance: %d, Amplitude: %d, threshold: %f\n", detection.
        index, detection.amplitude, detection.threshold);
    //  after this the data could be discarded, if no fft is needed, and
         a new measurement could be performed in parrallel to the
        calculations below, if a higher measurement rate is needed.
    // the detection condition is distance related, the closer the
        object the higher the expected amplitude.

    if (detection.amplitude > detection.threshold * ((envelope_metadata.
        start_m + envelope_metadata.length_m) / detection.h) &&
        max_count > broadcast_data.measurement_counts)
    {
        if (broadcast_data.measurement_counts == 0) // new measurement
            initialize variables.
        {
            m_oldM = m_newM = detection.h;
            m_oldS = 0.0f;
        }
        else
        {
            m_newM = m_oldM + (detection.h - m_oldM) / broadcast_data.
                measurement_counts;
            m_newS = m_oldS + (detection.h - m_oldM) * (detection.h -
                m_newM);

            // set up for next iteration
            m_oldM = m_newM;
            m_oldS = m_newS;
        }
        broadcast_data.measurement_counts++;
        broadcast_data.avg_distance_mm = (uint16_t)(int16_t)(m_newM *
            1000);
        broadcast_data.avg_distance_variance = (uint16_t)(int16_t)(
            m_newS * 1000); // remember to divide variance later again
        broadcast_data.background_noise_amplitude = ((broadcast_data.
            background_noise_amplitude * (broadcast_data.
            measurement_counts-1)) + detection.threshold) /
            broadcast_data.measurement_counts;
        // check if new distance is smaller than previous one.
        if (broadcast_data.min_distance_mm == 0 || (detection.h * 1000)
            < broadcast_data.min_distance_mm)
        {
            broadcast_data.min_distance_mm = (uint16_t)(int16_t)(
                detection.h * 1000);
            broadcast_data.min_distance_amplitude = detection.amplitude;

        }
    }
    else
    {

        if (broadcast_data.measurement_counts > min_count || count >
            max_count) // if a car was detected or 5 secs elapsed
        {
#ifdef FIXED_MEASUREMENT_INTERVAL
            double car_travel_length = 2.0 * (broadcast_data.
                min_distance_mm / 1000.0) * tan(0.5 * 40 * M_PI / 180.0)
                ;
                                  // calculate the estimated distance
                the car traversed if the sensor would be stationary.
            broadcast_data.avg_speed_cmps = (car_travel_length / ((
                SUSPEND_TIME_BETWEEN_UPDATES_MS / 1000.0)) *
                broadcast_data.measurement_counts) * 100; // calculate
                the estimated relative speed of the car in m per second,
                 by dividing the distance by the time it took to measure
                 it. Due to bluetooth transmission restrictions, convert
                 it to cm/s and save it as uint16_t

#endif

            update_beacon(&broadcast_data);
            count = 0;
        }
        else
        {
            count++;
        }
    }

    // Extract velocity, inter and intra frame velocity. Is the
        approaching car closer to the background? if yes how much in
        relation to the measurement interval -> speed.
    // See https://docs.acconeer.com/en/latest/exploration_tool/algo/
        a111/speed_sparse.html for more information.
    // package results to broadcast via GATT to the phone when the car
        passed the bicycle.
    // Result consists of: distance median or average, shortest distance
        , velocity median or average, highest velocity, duration, and
        some sort of estimation if it really was a car or not, and the
        battery information if relevant.
    //  See https://docs.acconeer.com/en/latest/exploration_tool/algo/
        a111/presence_detection_sparse.html#sparse-presence-detection
        for ideas how to extract the estimation

// Sleep until wakeup
#ifdef FIXED_MEASUREMENT_INTERVAL
    acc_integration_sleep_until_periodic_wakeup();
#endif
}
```

Listing 6.4: Power bin configuration.

```
const uint16_t BIN_SIZE = 20; // mm
const float start_m = 0.5f;
const float length_m = 3.5f;
#define MAX_DURATION 0.5

                            // max duration in seconds until broadcast
    is forced.
#define MEASUREMENT_INTERVAL 25

                    // 25ms = 40Hz
const uint16_t MAX_MEASUREMENTS_PER_BROADCAST = MAX_DURATION * 1000 /
    MEASUREMENT_INTERVAL; // number of measurements to take before
    broadcasting data even when detection is not over.
const uint16_t MIN_MEASUREMENTS_FOR_DETECTION = 2;
const uint16_t CFAR_GUARD_WIDTH_LEFT = 10;
               // left CFAR GUARD
const uint16_t CFAR_GUARD_WIDTH_RIGHT = CFAR_GUARD_WIDTH_LEFT; // right
    CFAR GUARD
const uint16_t REQUIRED_ADJACIENT_BINS = 2;
               // number of adjacient bins, to the left and right that
    must be above threshold. 1 means 3 bins in total, with the center
    bin being the peak.
const double CFAR_FACTOR = 1.4;
                       // used as factor for the calculated cfar
    threshold. 0 means everything will be above threshold. 1 means that
    the threshold is used as is. 2 means the amplitude must be twice the
     claculated threshold
const double DISTANCE_FACTOR = 0.3;
                       // distance is used as a weight for the amplitude
    . 1 means that the amplitude is used as is.

struct detection
{
uint16_t peak_bin;
uint16_t peak_value;
double peak_distance;
double threshold;
};

#define detection_t struct detection

detection_t find_peak(uint16_t *data, acc_service_power_bins_metadata_t
    power_bins_metadata);

/*Initialization code has been omitted*/
```

Listing 6.5: Power bin implementation.

```
while (true)
    {
        success = acc_service_power_bins_get_next(handle, data,
            power_bins_metadata.bin_count, &result_info);

        if (!success)
        {
            printf("acc_service_power_bins_get_next() failed\n");
            break;
        }
        detection_t detection = find_peak(data, power_bins_metadata);

        if (detection.peak_bin != 0) // detection
        {
            // Welford's online algorithm
            broadcast_data.measurement_counts++;
            //  Update avg distance
            int delta = detection.peak_distance * 1000 - broadcast_data.
                avg_distance_mm;
            broadcast_data.avg_distance_mm += delta / (broadcast_data.
                measurement_counts);
            int delta2 = detection.peak_distance * 1000 - broadcast_data
                .avg_distance_mm;
            m_squared += delta * delta2;
            // Update avg amplitude
            broadcast_data.avg_distance_amplitude = (broadcast_data.
                avg_distance_amplitude * (broadcast_data.
                measurement_counts - 1) + detection.peak_value) /
                broadcast_data.measurement_counts;
            // Update min distance
            delta = detection.first_distance * 1000 - broadcast_data.
                min_distance_mm;
            broadcast_data.min_distance_mm += delta / (broadcast_data.
                measurement_counts);
            delta2 = detection.first_distance * 1000 - broadcast_data.
                min_distance_mm;
            min_squared += delta * delta2;
            // Update min amplitude
            broadcast_data.min_distance_amplitude = (broadcast_data.
                min_distance_amplitude * (broadcast_data.
                measurement_counts - 1) + detection.first_value) /
                broadcast_data.measurement_counts;
            // Update background noise value
            broadcast_data.background_noise_amplitude = (broadcast_data.
                background_noise_amplitude * (broadcast_data.
                measurement_counts - 1) + detection.threshold) /
                broadcast_data.measurement_counts;

