Infrasound Detection of Meteoroids

MSc. Thesis report

Jarno de Wit

Delft University of Technology

TU Delft

Infrasound Detection of Meteoroids

MSc. Thesis report

by

Jarno de Wit

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on April 24, 2026 at 13:00.

Student number: 5327156
Project duration: March 24, 2025 – March 15, 2026
Thesis committee:

Dr.ir. W. van der Wal	TU Delft, chair
Dr. S.J. de Vet	TU Delft, supervisor
Dr.ir. B.C. Root	TU Delft, committee member

Cover: Long exposure image of the Geminid meteor shower, 2009 by NASA under CC BY-NC 2.0 (Modified)

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

The investigation outlined in this report has been carried out as part of the MSc Aerospace Engineering at Delft University of Technology. The goal of this thesis was to assess the approach for detecting meteor entries using infrasonic signals, a process which I often described to friends and family simply as ”listening to falling rocks”. Although the subject is of course significantly more complex than just listening, taking a step back in such a way can often help regaining the picture of what you are actually attempting to achieve.

During the investigation, I had the pleasure of working with many very smart people, each of whom assisted me in their own amazing ways. From within the TU Delft, I would first like to thank Sebastiaan de Vet, who helped guide me through this research as the thesis supervisor for this project. In times where I might have lost sight of what I was trying to achieve, it were the common progress meetings which helped me get back on track. Next up, I would like to thank Martin Søndergaard, the technical genius responsible for the rooftop lab of the TU Delft. It would not have been possible to perform the experiments without all the technical knowledge he provided. Special thanks also go out to Jelle Assink at the KNMI (Royal Netherlands Meteorological Institute), who provided valuable insight into the setup and usage of infrasound sensors through his experience with the technology.

I would also like to thank my family for their unwavering support during this process and for the wide selection of random tools and random components needed for constructing and mounting the different filter setups to the infrasound sensors. Next, I would like to thank my friends who, despite all our busy schedules, still found time to encourage me during this process and provide the needed relaxations during many of the late night (board) game sessions.

Lastly, I want to thank you, the reader, for taking the time to read this report. I sincerely hope you enjoy reading this even more than I enjoyed writing this report, and that hopefully you learn something new.

Jarno de Wit
Delft, March 2026

Abstract

As fragments floating through space, asteroids contain a lot of information regarding the history and development of our universe. As such, for a long time humans have been fascinated by meteor entries into the atmosphere, and have devised large scale camera networks to track the entries of such meteors. Despite the effort put into these camera networks, this approach to detecting meteors has serious limitations regarding its effectiveness during daytime or during cloudy nights as this prevents the fireballs from being visible. As such, there is interest in expanding these detection networks with infrasound sensors, highly sensitive microphones which can detect the low frequency sound that propagates from a meteoric entry shockwave.

A lot is still unknown regarding the effective implementation of such infrasound sensors into a detection network. As such, this study investigates three important aspects of such a potential implementation, being the detection pipeline/algorithm, methods for reducing the wind noise, and the layout of a potential future network.

Using publicly available infrasound data and meteor detections, a pipeline was developed for detecting meteoric entry signatures in this data. Although some likely meteor signatures were found within this data, the true detection rate is only around 3%-4% of the events included in this analysis, with a similar number of false positive detections being flagged.

To achieve better detections in a properly integrated network, two different approaches have been tested for reducing wind noise at the infrasound sensor. Both the tested porous hose filter and the fabric dome filter successfully reduced the measured noise level by 10 to 15 dB between 10 and 30 Hz. At slightly higher frequencies, the porous hose filter started to perform worse as a filter, and began introducing artifacts into even coherent signals.

Lastly, the layout of a potential infrasound network within the Netherlands has been investigated. For this network, three infrasound sensors would be placed throughout the Netherlands, with stations guaranteed at Delft University of Technology and Tilburg, and a third sensor at a yet undetermined location. From the available candidate locations, the Elburg location was found to provide the most versatile detection network for future research into infrasound detection of meteors.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
ASGARD	All Sky and Guided Automatic Real-time Detection
CAMO	Canadian Automated Meteor Observatory
CTBTO	Comprehensive Nuclear Test-Ban Treaty Organisation
DOERAK	Dutch Observers of Entries to Recover Asteroidal Krumbs
EN	European (Meteor) Network
FDSN	Federation of Digital Seismograph Networks
FFT	Fast Fourier Transform
FRIPON	Fireball Recovery and InterPlanetary Observation Network
GMN	Global Meteor Network
IMS	International Monitoring System
ISA	International Standard Atmosphere
PSD	Power Spectral Density
STD	Standard Deviation

Symbols

Symbol	Definition	Unit
f	Frequency	[Hz]
R_0	Blast radius	[m]
T	Temperature	[K]
t	Time	[s]
V	Velocity	[m/s]
γ	Specific heat ratio	[-]
λ	Lapse rate	[K m^{-1}]
ρ	Density	[kg/m^3]

1

Introduction

More than 66 million years ago, a large rock found its way through the earths' atmosphere, producing a bright flash and a tremendous sound as it forced its way past the air molecules blocking its path. This meteor impact marked the end of the era of dinosaurs walking the earth.

The same way that this meteor tells a story about the history of life on earth, there are many more asteroids which can tell us about the history and formation of the planets, life, and the universe. Every day, some of these asteroids arrive at the earth as meteors, producing brilliant spectacles of light across the night sky, at which we gaze with wide open eyes. But what could we learn from these meteors if we didn't just look at their fireballs, but also listened to what stories they can tell?

1.1. Context

Historically, the main way that researchers have sought to detect meteor entries into the earths' atmosphere has been through the use of large scale camera networks continuously scanning the sky for fireballs. Although this approach has provided many successful meteor detections, these networks tend to struggle when visibility of the night sky is obstructed. As such, an alternative approach for meteor detections through the use of low frequency sound measurements has been identified as a way to augment such camera networks. Although there have been multiple studies into the physics of the shockwave generation and propagation (Drob *et al.*, 2003; Edwards & Hildebrand, 2004; Edwards, 2009; Haynes & Millet, 2013; ReVelle, 1976), these studies generally start from the assumption that an infrasound signal has been effectively recorded. For larger meteors, this detection can be done using the sparse but global CTBTO infrasound network, detecting long range infrasound signals that propagate around the earth through atmospheric ducts (ElGabry *et al.*, 2017; Ens *et al.*, 2012). For smaller meteors, which do not create powerful enough shocks to be detected at such ranges, research has remained more limited with the investigations that have been done relying on specialised infrasound sensors (Edwards *et al.*, 2008; Silber *et al.*, 2015) which limit their use for a large scale deployment due to their cost and complexity. As such how infrasound measurements

could however be effectively integrated into the dense camera based detection networks to expand their capabilities has mostly evaded extensive studies.

One of these camera based detection networks currently in operation is the FRIPON camera network, which has cameras spread throughout Europe, Canada, Southern America, Africa, and Australia. Within this network, the cameras in multiple countries are operated by regional branches / sub-networks, such as the DOMe network in Canada, the SCAMP network in the United Kingdom, and the DOERAK network in the Netherlands. Within the Dutch DOERAK branch, the desire has been growing to expand the network capabilities by integrating infrasound detection capabilities into the existing network of fireball detection cameras. As such, this research will focus on the considerations for setting up and running such a network within this Dutch sub-network where such a focus is necessary.