            missed_measurements = 0;
            if (broadcast_data.measurement_counts >=
                MAX_MEASUREMENTS_PER_BROADCAST &&
                MAX_MEASUREMENTS_PER_BROADCAST != 0) // if limit reached
                , send data
            {
                // retrieve variances
                broadcast_data.avg_distance_variance = m_squared / (
                    broadcast_data.measurement_counts - 1);
                broadcast_data.min_distance_variance = min_squared / (
                    broadcast_data.measurement_counts - 1);
                m_squared = 0;
                min_squared = 0;
                update_beacon(&broadcast_data);
                missed_measurements = 0;
            }
        }
        else
        {
            // no detection
            if (broadcast_data.measurement_counts >
                MIN_MEASUREMENTS_FOR_DETECTION || (missed_measurements >
                 MAX_MEASUREMENTS_PER_BROADCAST &&
                MAX_MEASUREMENTS_PER_BROADCAST != 0)) // old measurement
                 to send, or no detection for too long
            {
                if (missed_measurements > REQUIRED_MISSES)
                {
                    // retrieve variances
                    broadcast_data.avg_distance_variance = m_squared / (
                        broadcast_data.measurement_counts - 1);
                    broadcast_data.min_distance_variance = min_squared /
                         (broadcast_data.measurement_counts - 1);
                    m_squared = 0;
                    min_squared = 0;
                    // send data
                    update_beacon(&broadcast_data);
                    missed_measurements = 0;
                }
                else
                {
                    missed_measurements++;
                }
            }
            else
            {
                if (missed_measurements > MAX_MEASUREMENTS_PER_BROADCAST
                    )
                {
                    missed_measurements = 0;
                    update_beacon(&broadcast_data);
                }
                else
                {
                    missed_measurements++;
                }
            }
        }
#ifdef FIXED_MEASUREMENT_INTERVAL
        // Wait for next measurement interval:
        acc_integration_sleep_until_periodic_wakeup();
#endif
    }
```

Listing 6.6: struct of the broadcasted data.