1.2. Research questions

Based on the context given in section 1.1, the main goal of this research can be formulated as follows:

How can the detection capabilities of meteor detection networks be improved through infrasound infrasound measurements.

Since this research question is still comparatively broad, this question has been split into the following three sub-questions, each of which can be investigated separately during the course of this research:

1. How can meteor generated infrasound signature be automatically detected from infrasound sensors?
2. How can the noise characteristics for infrasound sensors be optimised with a reasonable sensor footprint?
3. What would an effective integrated infrasound sensor network need to look like to achieve a high coverage?

The first sub-question focusses on the detection of meteor signatures in measurements generated by infrasound sensors. This question aims to set up a potential detection pipeline and methodology using public infrasound data from the Raspberry Shake infrasound network (Raspberry Shake, S.A., 2016). This network is used as the Raspberry Shake infrasound sensors are considered for a future infrasound network, and a public network of these sensors already exist operated by private individuals.

The second sub-question focusses on methodologies for noise reduction. Although there have been multiple studies already into noise reduction methodologies for infrasound sensors (Noble *et al.*, 2014; Walker & Hedlin, 2009), most of the experiments in these studies relied on relatively large filters. For an infrasound network to be incorporated into a network such as DOERAK/FRIPON, the sensor might need to be located in places where the space for such

filtering setups is limited. As such, this question will focus on simple smaller wind noise filtering setups rather than the large systems typically investigated in existing literature.

The third sub-question assumes that the problem of detection and noise reduction has been solved, and looks at how these infrasound sensors can then be incorporated and spread out over the detection network. This involves looking at optimal placement locations for infrasound sensors within this network, and the minimum sensor density required to achieve a high level of coverage within the relevant area.

1.3. Structure

This report is structured into 5 chapters. First, this introduction chapter provides a brief introduction to the goal of this research, and presents the research questions which the report aims to answer. Next, chapter 2 provides insight into the current state of the research into the field of meteoric entries and their detection, to help more clearly define the knowledge gap and discuss any required prerequisite knowledge for the research approach used in this investigation. After this, chapter 3 outlines the methodology used for answering each of the research questions given in section 1.2. For this purpose, this chapter is subdivided into three sections corresponding to each of these questions. Similarly, chapter 4 presents the findings from the investigations into each of these research questions, also being subdivided into separate sections for each question. Lastly, chapter 5 concludes the investigation and provides recommendations for future work.

2

Background

This chapter introduces the background knowledge on the subject of meteor entry and detection, based on a study of the available literature on this subject.

2.1. Meteor entry trajectory

Before understanding the detection of meteors, it is important to first understand the entry trajectories of such meteors as they pass through the earth's atmosphere. The path taken by the meteor during this entry phase is a major contributing factor to the detectability of the meteor entry, since this directly determines the propagation direction of the sound waves generated by this entry. To understand typical entry trajectories of meteors, first it is important to understand the source of these meteors as these can help constrain the speed range of meteor entries.

The vast majority of meteoroids that enter the earths' atmosphere originate from within the solar system. As found by Hajdukova *et al.*, 2020, of the hundred thousand asteroids contained in their data set, approximately 90% had speeds lower than the escape velocity of the Sun, thereby being unlikely to come from an interstellar trajectory. Additionally, for the meteors which had speeds in excess of the escape velocity of the Sun, this does not act as concrete proof that these meteors actually came from interstellar space. Instead, this speed discrepancy can also be caused by effects such as gravitational slingshots around other bodies and outgassing of meteors, launching particles outwards at higher speeds before being intercepted by the earths' atmosphere on their path out of the solar system. As such, Hajdukova *et al.*, 2020 concluded that there is no conclusive proof that any interstellar meteors have been detected at all.

The observation that asteroids generally typically originate from within the solar system provides an upper bound to the entry velocities given by the escape velocity at the earths' orbital altitude plus the earths' own orbital velocity which leads to an upper bound of approximately 72 km s^{-1}. A more important bound on the entry velocity can be derived from the escape velocity of the earth. As meteors originate from orbit of the Sun, their velocity upon reaching the atmosphere must exceed the escape velocity of the earth at that point since the descent

into the earths' gravity well will have imparted this velocity on the meteor. As such, it can be found that all meteors will have an entry speed in excess of 11.2 km s$^{-}1$ (Edwards, 2009).

Whereas this 11.2 km s^{-1} to 72 km s^{-1} limit provides a global bound on meteor entry speeds, it is commonly possible to determine significantly stricter constraints on meteor entry velocities if these meteors originate from known asteroid belts, or if other constraints are put on the detected meteors. For example meteors originating from the Geminids meteor belt have typical velocities of approximately 36.4 km s^{-1}, from the Perseid meteor shower around 60.5 km s^{-1}, and from the Leonids approach the upper speed limit coming in at approximately 71.5 km s^{-1} (Molau & Rendtel, 2009). Alternatively, if you apply the constraint that you want to detect meteors which have a reasonable chance of survival, it was found by Borovička *et al.*, 2019 that meteors typically only survive the atmospheric entry if their entry velocities are below 30 km s^{-1}.

Due to their high entry speeds, meteors typically last less than a second before they have fully burned up or break apart due to aerodynamic forces. As such, the total acceleration of gravity during this entry is multiple orders of magnitude smaller than the speed of the meteor, and can thus be neglected. Therefore, the only relevant force during the entry of the meteor is the atmospheric drag caused. Since drag acts opposite to the direction of motion, the meteors' trajectory will effectively be a straight line. During this straight trajectory, the speed of the meteor is give by Ceplecha *et al.*, 1998 to follow equation 2.1.

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\Gamma A \rho_m^{-2/3} \rho_{atm} m^{-1/3} v^2 \tag{2.1}$$

In this equation, Γ is the drag coefficient, A is the shape factor of the meteor, ρ_m is the meteor density, ρ_{atm} is the atmospheric density, m is the meteor mass, and v is the meteor velocity. Only if a meteor survives this rapid heating phase and turns into a meteorite will the trajectory show significant curvature once atmospheric drag has slowed the meteor down significantly.

2.2. Fireball detection

This section will cover the detection of meteors using camera networks. First, subsection 2.2.1 will cover the setup of the camera's in fireball detection networks. Next, subsection 2.2.2 will cover the analysis of data produced by such camera networks.

2.2.1. Camera setup

An important method of detecting meteor entries is by detecting the fireballs produced during entry into the atmosphere. For this purpose, multiple different camera networks have been set up around the world to monitor such fireballs. These networks include ASGARD network (which is a part of the broader Southern Ontario Meteor Network) (Brown *et al.*, 2010), the FRIPON network (Colas *et al.*, 2020), the GMN network (Vida *et al.*, 2021), the AllSky7 network Hankey *et al.*, 2023, and the EN network (OBERST *et al.*, 1998). Such camera networks consist of multiple of all-sky cameras which continuously take pictures of the entire sky to detect light sources such as those caused by meteors. Additionally, some networks such as the Canadian Automated Meteor Observatory (CAMO) extend this network with a narrow field camera, which

actively tracks identified meteor events in real time thereby giving significantly higher resolution imagery of the meteor and its luminous tail (Subasinghe *et al.*, 2016).

To be able to correlate the data from the many different sensors in the network, the configuration of both the individual sensors and the network as a whole plays an important role. Typical sensors, such as those used in the FRIPON network, make use of optical sensors which detect light in the visible wavelengths. To ensure the sensors achieve comprehensive coverage of the skies, these sensors are generally equipped with fish eye lenses which give the camera the wide field of view required to cover the full sky with a single sensor. Since these cameras have to be mounted outside to achieve a proper view of the sky, the sensor is placed under a transparent dome that protects the sensor from rain, wind, and other weather effects. Additionally, to prevent dew build-up on the dome and sensor, the cameras may be equipped with heaters and air circulation fans.

To ensure the camera does not detect nearby objects such as trees or cars driving by, the cameras need to be provided with a mask. Such a mask is a software-defined region within which the camera is allowed to process any light level variations. Any movement detected outside of this mask is simply discarded. Additionally, to ensure that the light level of a fireball can be correctly quantified for the creation of a light curve and corresponding entry energy, the camera must be calibrated with a light source with known brightness. This process must be done repeatedly, as local phenomena such as thin cloud cover, dirt, or refraction can alter the sensitivity of the sensor over time. Therefore, this calibration is typically carried out using stars of known brightness (Colas *et al.*, 2020). Since these stars are all at known locations, it is possible to correlate the light spots detected by the camera with these stars, thereby allowing the camera to update its sensitivity in those regions of its field of view as they change over time.

2.2.2. Fireball analysis

Analysing the visual signature of a meteors' luminous trail can provide valuable data regarding meteor entries such as the trajectory of the meteor, the entry velocity, and the light curve emitted by the meteor. The light curve contains information regarding the brightness of the fireball as it passes through the atmosphere as a function of time (and therefore also the distance the meteor has travelled in the atmosphere).

By analysing properties such as the duration of the light emitting phase, the peak brightness of the meteor entry, and the general shape of the light curve, it is possible to determine physical properties of the meteor. The light emitted by the meteor is caused by the meteor heating up as it collides with the air molecules at such high speeds. This heating causes the meteor to radiate a part of this energy away in the form of light. As given by Brykina and Tirskiy, 2017, the luminosity of the fireball is related to the entry velocity and mass loss rate of the meteor following equation 2.2.

$$I = -\tau \frac{\mathrm{d}}{dt}\left(\frac{mv^2}{2}\right) \tag{2.2}$$

In this equation, τ is the radiation efficiency, m is the meteor mass, and v is the velocity.

An interesting observation from this equation is that the $mv^2/2$ term constitutes the energy of the meteor. As such, the luminosity of the meteor is directly related to the change in energy of the meteor. Another observation from this is that if the speed is known or can be reasonably estimated over a detected light curve, it is possible to provide a rough estimate of the mass of the meteor. A complicating factor for this however is the lack of information about the luminous efficiency of meteors. Although multiple studies have been done to find the luminous efficiency of meteors, these have often given different values dependent on the meteors included in the data set. Ceplecha, 1996 has found a luminous efficiency at 13 km s$^-1$ of 6.1%, dropping to 1.2% when the meteor has slowed down to 4 km s$^{-1]}$. A more recent study by Subasinghe and Campbell-Brown, 2018 on the other hand found a luminous efficiency of <1% independent of the entry speed, with a positive trend developing for smaller meteors. A different study by Loehle *et al.*, 2024 conducted using a high speed wind tunnel similarly found the luminous efficiency of meteors to lie between 0.01% and 1%, depending primarily on the composition of the material. In this study, it was found that an important indication for the luminous efficiency is the iron contents of the meteor, with higher iron contents typically yielding higher luminous efficiencies. This mechanism primarily responsible for this phenomenon was determined to be the excitation of the electrons around the iron atoms themselves when colliding with nitrogen atoms. This effect causes increased emissions particularly in the related iron spectral band around 500 nm. Identifying this with an all-sky meteor detection camera could however prove more complicated, as this would require the determination of not just the light curve, but a spectral analysis of the detected fireball.

Another complicating factor in using the light curve to determine the photometric mass of a meteor occurs when a meteor fragments into multiple smaller pieces. During such a fragmentation event, the shape and size of the meteor chunks rapidly changes, further complicating the light signature emitted by the meteor.

2.3. Shock wave formation

When a meteor collides with air molecules in the atmosphere, besides causing significant heating to the meteor itself, this interaction also causes a shock wave in the surrounding atmosphere. The angle at which the shock is formed is related to the speed of the source through equation 2.3.

$$\sin(\alpha) = \frac{1}{M} = \frac{C_s}{v} \tag{2.3}$$

In this equation, α is the shock angle (the angle between the centreline of the cone and the cone itself), C_s is the speed of sound in the surrounding atmosphere, and v is again the meteor velocity. As stated in section 2.1, meteor entry velocities have a lower bound at 11.2 km s^{-1}. In the earths' atmosphere, the speed of sound typically ranges between $\sim$290 to $\sim$350 m s^{-1}. As such, it can be found that the Mach number of meteors at the point where they start to hit the atmosphere range from Mach 35 up to 240. At such speeds, the shock angle can be found to be typically <1°. Therefore, the shock cone produced by the meteor can be assumed to be effectively equal to a cylindrical shock as would be produced by an instantaneous line source.

Whereas the cylindrical shock wave propagates radially outwards with respect to the meteor

trajectory, this cylinder is capped on the front by a more spherical shock ”cap” (Edwards, 2009). Despite the seemingly simple distinction between these two wavefront geometries, Edwards *et al.*, 2008 found there to be no hard boundary between where each of these two geometries was dominant. Instead, it was found that for rays up to 20° from the lateral direction were primarily dominated by the ballistic shock produced by the cylindrical shock, with the range between 20° and 35° showing no clear distinction. Only after this 35° deviation from the normal could it be assumed that the shock cap is the dominant shock front. This uncertainty is partially caused by the high temperatures near the meteor causing the shock to locally have deviations of as much as 25° from the perpendicular direction.

More complex shock patterns are found when the meteor breaks apart due to aerodynamic forces. At the initial fragmentation point, Edwards, 2009 states that the meteor can additionally act like a point source. Additionally, after breaking apart Silber and Brown, 2014 found that the multiple fragments now each causing shock waves could lead to multiple shock waves travelling in close succession. Properly predicting these shock patterns is therefore increasingly difficult.

2.4. Infrasound detection

When the shock waves produced by the meteor travel away from the meteor, they will slowly decay into regular sound waves. Initially, these shock waves will have a pressure waveform closely resembling the capital letter N, consisting of the initial high pressure bow shock, followed closely by a low pressure zone behind which the air settles back to ambient conditions. Due to this similarity, these shockwaves are commonly referred to as N-waves (Beyer, 1974). As the high frequencies contained within the initial shockwave suffer from a stronger attenuation than the lower frequencies, the resulting sound waves primarily remain at frequencies below 20 Hz, commonly referred to as infrasound signals due to falling outside of the human range of hearing. This terminology mirrors that for infrared light, which equally has a longer wavelength than visible light. Although the exact upper boundary of the infrasound regime remains somewhat fuzzy due to the uncertainty regarding the limits of human hearing, the 20 Hz boundary has been generally accepted as the start of the infrasound regime (Leventhall, 2007). Another important distinction to make is that the sound covered in this investigation is in particular the sound resulting from the shockwave produced from the meteor passing through the atmosphere. Although meteors can also produce audible sound through other mechanisms, such as photoacoustic effects where localised heating due to radiative effects can cause the formation of sound pressure waves (Spalding *et al.*, 2017), these effects are not investigated in this thesis due to their wildly different formation mechanisms. Additionally, with the focus of this investigation being on improving the detection of meteors, this effect is likely to be of lesser interest as it does not meaningfully improve the detection capabilities in scenarios where existing camera networks fall short such as during daytime when the effects of sunlight will be significantly greater than of the meteors luminosity.