```
struct BroadcastData
{
             // Structure for the broadcast data, current size is 18
    bytes. using only uint16_t results in nicely padded data, and should
     fit in one ble packet (29 bytes payload i think?)
    uint16_t sequence_id;                                // Sequence id
        of the data, just for debugging and checking no data was lost,
        will eventually overflow and start over.
    uint16_t avg_distance_mm;                    // average distance
        measured in mm
    uint16_t avg_distance_variance;              // variance of the
        distance measurement
    uint16_t avg_distance_amplitude;
    uint16_t min_distance_mm;                    // minimal distance
        measured
    uint16_t min_distance_variance;
    uint16_t min_distance_amplitude;     // amplitude of the minimal
        distance measurement (indicated the likeliness of this being a
        car or other bicycle or pedestrian etc)
    // uint16_t avg_speed_cmps;                  // average speed
        estimation of car, if available in cm/s
    // uint16_t speed_variance;                  // variance of the
        speed estimation
    // uint16_t max_speed_cmps;                  // maximal speed
        estimation of car, if available in cm/s
    uint16_t measurement_counts;                 // number of
        measurements used in the calculation of the above values
    uint16_t background_noise_amplitude; // Initially measured
        background noise amplitude
};
```

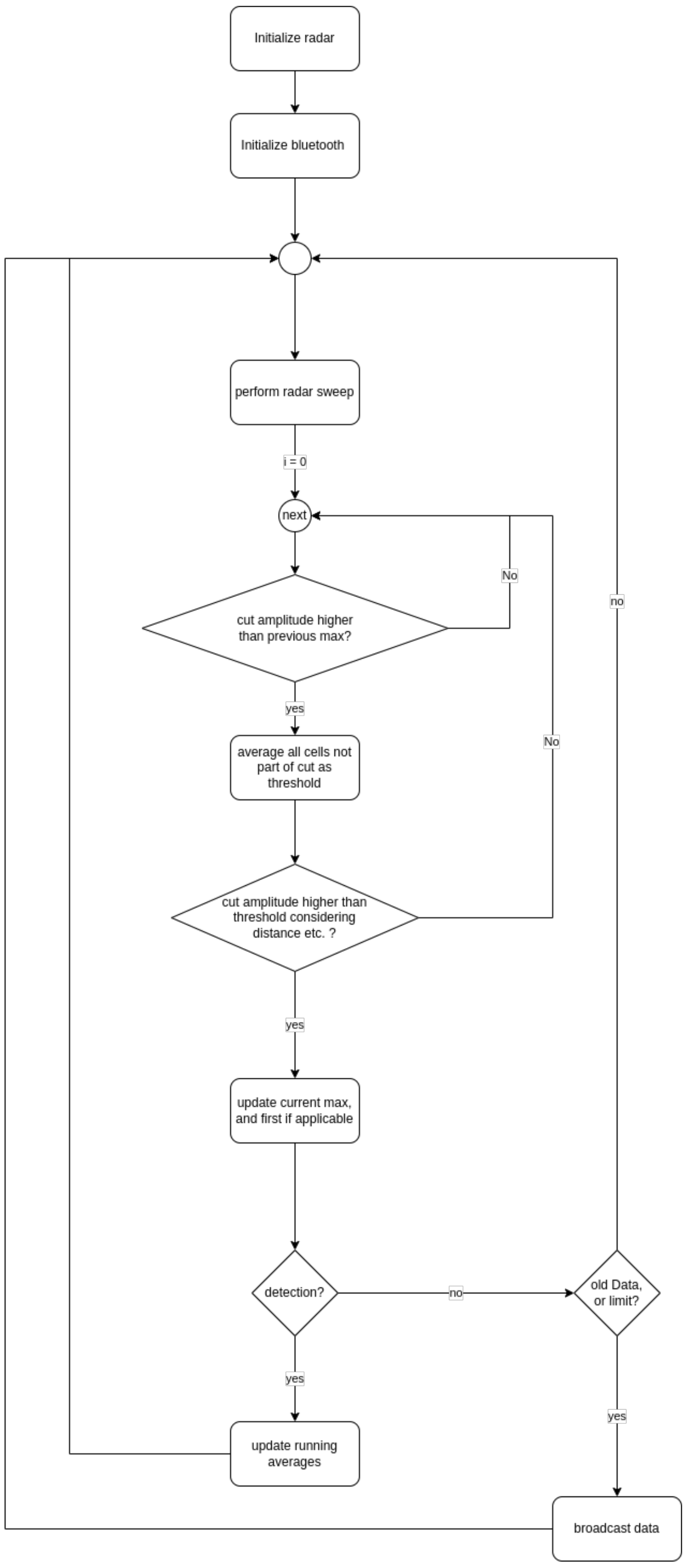

Initialize radar
Initialize bluetooth
perform radar sweep
i = 0
next
cut amplitude higher than previous max?
No
no
yes
average all cells not part of cut as threshold
No
cut amplitude higher than threshold considering distance etc. ?
yes
update current max, and first if applicable
detection?
no
old Data, or limit?
yes
yes
update running averages
broadcast data

Figure 6.1: The flow chart of the implemented Signal processing on the XM122 device. Most notably it consists of two loops, with unknown iterations. The outer loop is the main loop, basically running forever and the inner loop is the loop that handles the aggregation of measurements as long as a car is detected. Once a car was detected at least a given amount of times, the measurements are broadcasted to the phone once it is no longer detected. Each measurement will use an individual threshold, ensuring a constant low false positive rate, by using the CA-CFAR algorithm. This also means that the performed background measurement is an artifact of the previous implementation using a fixed threshold, and is no longer necessary.

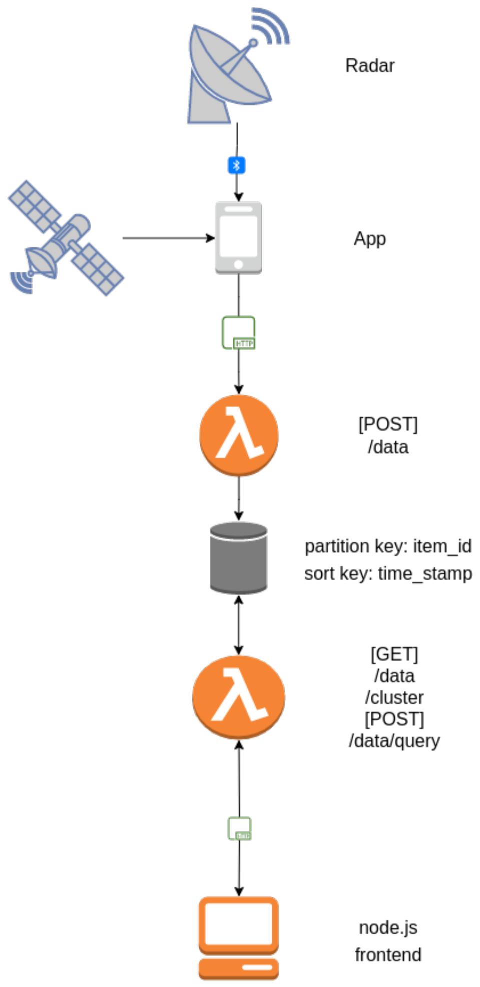


Figure 6.2: Architectural overview of the artifact. The radar transmits measurements via BLE. The Android app adds location information and stores the data. At the end of the tour, the data is uploaded to the backend where it is pre-processed and stored in a dynamodb. A node.js web application is used to visualise the data, and additional lambdas are used to simplify database access.

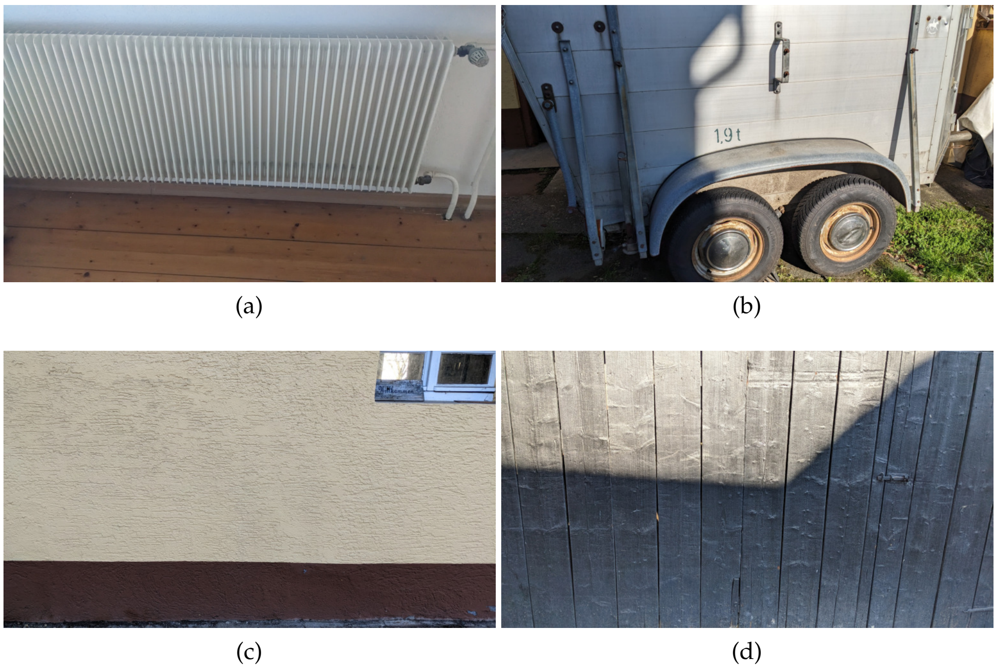
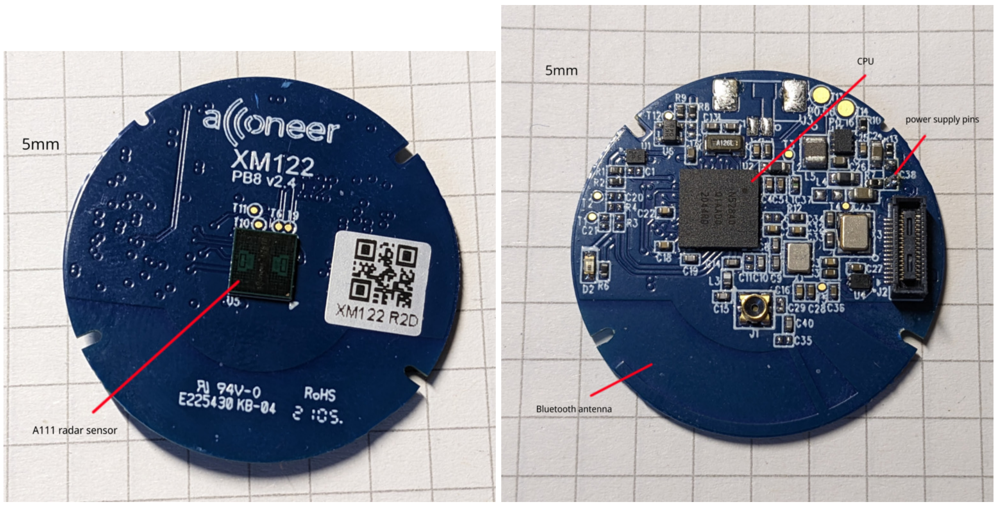


Figure 6.3: images of the selected targets. (a) household radiator. it's RCS is close to a car above 1.5m, but the reflection is stronger at shorter ranges, most likely due to the ribs increasing the RCS(b) trailer. For all intents and purposes shows the RCS of a car(c) wall. (d) wooden wall, etc garden fence

Listing 6.7: sam template.yml to deploy the infrastructure.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An app that receives data from a mobile app
    , assigns a cluster if applicable, and stores the
    data in a DynamoDB table. Furthermore it exposes a
    rest api for querying statistics, raw data and maps.
#Globals:
#   Api:
#     Cors:
#       AllowMethods: "'GET,POST,OPTIONS'"
#       AllowHeaders: "'content-type'"
#       AllowOrigin: "'*'"
#       AllowCredentials: "'*'"
Parameters:
  TABLENAME:
    Type: String
    Description: The DynamoDB table for storing data.
    Default: 'raddar-data'

  REGIONNAME:
```



(a) (b)

Figure 6.4: (a) front view of the sensor. (b) back view of the sensor.

```
      Type: String
      Description: Default region for deployment.
      Default: 'eu-central-1'

   AWSENVNAME:
      Type: String
      Description: AWS Environment where code is being
          executed (AWS_SAM_LOCAL or AWS).
      Default: 'AWS_SAM_LOCAL'


Resources:

   dataHandler:
      # A function that writes to a DynamoDB table on a
          schedule
      Type: 'AWS::Serverless::Function'
      Properties:
         Handler: handleData.lambda_handler
         Runtime: python3.9
         CodeUri: src/.
         Description: A function that handles the data from
             a DynamoDB table
         MemorySize: 1024
         Timeout: 240
         Policies:
         # Read more about SAM policy templates here
```