Although the infrasound generated by the meteor shockwaves is inaudible to the human ear, it is possible to detect these signals with the use of sensitive microphones. For the infrasound

to reach these microphones, there are two important paths the sound waves can take from the source. For short distances, the sound waves can travel directly from the source to the observer in a direct line. This direct propagation will be covered further in subsection 2.4.2. For longer distances, a direct line between the source and the observer is no longer possible. Instead, the sound waves reach the observer by refracting and reflecting of different atmospheric layers. This propagation will be covered in subsection 2.4.1.

2.4.1. Long range propagation

When a meteor produces a shock wave, the sound propagates outwards in many directions, including directions which would not directly lead to an observer on the surface of the earth. This does however not mean that these sound waves cannot be detected on the surface. Due to the different temperatures within different layers of the earths' atmosphere it is possible for the sound waves to be curved or even be reflected completely at the interfaces between such different layers. This phenomenon leads to the possibility that, if the atmospheric layers are stacked in certain ways, this can create a "duct" through which the sound can propagate over very long distances. This long distance propagation is facilitated by the fact that the attenuation of the sound energy, as stated by Ens *et al.*, 2012, is proportional to the square of the frequency. As found by Drob *et al.*, 2003, this can result in an attenuation of just 0.01 dB km^{-1}. Therefore, as the frequency of infrasound is very low, meteor generated infrasound can travel significant distances before fading into the background noise. In the case of a meteor which entered above the Atlantic ocean in February 2016, this resulted in an infrasonic detection at a distance of about 5000 km from the fireball location (ElGabry *et al.*, 2017). In a more general case study, Ens *et al.*, 2012 found that detections could even be achieved at distances up to 17000 km.

As found by Drob *et al.*, 2003, there are three main ducting layers in which infrasound can travel. The first duct can be found in the troposphere, which for low level sources was found to contain up to 20% of the total infrasound energy. This duct was found to primarily form for sources that at higher latitudes. An important factor in the effectiveness of this specific duct was found to be the presence of jet streams, which can help steer the sound waves along the duct. The second duct was found to exist in the stratosphere. Similar to the tropospheric ducting, this duct is most clearly seen at higher latitudes, where it can contain up to 40% of the total wave energy. Once again, an important contributing factor to this ducting is the presence of jet streams pushing the sound waves along the ducting path. An important distinction between these two ducting types is the primary direction along which the duct exists. Whereas the tropospheric duct primarily exists in the north-south direction, the stratospheric duct is more pronounced in the east-west direction due to the predominant wind direction in the jet streams at each of these altitudes. The third duct was found to exist in the thermospheric layer. Unlike the previous two ducts, this thermospheric duct is more pronounced closer to the equator. This ducting does not rely on jet streams for its formation, instead being more driven by the temperature gradients in the atmosphere. Due to this, this ducting does not show a significant directional preference, radiating roughly equally in all directions. A consequence of this omnidirectional ducting is that the signal will suffer from higher attenuation due to the

geometrical spreading of the sound waves over a larger area. Depending on the latitude, this duct may contain anywhere between 40% of the total infrasound energy (at higher latitudes) up to 85% near the equator.

This long range propagation of infrasound has been used in multiple studies to detect meteor entries at long ranges, which is especially important if the meteor enters at a location where a denser detection network is not feasible. One such study was performed by Ens *et al.*, 2012. For this study, infrasound measurements were generated using the International Monitoring System (IMS), a global network of infrasound sensors set up to monitor compliance to the Comprehensive Nuclear Test Ban Treaty Organisation (CTBTO). Using the measurements of this system, Ens *et al.*, 2012 was able to set up a relation between the detected infrasound frequency as given by equation 2.4.

$$\log(E) = a \log(\tau) + b \tag{2.4}$$

In this equation, E is the total energy of the meteor in kT of TNT equivalent, τ is the measured dominant period of the signal measured in seconds, and a and b are regression constants found to be 3.75 ± 0.19 and 0.50 ± 0.29 respectively.

A different study by ElGabry *et al.*, 2017 used detections of multiple long range infrasound stations to triangulate the entry point of the meteor. In this study, it was found that the position of the meteor entry could be accurately determined using this method by comparing the results with visual observations of the meteor.

Although these studies show that it is feasible to infer properties of a meteor using long range infrasound, an important downside of such long range infrasound detections is the limitations in detectable meteor sizes. Due to the background noise and attenuation of the signal, it was only possible to detect larger meteors using such techniques, as small meteors would not generate large enough infrasound signatures to be detectable at such long distances. For distances up to 5000 km, Ens *et al.*, 2012 found that the entry events needed to have an energy equivalent of at least 1 kT of TNT to be detectable. For a meteor travelling at the upper speed limit of 72 km s^{-1} defined in section 2.1, this would require a minimum meteor mass of 1.6 kg, whereas for a slower meteor such as those arriving during the Geminids meteor shower, a minimum mass of 6.3 kg would be required. Given that the FRIPON camera network can and regularly does detect fireballs for meteors as small as $\sim$1 cm (Colas *et al.*, 2020), that puts the mass of these meteors within the range of just a few grams, well below the long range detection limit for meteor infrasound signatures.

2.4.2. Short range propagation

At shorter propagation distances, the ducting covered in subsection 2.4.1 does no longer significantly affect the infrasound signal, and instead a more direct propagation path can be assumed. Measuring infrasounds closer to the source provides two large benefits over long range detections. Firstly, due to the shorter distance, the signal will suffer significantly less attenuation allowing for smaller meteors to be detected using this approach. Secondly, since the sound waves travel a shorter distance through the atmosphere, the signal will not be

distorted as much by the ambient conditions, allowing for higher resolution estimation of the meteor properties than is possible at longer ranges. The range until which it can be assumed that the sound travelled from the source to the observer in following a direct path (i.e. no reflections) is typically given at a propagation path length of approximately 300 km (Ens *et al.*, 2012, Silber *et al.*, 2015), although Edwards, 2009 uses a slightly stricter definition of regional infrasound of just 200 km of ground range.

For infrasound detection at such reduced ranges, it becomes more feasible to model the distortions of the sound waves over the trajectory, allowing better estimations of the meteor properties such as mass, diameter and velocity. For doing so, three main approaches have been explored. The first approach, as laid out by ReVelle, 1976 and later refined by Edwards, 2009 analytically calculates the distortion of the wave using a simplified model of the atmosphere. This model divides the infrasound trajectory into three main phases. The first propagation phase occurs close to the meteor. In this phase, the shock is strongly non-linear, which makes it hard to accurately model the behaviour of the wave. As such, this part of the trajectory is not modelled in detail. Instead, the outer boundary of this section is taken as the start of the propagation. To determine where this strongly non-linear region ends, the blast radius of the meteor is defined following equation 2.5.

$$R_0 = \left(\frac{E_0}{p}\right)^{1/2} \tag{2.5}$$

In this equation, R_0 is the blast radius, E_0 is the energy deposited into the air per unit length along the meteors' trajectory through the atmosphere, and p is the ambient atmospheric pressure at the relevant altitude. Since the energy deposited by the meteors is directly related to the atmospheric drag, this equation can often be rewritten as equation 2.6, with M being the Mach number of the meteor, and d_m being the diameter of the meteor.

$$R_0 \approx M d_m \tag{2.6}$$

To extent of the first highly non-linear phase is then simply defined as being equal to $10R_0$, after which the shock transitions to a more weakly non-linear shock. At the start of this second phase, it was found by ReVelle, 1976 that the fundamental frequency of the produced shock wave is related to this blast radius following equation 2.7.

$$f_0 = \frac{C_s}{2.81 R_0} \tag{2.7}$$

From these equations, it can be seen that more energetic meteor entries, either due to being larger or heavier meteorites, or by having higher entry velocities will lead to a larger blast radius R_0, and thus a lower fundamental frequency f_0. During this weakly non-linear shock phase, the wavelength of the produced shock wave will slowly become longer as the shock wave loses some of its energy due to attenuation and due to the higher air densities it encounters as it reaches lower into the atmosphere. Additionally, the non-linear peaks present in the N-wave produced by the meteor will smoothen out into a more classical wave-like shape. These effects are algebraically propagated as explained in more detail in Edwards, 2009.

During the final propagation phase, the wave is assumed to have transitioned into a linear sound wave without any significant shock-like properties anymore. Once the wave has transitioned to this phase, its speed and therefore its frequency and wavelength remain effectively constant, with the only remaining attenuation causing a lower amplitude. The exact point at which this transition occurs however is quite difficult to determine. ReVelle, 1976 proposes that this transition point should be placed at a distance from the source such that the total change in fundamental period of the wave is less than 10%. This distance is given by equation 2.8.

$$d' = \frac{C_s \tau}{34.2 \frac{\Delta p}{p}} \tag{2.8}$$

In this equation, d' is the "distortion distance" (i.e. the distance from the observer at which the transition occurs), C_s is the local speed of sound, τ is the fundamental period of the wave at transition, and $\frac{\Delta p}{p}$ is the overpressure ratio of the wave. Silber *et al.*, 2015 on the other hand suggests, based on a study where the theory laid out by ReVelle, 1976 was applied to a dataset of meteor entries, that the distortion distance should be taken significantly smaller than this value, which would limit the distortion to just a few percentage.

An alternative model has been proposed by Haynes and Millet, 2013, which makes use of the Whithams F-functions to determine the sound wave overpressure ratio at a receiver. This equation is given in equation 2.9, where p'/p is the overpressure ratio, K is a constant form factor, and R_0 is the blast radius. Furthermore R is the distance between the meteor and the sensor, H is a constant height scaling factor defined as $a_0^2/(\gamma g)$, and Z is the meteor height. In this equation for the height factor, a_0 is the speed of sound at the blast radius R_0, γ is the specific heat ratio for air set at 1.4, and g is the gravitational constant.

$$\frac{p'}{p} = 2\frac{K}{\bar{\Phi}}\sqrt{\frac{R_0}{R}}\,\mathrm{exp}\left(\frac{R_0 - R}{2HR}Z\right) \tag{2.9}$$

The term $\bar{\Phi}$ is calculated using equation 2.10.

$$\bar{\Phi} = \frac{\gamma + 1}{\gamma a_0}\int_0^R \sqrt{\frac{R_0}{r}}\,\mathrm{exp}\left(\frac{R_0 - r}{2HR}Z\right)\mathrm{d}r \tag{2.10}$$

This equation has however not yet been as extensively tested against known meteor entries. Additionally, this method has not been covered as much in literature as the method proposed by ReVelle, 1976. As such, this model will not be investigated in further detail in this report.

The third method for determining the propagation of sound waves is through the use of numerical methods. For this purpose, a tool called SUPRACENTER was developed by Edwards and Hildebrand, 2004. This tool takes in a model of the atmosphere, which is then used to numerically simulate many different potential ray paths between the source and the observer. Although this program is primarily focussed on generating accurate arrival times, the parameters given by these trajectories can be used to further constrain the distortion of the infrasound signal. Notably, this program was used by Silber and Brown, 2014 to get better

determinations of the angle from the shock towards the observer to study the ray deviation at which the infrasound leaves the ballistic regime and becomes a spherical shock. The trajectories generated by SUPRACENTER allowed for better interpretation of the gathered infrasound signatures by being able to better constrain the source location of the infrasound signature along the meteor trail.

2.4.3. Infrasound measurement

Due to the long wavelengths of infrasound signals, conventional microphones are generally not well suited for properly measuring these signals. As found by Zhang *et al.*, 2020, the lower frequency limit detectable using modern conventional microphones is around 2 Hz. Since meteor generated infrasound can reach well below this limit, going down to < 1 Hz especially for larger meteors, alternative sensors have to be used instead. As such, many infrasound sensors such as the ones in use at the IMS operate using microbarometers (Christie & Campus, 2009). Microbarometers are highly sensitive barometers which measure the pressure differential between their sensor port and a reference pressure which allow a significantly higher accuracy compared to sensors which measure the absolute air pressure (Grangeon & Lesage, 2019). As the ambient pressure can still change due to other sources such as the weather and ambient temperature, the reference pressure typically consists of a secondary port equipped with a low pass filter allowing this pressure to slowly adjust to the ambient air pressure.

For such microbarometers, there are multiple sources of noise. As with any sensor, the microbarometer will have some level of quantisation noise, and can be sensitive to noisy power supplies affecting the sensor performance too. Additionally, other sources of infrasound than the one of interest, such as microbaroms caused by oceanic waves create ambient noise floors which cannot effectively be filtered. The main cause of noise in infrasonic measurements however is the pressure variations caused by the wind interacting with the surroundings of the sensor, causing additional pressure variations at the sensors' input port(Bowman *et al.*, 2005). The amplitude of this noise can commonly exceed the strength of the signal received at the infrasound sensor, especially when attempting to detect low energy events such as smaller meteors entering the earth's atmosphere. As such, for performing infrasonic measurements, multiple methods have been devised to reduce the effect of wind noise on the sensor.

The first method that is commonly used to reduce the wind noise is to attach one or more porous hoses to the infrasound sensor. Since the signal that the sensor is trying to measure arrives at the sensor as a coherent wave, the effect of the signal on the individual pores will be similar thus passing through the hose without much attenuation. Wind noise on the other hand is significantly less coherent, even over relatively small length scales (Walker & Hedlin, 2009). As such when the effect of this noise is averaged over the pores, the noise is expected to mostly cancel out, thereby allowing only the signal to pass through to the actual sensor.
A downside that is often noted for such porous hose setups, is the sensitivity of such systems to introduce additional artifacts into the received signal. Due to the shape of the hose, it is possible for specific wavelengths to resonate in the hose and form standing waves which alters the

response of the system to signals in these frequencies. The frequencies at which this occurs depends on the geometry of the hose, where for longer hoses the resonant frequency will shift downwards, as this allows longer wavelength standing waves to persist. As such, determining the length of the porous hose requires a trade-off regarding the filtering capabilities, which improve for longer hoses due to covering a larger area, and signal artifacts due to resonance which get worse for longer hoses.