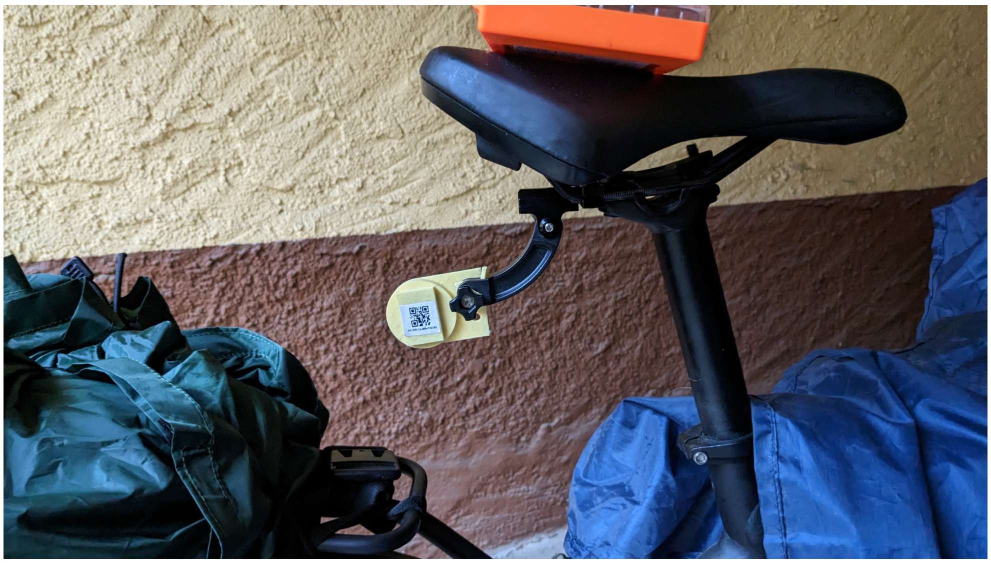


Figure 6.5: Picture of the mounted sensor. The sensor is mounted under the saddle, and the power supply can be either an external powerbank, an integrated 5V battery or any other supply such as compatible e-bikes and phones. The handlebar width to each side is 0.35m, resulting in a total width of 0.7m. This means the sensors measurements, have to be reduced by 0.35m in order to get the actual distance to the object, because the sensor is mounted in the center. This also compensates for the blindspot of 0.3m that we have due to possible direct leakage of the radar signal.

```
# https://docs.aws.amazon.com/serverless-
    application-model/latest/developerguide/
    serverless-policy-templates.html
  - AWSLambdaExecute
  - DynamoDBCrudPolicy:
      TableName: !Ref DataTable
Environment:
   Variables:
      TABLE: !Ref TABLENAME
      REGION: !Ref REGIONNAME
      AWSENV: !Ref AWSENVNAME
Events:
   HttpPost:
      Type: Api
      Properties:
         Path: '/data'
         Method: POST
   HttpGet:
      Type: Api
      Properties:
         Path: '/data'
         Method: GET
```

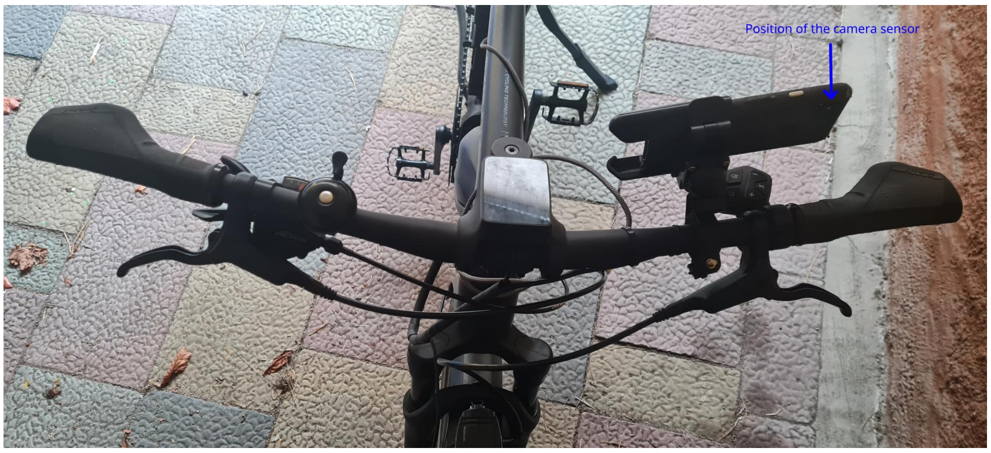
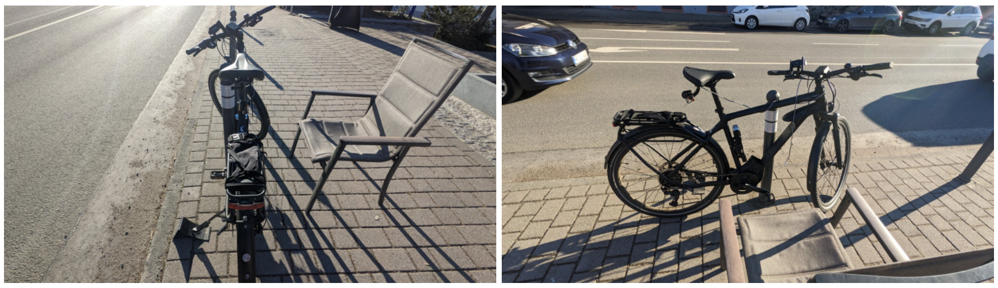


Figure 6.6: shows the mount point of the camera. This is relevant, when considering a visual interpretation of the recorded situations. The leftmost mounting of the camera, causes the visual impression of the car being closer to the cyclist, while at the same time causing the impression that the cyclist was cycling not as far to the right as he actually was.



(a) (b)

Figure 6.7: (a) back view. (b) side view.

```
clusterHandler:
  # A function that writes to a DynamoDB table on a
      schedule
  Type: 'AWS::Serverless::Function'
  Properties:
    Handler: getClusters.lambda_handler
    Runtime: python3.9
    CodeUri: src/.
    Description: A function to query clusters in an
        area
    MemorySize: 1024
    Timeout: 240
    Policies:
    # Read more about SAM policy templates here
    # https://docs.aws.amazon.com/serverless-
        application-model/latest/developerguide/
        serverless-policy-templates.html
```