Another common approach for reducing wind noise in infrasonic measurements is through the use of a porous dome placed over the infrasound sensor. Such domes are commonly made either from porous fabric (Noble *et al.*, 2014), or a metal mesh. The filtering mechanism behind these domes is similar to the filtering of the porous hose. Due to the large amount of pores in such a dome, incoherent signals such as those generated by wind noise will have a net zero pressure contribution when averaged out over the pores. Therefore, these noise signals will be unable to penetrate the dome and reach the sensor port. Coherent signals on the other hand will, due to the long wavelengths of infrasound signals, have an approximately equal effect on each of the pores, therefore not cancelling out and thus being detected at the sensor inside the dome. As found by Noble *et al.*, 2014, for the effectiveness of fabric domes, the porosity of the material can play an important role. If the material has too high porosity, the turbulences can pass through the dome too easily and the averaging effect of the dome is thereby greatly reduced. If on the other hand the porosity is too low, the dome might start to behave closer to a fully sealed enclosure, and will also attenuate any potential coherent signals significantly, making it harder to detect them. This research however didn't cover the optimal porosity, but rather compared different available fabric types, thereby leaving this question of optimal porosity unanswered.

3

Methodology

As stated in chapter 1, the main research objective is to determine how to improve meteor detection capabilities by using infrasound. To answer this overarching question, three research aspect will be conducted as part of this investigation, to each answer one of the sub-questions. First, section 3.1 will aim to provide better insights into sub-question 1 how meteors can be detected in infrasound measurements. Next, section 3.2 explains how the individual sensors should be set up to provide the meteor detection capabilities in accordance with sub-question 2. Lastly, section 3.3 will investigate how the layout of a possible infrasound detection network would affect its capabilities as per sub-question 3. The code used for the various analysis done during this investigation can be found in appendix C.

3.1. Infrasound detection

The first part of the investigation focusses on the detection of meteoric infrasound signals using existing public infrasound networks. First, section 3.1.1 will describe the data sources and their retrieval process. Next, section 3.1.2 describes the preprocessing that need to be done on the infrasound data before it can be analysed. Subsequently, section 3.1.3 explains the calculations for determining the arrival window for the infrasound signature. Finally, section 3.1.4 describes the detection algorithm used for detecting infrasound signatures. An overview of the full pipeline from meteor entry to detection can be seen in figure 3.1.

3.1.1. Data retrieval

To detect the presence or absence of meteoric infrasound signatures, two types of data are required. First of all, one needs infrasound measurements performed during the period of interest. For this purpose, the Raspberry Shake network is used (Raspberry Shake, S.A., 2016). This network is a community driven network of seismological and infrasound sensors. The network consists of more than 2500 sensors worldwide, of which more than 250 are equipped with infrasound sensors. Of these infrasound sensors, approximately 80 of these sensors are located throughout Europe.

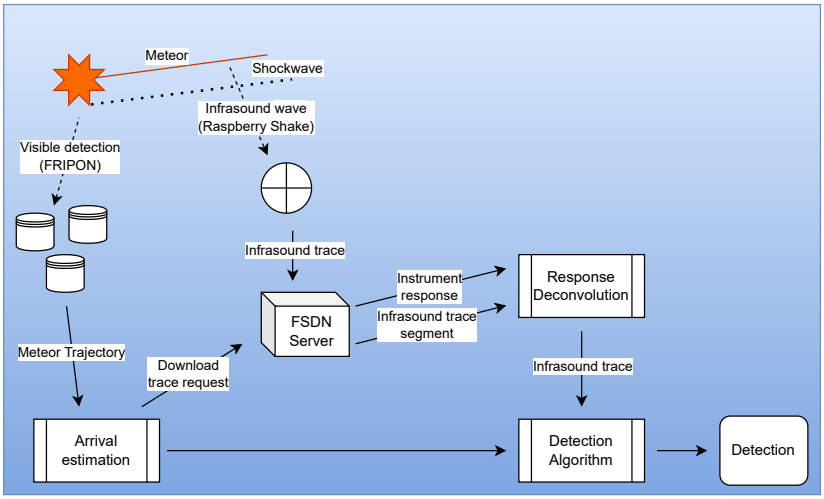


Figure 3.1: The detection pipeline for detecting meteor infrasound signatures

The infrasound data generated by the sensors in the Raspberry Shake network can be retrieved from the FDSN web service. To retrieve the data, a get request can be sent to the FDSN web servers formatted with the specification of the desired data. The relevant information to be included in this request is the network identifier (AM for the Raspberry Shake network), the station identifier, and the start and end times for the data. This way, data can be downloaded for single stations in chunks up to 24 hours in length. To limit the amount of data that has to be downloaded and processed, during this research a more targeted approach is used. First the relevant stations and arrival time of the infrasound wave are determined using the procedure explained in section 3.1.3. From this information, the data for all relevant data is downloaded with a rate limit of 30 requests per minute as dictated by the FDSN web server.

The second data type required is a database of meteor entries which could potentially be detected. For this purpose, the FRIPON fireball detection network is used (Colas *et al.*, 2020). This network consists of 372 all-sky cameras which continually monitor the sky for fireballs which indicate meteoric entries. The FRIPON consortium has a yearly data release, containing timestamps for detections of fireballs, the trajectory of these fireballs, and properties of the meteor to the extent of how far they could be determined based on the camera observations. This dataset is made freely available for non commercial purposes and can be retrieved in various formats from https://fireball.fripon.org/list_pipeline.php. This dataset currently contains every meteor entry logged by the FRIPON network during the timeframe from the beginning of January 2016 up through the end of December 2022. Due to the locations of the FRIPON camera stations, the majority of these detected fireballs are located above France, Belgium, The Netherlands, and the Northern part of Italy. Additionally, the network has some cameras located in Canada surrounding Montreal and throughout Romania, although these made up a fora comparatively small amount of the total network size, leading to these areas not significantly contributing to the overall number of detections.

3.1.2. Data preprocessing

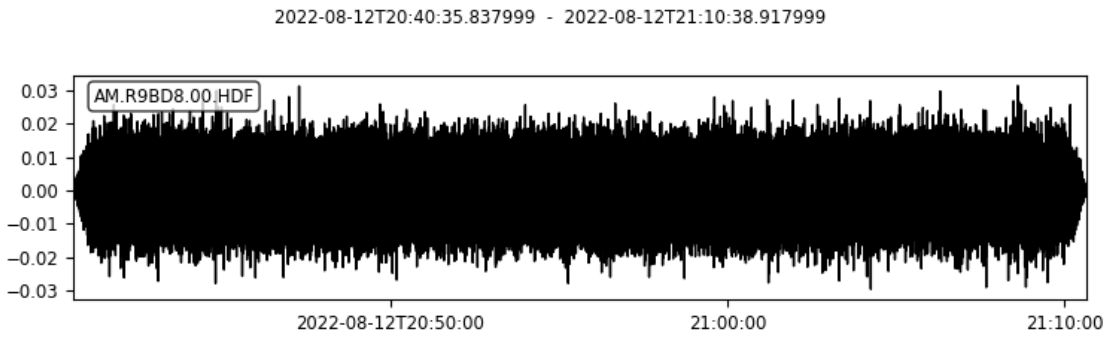
Before the infrasound measurements from the Raspberry Shake network can be analysed, they first need to be converted from their base unit of counts to atmospheric pressure values. This preprocessing of the data can be done using the obspy python library (Beyreuther *et al.*, 2010). To do this, first the instrument response of the sensor has to be downloaded from the FDSN web servers using a similar process used to download the data as explained in section 3.1.1. Using this data it is then possible to remove the response of the instrument, giving back pressure measurements in Pa again.

Due to the nature of the deconvolution process of removing the instrument response, it is necessary to apply a filter to the data beforehand to prevent the results from being overly amplified or suppressed due to the poles present in the instrument response. That is, without applying a filter, at frequencies where the sensor has a very low sensitivity removing the instrument response could lead to over amplification of the signal or noise in these bands which could hide signals present in other parts of the spectrum or hide signals in the time domain. For this purpose, the obspy library provides two filtering methods which can be used independently or in conjunction with one another.