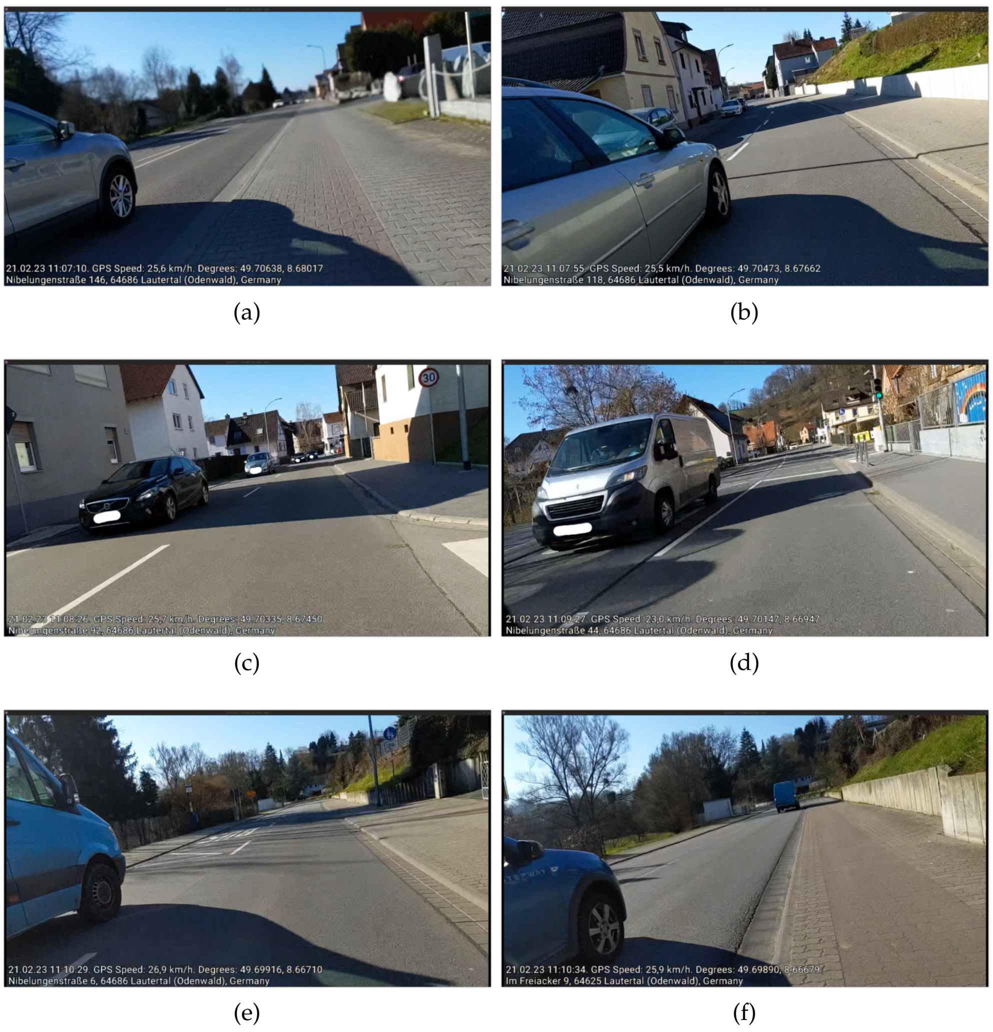

21.02.23 11:07:10. GPS Speed: 25,6 km/h. Degrees: 49.70638, 8.68017
Nibelungenstraße 146, 64686 Lautertal (Odenwald), Germany
(a)
21.02.23 11:07:55. GPS Speed: 25,5 km/h. Degrees: 49.70473, 8.67662
Nibelungenstraße 118, 64686 Lautertal (Odenwald), Germany
(b)
21.02.23 11:08:26. GPS Speed: 25,7 km/h. Degrees: 49.70335, 8.67450
Nibelungenstraße 92, 64686 Lautertal (Odenwald), Germany
(c)
21.02.23 11:09:27. GPS Speed: 23,0 km/h. Degrees: 49.70147, 8.66947
Nibelungenstraße 44, 64686 Lautertal (Odenwald), Germany
(d)
21.02.23 11:10:29. GPS Speed: 26,9 km/h. Degrees: 49.69916, 8.66710
Nibelungenstraße 6, 64686 Lautertal (Odenwald), Germany
(e)
21.02.23 11:10:34. GPS Speed: 25,9 km/h. Degrees: 49.69890, 8.66679
Im Freiacker 9, 64625 Lautertal (Odenwald), Germany
(f)

Figure 6.8: those are the reference images. The images are taken from the reference video.

```
        - AWSLambdaExecute
        - DynamoDBCrudPolicy:
            TableName: !Ref DataTable
    Environment:
      Variables:
        TABLE: !Ref TABLENAME
        REGION: !Ref REGIONNAME
        AWSENV: !Ref AWSENVNAME
    Events:
      HttpGet:
        Type: Api
        Properties:
          Path: '/data/cluster'
          Method: GET
```

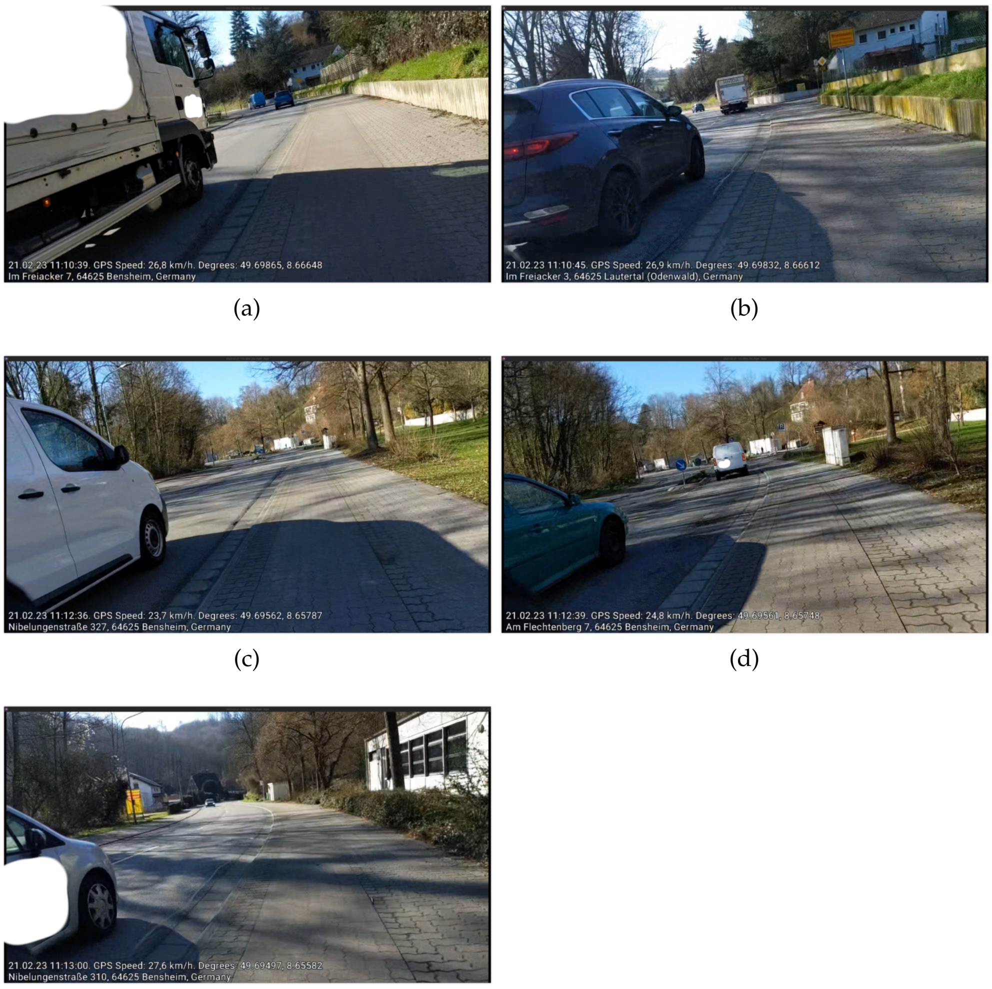

21.02.23 11:10:39. GPS Speed: 26,8 km/h. Degrees: 49.69865, 8.66648
Im Freiacker 7, 64625 Bensheim, Germany
(a)
21.02.23 11:10:45. GPS Speed: 26,9 km/h. Degrees: 49.69832, 8.66612
Im Freiacker 3, 64625 Lautertal (Odenwald), Germany
(b)
21.02.23 11:12:36. GPS Speed: 23,7 km/h. Degrees: 49.69562, 8.65787
Nibelungenstraße 327, 64625 Bensheim, Germany
(c)
21.02.23 11:12:39. GPS Speed: 24,8 km/h. Degrees: 49.69561, 8.65748
Am Flechtenberg 7, 64625 Bensheim, Germany
(d)
21.02.23 11:13:00. GPS Speed: 27,6 km/h. Degrees: 49.69497, 8.65582
Nibelungenstraße 310, 64625 Bensheim, Germany
(e)