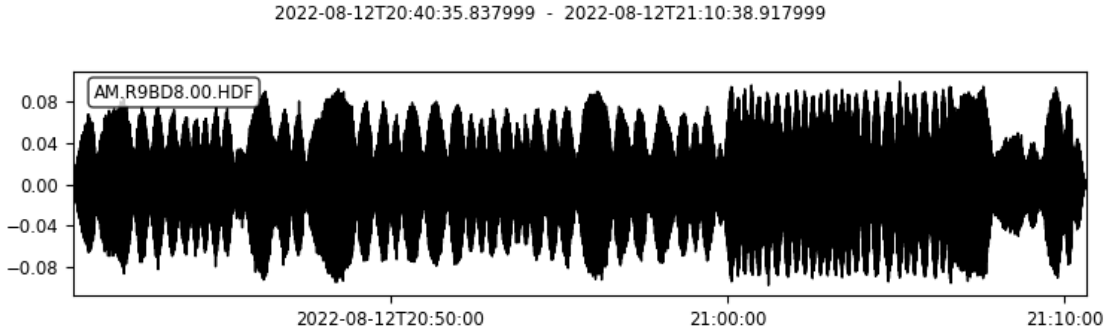
The first method available is through the use of what obspy calls a ”water level” filter. This water level filter assumes a minimum instrument sensitivity, which prevents the inverted instrument response (which corresponds to $1/$sensitivity) from having excessively high peaks at certain frequencies, since the sensitivity can never reach zero. This in turn ensures that the actual data does not get multiplied by excessively high values which would otherwise have caused the measurements to take on unrealistic values. This method does however come with the downside of suppressing potentially desired frequencies if the water level is set too high as to where it starts clipping in the relevant frequency bands. As such, choosing a an appropriate water level is an important aspect of using this filter. Unfortunately, finding this water level is a difficult process which is equally difficult to validate without having access to reliable verification data.

The second method available for this preprocessing is through the use of a frequency pre-filter. This method instead applies a cosine tapered window in the frequency domain before the deconvolution. For this window, the user can define the four corner frequencies where the window begins to rise, and the flat section of the window where it does not affect the signal. In this approach, the user is responsible for defining a window where the sensor has a good sensitivity and where the deconvolution thus will not cause any excessive multiplications. Since this filter is guaranteed not to negatively affect the signal within the flat section of the window and is therefore significantly easier to dial in on a correct value, this method is used over the water level filter. The corner frequencies for this filter were generally set to 0.5 and 1 Hz for the lower bound, and 45 and 50 Hz for the upper bound. This lower bound is high enough as to where the sensitivity of the instrument is still high enough as to not cause any notable artifacts in the important frequencies, while being low enough as to not reduce the measurable range of the instrument (which extends down to a full sensitivity down to approximately 1 Hz for most Raspberry Shake sensors). The upper limit is set to reach zero at the theoretical

maximum frequency measurable according to the Nyquist limit for the sampling rate of the sensor. The Nyquist limit states that the maximum frequency that can be distinguished in a measured waveform is equal to half the sampling frequency, such that each period contains at least two sampling points (McLean *et al.*, 2005). For the Raspberry Shake sensor used in this investigation, which has a sampling frequency of 100 Hz, this gives a maximum measurable signal frequency of 50 Hz. If the upper limit of the pre-filter is raised beyond the sampling limit of the sensor, the deconvoluted result will show significant artifacting in the final waveform, potentially suppressing relevant values. This artifacting can be seen in figure 3.2, where subfigure 3.2a shows an infrasound measurement with the pre-filt upper limit set to 45/50 Hz, whereas subfigure 3.2b shows the same trace with the upper limit set to 50/55 Hz



(a) A deconvoluted infrasound trace with a pre-filter upper bound of 45/50 Hz



(b) A deconvoluted infrasound trace with a pre-filter upper bound of 50/55 Hz

Figure 3.2: Comparison of an example infrasound measurement trace deconvoluted with different pre-filter settings

3.1.3. Arrival calculation

To correlate fireballs with their infrasound signature measured by an infrasound station, there are three parameters that need to be calculated. The first parameter is the distance between the fireball and the infrasound station. As found in section 2.4.2, the typical range at which direct propagation trajectories can be assumed is for a distance up to 300 km. As such for an infrasound trace to potentially contain the infrasound signature of the fireball, the station must lie within this distance. To calculate the distance, first the trajectory of the fireball is retrieved from the FRIPON fireball database described in section 3.1.1. The trajectory of the fireball

in this dataset is given as two points in the WGS84 coordinate system, each point consisting of a latitude, longitude and altitude value. These points define the points where the fireball initially became visible, and the point where the luminous phase of the meteor ended. To simplify future calculations, these points are converted to the EPSG:4978 coordinate system, a geocentric Cartesian coordinate system, using the Pyproj python library. This coordinate system transformation allows for the application of basic linear algebra principles such as the dot product and certain trigonometric identities to more easily calculate distances and angles between points. Additionally, due to the absence of any curvature in a Cartesian coordinate system, the approximately linear path of a fireball can be more easily described than in the spherical coordinate system. Due to the high speed of the meteor entry, as discussed in section 2.1, the full trajectory of the meteor is simply taken as a straight line connecting these two points.

Due to the nature of infrasound propagation, being effectively perpendicular to the trajectory in the cylindrical shock regime and spherical in the shock cap region, the distance which the infrasound has to travel to reach the sensor can be approximated as the distance between the station and the closest point along the trajectory. As such, the nearest point on the trajectory is found using equation 3.1.

$$\vec{p}_{nearest} = \vec{p}_{start} + (\vec{p}_{end} - \vec{p}_{start}) \cdot clip01\left((\vec{p}_{station} - \vec{p}_{start}) \cdot \left(\frac{\vec{p}_{end} - \vec{p}_{start}}{|\vec{p}_{end} - \vec{p}_{start}|^2}\right)\right) \tag{3.1}$$

In this equation, $\vec{p}_{start}$ and $\vec{p}_{end}$ are the starting and final locations of the luminous trajectory in the cartesian coordinate system, $\vec{p}_{station}$ is the station location, and the $clip01$ function limits its outputs to the range [0, 1] as defined as equation 3.2.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x \leq 1 \\ 1 & 1 < x \end{cases} \tag{3.2}$$

Using this nearest point along the meteor trajectory, the distance between the trajectory and the station can now simply be found using equation 3.3.

$$D_{station} = |\vec{p}_{station} - \vec{p}_{nearest}| \tag{3.3}$$

The second important parameter to calculate, is the angle of the infrasound propagation with respect to the meteors trajectory. Using the nearest point calculated using equation 3.1, this angle can then be found using equation 3.4.

$$a_{propagation} = \arccos\left(\frac{\vec{p}_{station} - \vec{p}_{nearest}}{|\vec{p}_{station} - \vec{p}_{nearest}|} \cdot \frac{\vec{p}_{end} - \vec{p}_{start}}{|\vec{p}_{end} - \vec{p}_{start}|}\right) \tag{3.4}$$

The last parameter to be determined is the estimated arrival time of the infrasound wave. For this calculation, the infrasound wavefront is assumed to travel between the nearest point along the visible line to the station following a straight path at the local speed of sound. To determine this local speed of sound the environmental conditions are estimated using the extended

International Standard Atmosphere (ISA) as defined by (National Oceanic and Atmosperic Administration *et al.*, 1976). This model provides standardised atmospheric properties for the entire altitude range up to infinity. For the estimation of the arrival time, an upper limit of 120 km above sea level is employed. Up to this altitude, the temperature profile can be described fully through linear gradients and isothermal layers, whereas for altitudes above this point the temperature profile is assumed to asymptotically approach 1000 K as the altitude approaches infinity. The properties for each of these layers is specified in table 3.1. For this upper region, no clear lapse rate is defined, making it hard to estimate the environmental properties such as the speed of sound for this region. Additionally in this uppermost region, the composition of the atmosphere begins to differ significantly from the atmospheric composition at lower altitudes, further complicating the calculation of the local speed of sound. Lastly, due to the low density of the atmosphere in these upper regions, it is unlikely that a sound- or shockwave could form and propagate in this region with any reasonable amplitude, making this region insignificant for the detection of meteor generated infrasound. As such, a cut-off altitude for the time estimation calculations is made at 120 km of geometric altitude.