Figure 6.9: shows the remaining reference images . The images are taken from the reference video.

```
queryHandler:
  # A function that writes to a DynamoDB table on a
     schedule
  Type: 'AWS::Serverless::Function'
  Properties:
    Handler: queryData.lambda_handler
    Runtime: python3.9
    CodeUri: src/.
    Description: A function to query data
    MemorySize: 1024
    Timeout: 240
    Policies:
    # Read more about SAM policy templates here
```

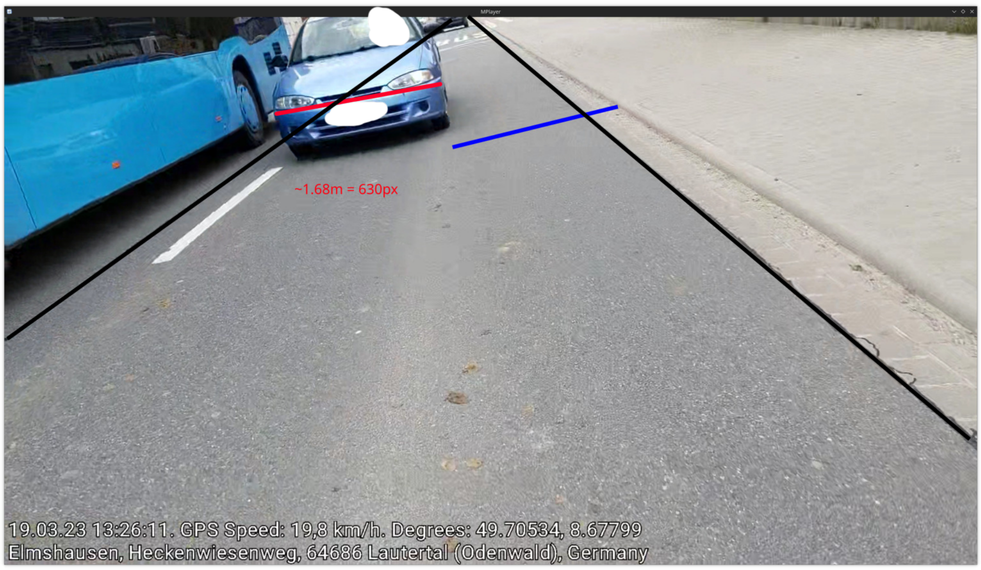

~1.68m = 630px
19.03.23 13:26:11. GPS Speed: 19,8 km/h. Degrees: 49.70534, 8.67799
Elmshausen, Heckenwiesenweg, 64686 Lautertal (Odenwald), Germany

Figure 6.10: oncoming traffic example representing a value measurement. Google Lens identified the car as a Mitsubishi colt of the fifth generation with a width of 1.68m (Without mirrors). A rough estimation let us conclude that in this case the remaining street width is approximately 1.68m as well.

```
          # https://docs.aws.amazon.com/serverless-
            application-model/latest/developerguide/
            serverless-policy-templates.html
          - AWSLambdaExecute
          - DynamoDBCrudPolicy:
              TableName: !Ref DataTable
        Environment:
          Variables:
            TABLE: !Ref TABLENAME
            REGION: !Ref REGIONNAME
            AWSENV: !Ref AWSENVNAME
        Events:
          HttpGet:
            Type: Api
            Properties:
              Path: '/data/query'
              Method: GET
        Events:
          HttpGet:
            Type: Api
            Properties:
              Path: '/data/query'
              Method: POST
    graphHandler:
```

```
    # A function that writes to a DynamoDB table on a
        schedule
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: getGraphs.lambda_handler
      Runtime: python3.9
      CodeUri: src/.
      Description: A function to query graphs
      MemorySize: 1024
      Timeout: 240
      Policies:
      # Read more about SAM policy templates here
      # https://docs.aws.amazon.com/serverless-
          application-model/latest/developerguide/
          serverless-policy-templates.html
        - AWSLambdaExecute
        - DynamoDBCrudPolicy:
            TableName: !Ref DataTable
      Environment:
        Variables:
          TABLE: !Ref TABLENAME
          REGION: !Ref REGIONNAME
          AWSENV: !Ref AWSENVNAME
      Events:
        HttpGet:
          Type: Api
          Properties:
            Path: '/data/graph'
            Method: GET

  DataTable:
    Type: AWS::DynamoDB::Table
    DeletionPolicy: Retain
    Properties:
      TableName: !Ref TABLENAME
      AttributeDefinitions:
        - AttributeName: item_id
          AttributeType: S
        - AttributeName: time_stamp
          AttributeType: N
      KeySchema:
        - AttributeName: id
          KeyType: HASH
        - AttributeName: time_stamp
          KeyType: RANGE
      ProvisionedThroughput:
        ReadCapacityUnits: 1
        WriteCapacityUnits: 1
```

Listing 6.8: python handler to add and scan all data.

```
import simplejson as json
import os
import boto3
import logging
client = boto3.client('dynamodb')
log = logging.getLogger()
log.setLevel(logging.DEBUG)


def lambda_handler(event, context):
    table_name = os.environ['TABLE']
    region = os.environ['REGION']
    aws_environment = os.environ['AWSENV']
    if os.environ['AWS_SAM_LOCAL']:
        data_table = boto3.resource(
            'dynamodb', endpoint_url="http://dynamodb:8000").Table(
                table_name)
    else:
        data_table = boto3.resource('dynamodb').Table(table_name)
    log.debug(event['body'])
    if event['httpMethod'] == 'GET':
        return load_data(data_table)
    elif event['httpMethod'] == 'POST':
        return store_data(data_table, json.loads(event['body'])['data'])
    elif event['httpMethod'] == 'OPTIONS':
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'X-Requested-With': '*',
                'Access-Control-Allow-Headers': 'Content-Type,X-Amz-Date
                    ,Authorization,X-Api-Key,x-requested-with',
                'Access-Control-Allow-Methods': 'POST,GET,OPTIONS'}
        }
    else:
        return {
            'statusCode': 501,
            'body': json.dumps('Not implemented!')
        }


def load_data(data_table):
    log.info("Loading items from table")
    response = data_table.scan(
    )
    log.info(response)
    return {
        'statusCode': 200,
        'body': json.dumps(response['Items']),
        'headers': {
            'count': response['Count'],
            'Access-Control-Allow-Origin': '*',
            'X-Requested-With': '*',
            'Access-Control-Allow-Headers': 'Content-Type,X-Amz-Date,
                Authorization,X-Api-Key,x-requested-with',
            'Access-Control-Allow-Methods': 'POST,GET,OPTIONS'}
    }


def store_data(data_table, data):
    log.info("Adding {} items to table".format(len(data)))
    for item in data:
        log.debug(item)
        try:
            # convert data to dynamodb comaptible values if they are in
                the post data
            if 'speed' in item:
                item['speed'] = int(item['speed']*3.6)
            if 'ownSpeedAccuracy' in item:
                item['ownSpeedAccuracy'] = int(item['ownSpeedAccuracy'
                    ]*3.6)
            # round gps accuracy to integer
            if 'gpsAccuracy' in item:
                item['gpsAccuracy'] = int(item['gpsAccuracy']*10)
            if 'bearing' in item:
                item['bearing'] = int(item['bearing']*10)

            # convert lat and lon to integer
            if 'lat' in item:
                item['lat'] = int(item['lat']*1000000000)
            if 'lon' in item:
                item['lon'] = int(item['lon']*1000000000)
            response = data_table.put_item(Item=item)
        except Exception as e:
            log.error("Error adding item to table")
            log.error(item)
            return {
                'statusCode': 400,
                'body': json.dumps(str(e)),
            }
        log.info(response)
        if response['ResponseMetadata']['HTTPStatusCode'] != 200:
            log.error("Error adding item to table")
            log.error(item)
            return response
    return {
        'statusCode': 200,
        'body': ""
    }
```

Listing 6.9: python handler to query specific data.

```
import simplejson as json
import sys
import os
import boto3
import logging

client = boto3.client('dynamodb')
log = logging.getLogger()
# log.setLevel(logging.ERROR)


def lambda_handler(event, context):
    table_name = os.environ['TABLE']
    region = os.environ['REGION']
    aws_environment = os.environ['AWSENV']
    # if this is running in a local environment, overwrite the endpoint
        url
    if os.environ['AWS_SAM_LOCAL']:
        data_table = boto3.resource(
            'dynamodb', endpoint_url="http://dynamodb:8000").Table(
                table_name)
    else:
        data_table = boto3.resource('dynamodb').Table(table_name)

    if event['httpMethod'] == 'GET' or event['httpMethod'] == 'POST':

        body = json.loads(event['body'])
        if len(body) == 0:
            return {'statusCode': 400, 'body': json.dumps('No query
                parameters provided!')}
        filterExpressionString = ""
        filterExpressionAttributeValues = {}

        # for all but the last parameter, add AND to the filter string
        for key, values in body.items():
            # TODO: cluster ID is a special case, as all cluster ID's
                should be returned
            # if key == "cluster_id":
            #    filterExpressionString += key + " = :" + key + " OR  "
            #    filterExpressionAttributeValues[":" + key] = values
            # when item ID is selected, only return the items with the
                given ID
            if key == "item_id":
                filterExpressionString += key + " = :" + key + " AND "
                filterExpressionAttributeValues[":" + key] = values
            else:
                # for all other parameters, select greater than min and
                    less than max
                filterExpressionString += key + " BETWEEN :" + \
                    key + "Min AND :" + key + "Max AND "
                filterExpressionAttributeValues[":" + key + "Min"] =
                    values[0]
                filterExpressionAttributeValues[":" + key + "Max"] =
                    values[1]

        # remove the last AND
        filterExpressionString = filterExpressionString[:-4]

        return load_data(data_table, filterExpressionString,
            filterExpressionAttributeValues)
    else:
        return {
            'statusCode': 501,
            'body': json.dumps('Not implemented!')
        }


def load_data(data_table, selectedFilterString,
    selectedFilterAttributeValues):
    loadedItems = []
    log.info("Loading items from table")
    response = data_table.scan(
        FilterExpression=selectedFilterString,
        ExpressionAttributeValues=selectedFilterAttributeValues
    )
    # append data
    loadedItems.extend(response['Items'])
    # if response is chuncked, load all items
    while 'LastEvaluatedKey' in response:
        response = data_table.scan(
            FilterExpression=selectedFilterString,
            ExpressionAttributeValues=selectedFilterAttributeValues,
            ExclusiveStartKey=response['LastEvaluatedKey']
        )
        loadedItems.extend(response['Items'])

    log.info("Items loaded from table")
    return {
        'statusCode': 200,
        'body': json.dumps(loadedItems),
        'headers': {
            'count': len(loadedItems),
            'Access-Control-Allow-Origin': '*'}
    }
```

Listing 6.10: python handler to query for clusters.

```
import simplejson as json
import os
import boto3
import logging
client = boto3.client('dynamodb')
log = logging.getLogger()
#log.setLevel(logging.DEBUG)
```

```
def lambda_handler(event, context):
    table_name = os.environ['TABLE']
    region = os.environ['REGION']
    aws_environment = os.environ['AWSENV']
    if os.environ['AWS_SAM_LOCAL']:
        data_table = boto3.resource(
            'dynamodb', endpoint_url="http://dynamodb:8000").Table(
                table_name)
    else:
        data_table = boto3.resource('dynamodb').Table(table_name)

    if event['httpMethod'] == 'GET':
        return load_data(data_table)
    else:
        return {
            'statusCode': 501,
            'body': json.dumps('Not implemented!')
        }


def load_data(data_table):
    loadedItems = []
    # get all data where clusterID is not -1
    response = data_table.scan(
        FilterExpression="clusterID > :clusterID",
        ExpressionAttributeValues={
            ':clusterID': -1
        }
    )

    # append data
    loadedItems.extend(response['Items'])
    # if response is chuncked, load all items
    while 'LastEvaluatedKey' in response:
        response = data_table.scan(
            FilterExpression="clusterID > :clusterID",
            ExpressionAttributeValues={
                ':clusterID': -1
            },
            ExclusiveStartKey=response['LastEvaluatedKey']
        )

        loadedItems.extend(response['Items'])

    log.info("Items loaded from table")
    return {
        'statusCode': 200,
        'body': json.dumps(loadedItems),
        'headers': {
            'count': len(loadedItems),
            'Access-Control-Allow-Origin': '*'}
    }
```

Listing 6.11: bootstrap script.

```
#!/usr/bin/env bash
set -e
# create the dynamodb container with
# docker network create lambda-local
# docker run -d -v "$PWD":/dynamodb_local_db -p 8000:8000 --network
    lambda-local --name dynamodb amazon/dynamodb-local
docker start dynamodb

aws dynamodb create-table --table-name raddar-data --attribute-
    definitions AttributeName=item_id,AttributeType=S AttributeName=
    time_stamp,AttributeType=N --key-schema AttributeName=item_id,
    KeyType=HASH AttributeName=time_stamp,KeyType=RANGE --provisioned-
    throughput ReadCapacityUnits=5,WriteCapacityUnits=5 --endpoint-url
    http://localhost:8000

cd raddar-backend
sam local start-api --docker-network lambda-local
docker stop dynamodb
```

## 6.1 TABLES

Table 6.1: Statistics of each individual measurement series for the envelope service. The mean, standard deviation, count, minimum and maximum of the distance measurements are shown. For each series the standard deviation is small, indicating very consistent measurements, but some series have measurements that are siginficantly different from the actual setup, indicating the experimental setup was not stable.

| | | mean (m) | std (mm) | count | minimum (m) | maximum (m) |
|---|---|---|---|---|---|---|
| datetime | note | | | | | |
| 2023-01-09 | radiator - 0.5 | 0.496 | 1.67 | 214 | 0.493 | 0.504 |
| | radiator - 1 | 1.016 | 0.51 | 228 | 1.014 | 1.017 |
| | radiator - 1.5 | 1.549 | 1.07 | 217 | 1.545 | 1.552 |
| | radiator - 2 | 2.028 | 0.61 | 250 | 2.027 | 2.034 |
| | radiator - 2.5 | 2.510 | 0.52 | 169 | 2.509 | 2.512 |
| | radiator - 3 | 3.002 | 6.94 | 175 | 2.986 | 3.018 |
| 2023-01-10 | radiator - 0.5 | 0.530 | 0.68 | 222 | 0.524 | 0.531 |
| | radiator - 1 | 1.012 | 0.38 | 211 | 1.011 | 1.013 |
| | radiator - 1.5 | 1.557 | 1.18 | 153 | 1.554 | 1.561 |

Continued on next page

Table 6.1: Statistics of each individual measurement series for the envelope service. The mean, standard deviation, count, minimum and maximum of the distance measurements are shown. For each series the standard deviation is small, indicating very consistent measurements, but some series have measurements that are siginficantly different from the actual setup, indicating the experimental setup was not stable.

| | | mean (m) | std (mm) | count | minimum (m) | maximum (m) |
|---|---|---|---|---|---|---|
| datetime | note | | | | | |
| | radiator - 2 | 2.031 | 0.53 | 157 | 2.030 | 2.032 |
| | radiator - 2.5 | 2.533 | 0.65 | 210 | 2.531 | 2.534 |
| | radiator - 3 | 3.029 | 0.88 | 218 | 3.026 | 3.031 |
| 2023-01-14 | radiator - 0.5 | 0.514 | 0.55 | 233 | 0.513 | 0.518 |
| | radiator - 1 | 1.064 | 0.92 | 222 | 1.061 | 1.071 |
| | radiator - 1.5 | 1.709 | 2.59 | 212 | 1.703 | 1.717 |
| | radiator - 2 | 2.042 | 0.58 | 69 | 2.041 | 2.043 |
| | radiator - 2.5 | 2.523 | 0.67 | 87 | 2.521 | 2.524 |
| | radiator - 3 | 3.062 | 0.98 | 108 | 3.059 | 3.064 |
| 2023-01-19 | snow - 2.5 | 2.516 | 1.20 | 280 | 2.511 | 2.518 |
| 2023-01-26 | wall - 0.5 | 0.503 | 0.75 | 176 | 0.501 | 0.505 |
| | wall - 1 | 0.642 | 202.38 | 215 | 0.485 | 1.476 |
| | wall - 1.5 | 1.503 | 1.27 | 145 | 1.500 | 1.506 |
| | wall - 2 | 1.989 | 1.53 | 223 | 1.985 | 1.992 |
| | wall - 2.5 | 2.365 | 315.46 | 130 | 0.835 | 3.048 |
| | wall - 3 | 2.795 | 432.22 | 57 | 1.395 | 3.395 |
| 2023-01-27 | trailer - 0.5 | 0.488 | 4.89 | 202 | 0.482 | 0.505 |
| | trailer - 1 | 1.020 | 1.83 | 240 | 0.999 | 1.031 |
| | trailer - 1.5 | 1.510 | 0.50 | 170 | 1.509 | 1.511 |
| | trailer - 2 | 2.004 | 0.77 | 225 | 2.002 | 2.007 |
| | trailer - 2.5 | 2.485 | 1.19 | 214 | 2.482 | 2.488 |
| | trailer - 3 | 2.995 | 1.65 | 211 | 2.973 | 2.997 |

Table 6.2: Statistics of each individual measurement series. The mean, standard deviation, count, minimum and maximum of the distance measurements are shown. For each series the standard deviation is small, indicating very consistent measurements, but some series have measurements that are siginficantly different from the actual setup, indicating the experimental setup was not stable.

| | | mean (m) | std (mm) | count | minimum (m) | maximum (m) |
|---|---|---|---|---|---|---|
| datetime | note | | | | | |
| 2023-02-15 | Trailer - 0.5 | 0.522 | 2.94 | 733 | 0.518 | 0.531 |
| | Trailer - 1 | 1.004 | 2.71 | 687 | 0.997 | 1.011 |
| | Trailer - 1.5 | 1.514 | 3.24 | 502 | 1.506 | 1.524 |
| | Trailer - 2 | 2.014 | 4.43 | 347 | 2.004 | 2.025 |
| | Trailer - 2.5 | 2.506 | 3.06 | 432 | 2.499 | 2.516 |
| | radiator - 0.5 | 0.506 | 2.43 | 852 | 0.499 | 0.512 |
| | radiator - 1 | 1.006 | 2.42 | 567 | 0.999 | 1.010 |
| | radiator - 1.5 | 1.507 | 2.24 | 757 | 1.500 | 1.513 |
| | radiator - 2 | 2.020 | 3.39 | 514 | 2.009 | 2.033 |
| | radiator - 2.5 | 2.502 | 4.07 | 782 | 2.489 | 2.518 |
| 2023-02-16 | wall - 0.5 | 0.500 | 3.22 | 781 | 0.491 | 0.509 |
| | wall - 1 | 1.009 | 2.01 | 858 | 0.999 | 1.017 |
| | wall - 1.5 | 1.509 | 3.89 | 361 | 1.498 | 1.531 |

Table 6.3: Relevant locations and their safety index, with measurements only below the safety distance.

| | longitude | latitude | avgDistanceM | avgAmplitude | measurementCount |
|---|---|---|---|---|---|
| 0 | 8.619861 | 49.682885 | 0.535 | 2401.0 | 1 |
| 1 | 8.619971 | 49.682871 | 0.701 | 2247.0 | 1 |
| 45 | 8.635953 | 49.684804 | 0.787 | 3563.0 | 1 |
| 109 | 8.644774 | 49.688862 | 0.807 | 3639.0 | 1 |
| 36 | 8.633829 | 49.683619 | 0.828 | 2122.0 | 1 |
| 47 | 8.636054 | 49.684848 | 0.837 | 2193.0 | 1 |
| 9 | 8.624846 | 49.681274 | 0.859 | 1782.0 | 1 |
| 176 | 8.663181 | 49.695655 | 0.996 | 2804.0 | 1 |
| 49 | 8.636510 | 49.685105 | 1.005 | 3120.0 | 1 |
| 15 | 8.625725 | 49.681548 | 1.021 | 2950.0 | 1 |

Table 6.4: Identified clusters with 0.05km as eps and 11 required points

| ID | Avg. [m] | Count | Min. [m] | Latitude | Longitude | Min. Latitude | Max. Latitude | Min. Longitude | Max. Longitude |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.52 | 15 | 1.00 | 49.6957 | 8.6634 | 49.695621 | 49.695786 | 8.662704 | 8.664252 |
| 2 | 1.57 | 18 | 0.81 | 49.6884 | 8.6444 | 49.687746 | 49.688982 | 8.643885 | 8.644774 |
| 0 | 1.61 | 12 | 1.10 | 49.6860 | 8.6393 | 49.685913 | 49.686016 | 8.638694 | 8.639685 |