Table 3.1: Layer temperatures for the extended ISA as defined by (National Oceanic and Atmosperic Administration *et al.*, 1976)

Layer base height [km]	Layer top height [km]	Base temperature [K]	Upper temperature [K]
0.00	11.02	288.15	216.65
11.02	20.06	216.65	216.65
20.06	32.16	216.65	228.65
32.16	47.35	228.65	270.65
47.35	51.41	270.65	270.65
51.41	71.80	270.65	214.65
71.80	86.00	214.65	186.8673
86.00	91.00	186.8673	186.8673
91.00	110.00	186.8673	240.0
110.00	120.00	240.0	360.0

To calculate the final arrival time, the time is calculated which it would take for a sound wave to propagate through each of the layers if it is travelling at the speed of sound. Since the waves will travel through these individual layers at an angle, since the meteor is unlikely to pass directly overhead of the relevant infrasound sensors with zero inclination, the vertical velocity of the sound wave is given by equation 3.5

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \cos(\alpha)\sqrt{\gamma R T} \tag{3.5}$$

In this equation, $\mathrm{d}y/\mathrm{d}t$ is the vertical velocity, α is the angle of the sound propagation direction relative to the earth surface normal pointing downwards, and γ is the specific heat ratio of the gas taken as 1.4 for air. R is the specific gas constant of air, taken equal to 287.05 J kg^{-1} K^{-1} for the entire domain, and T is the local temperature. As such, for isothermal layers the total

time it takes to traverse the layer can be found using equation 3.6.

$$\Delta t = \frac{\Delta h}{\cos(\alpha)\sqrt{\gamma R T}} \tag{3.6}$$

In this equation, Δt is the total time it takes the sound wave to traverse the layer, and Δh is the height difference between the start and end of the traversal through this layer.

For layers with a linear temperature gradient, the layer traversal time can instead be derived to be calculated using equation 3.7

$$\Delta t = \frac{2\left(\sqrt{T_{base} + \lambda \Delta h} - \sqrt{T_{base}}\right)}{\cos(\alpha)\sqrt{\gamma R}} \tag{3.7}$$

In this equation, T_{base} is the base temperature of the layer, and λ is the lapse rate of the layer, which is defined by equation 3.8.

$$\lambda = \frac{T_{top} - T_{base}}{h_{top} - h_{base}} \tag{3.8}$$

Using these equations, the total time it takes for the sound wave to reach the observer can then simply be calculated using equation 3.9.

$$\Delta t_{total} = \sum_{i=Layer_{min}}^{i=Layer_{max}} \Delta t_i \tag{3.9}$$

Although this approach can give a decent estimate of the final arrival time, this approach does have some limitations. Due to the use of the ISA, this approach does not take into account local conditions which might affect the speed of sound, such as a different temperature in any of the layers. As such, to account for the potential of an environmental temperature difference relative to those defined in the ISA a common approach is to offset the temperature curve given in table 3.1 such that the temperature of this offset curve match those measured on the surface. Since the exact environmental temperatures at the sensors used during this analysis is unknown due to it not being measured by the raspberry shake infrasound sensors, this approach is limited to assuming a temperature range within which the temperatures are likely to lie. Given the location of the FRIPON network being primarily focussed around France, and the relevant infrasound data therefore being focused on this same region, a temperature range of $T_{min} = 268.15$ K (-5° C) to $T_{max} = 308.15$ K (35° C) is used as a typical temperature range. Using these two temperature values, a likely arrival window is calculated within which any potential infrasound signature is likely to arrive.

Another limitation of this approach is that this approach assumes that the infrasound signature travels a perfectly straight path at exactly the local speed of sound. In reality, the path of the infrasound will likely not be a perfectly straight path, as the sound wave will refract during the temperature gradients causing it to take a slightly faster propagation path. Additionally while the signature is still a shockwave, it will propagate slightly faster than a linear sound wave would in the same medium, once more causing it to potentially arrive earlier than would be anticipated using this arrival window. To account for these unknowns, an additional buffer is added to the final arrival window of 15 s at the start of this window. Additionally, a similar buffer

of 15 s is applied at the tail of the window to account for the fact that the infrasound signature might not be just a single peak, but could attenuate into more of a multi-wave train, as found by (Edwards, 2009). This final window is then used to determine if any potential detected signatures could have potentially come from a meteor entry, or if these are more likely caused by unrelated events.

To aid in the statistical analysis of such detections, an additional reference time window is defined just before the arrival time window. This reference window has the same duration as the calculated arrival window, allowing this to be used to judge the resistance of the detection algorithm to false detections, since any peaks measured during this period must come from sources other than the meteor since its sound cannot have arrived yet.

3.1.4. Detection algorithm

To detect the presence of a meteor signature in the infrasound data, a threshold detection algorithm is used in the frequency domain. For this detection algorithm, first a spectrogram of the trace is created using the mlab.spectrogram function implemented in the matplotlib python library (Hunter, 2007). For the creation of the spectrogram, an FFT length of 256 samples per segment is used. This gives a sampling period of approximately 2.5 seconds, which is long enough as to not be limiting the lower end of the sensitive frequency range of the sensor, while not being too long as to where the potentially short infrasound signature (which could be as short as a single wave) would become harder to detect due to being lost in the long time period averaging. Additionally, each subsequent segment has an overlap to the previous segment of 128 samples. This half window overlap ensures that if a single-wave signal shows up in the data, it is always fully included in at least a single spectrogram segment, rather than being cut in half by the FFT window. No windowing function is applied to this data before the FFT is applied, since applying a windowing function could hide potential frequency peaks if these happen close towards the edge of the window. Although this downside of applying a windowing function could be reduced by increasing the overlap between subsequent FFT's, this does significantly increase the computational cost of running the detection.

To turn these FFT spectra into potential detections, the detection algorithm applies a threshold method. A baseline noise level is determined by averaging the power spectrum over the previous 30 seconds, which corresponds with 24 samples of the FFT. If the signal exceeds this reference value by more than the determined threshold value, a peak detection is triggered. During the duration of this detection, the threshold value is temporarily frozen to prevent the peak which triggered the detection from affecting the detection, which could otherwise lead to the algorithm assuming the detection has ended before it actually has. Once the signal drops below this threshold value for all relevant frequencies, the detection event is closed and the threshold is unfrozen again. To determine the threshold, two approaches have been tested. The first approach determines the threshold as a multiple of the baseline noise levels. This is the simplest approach, since this baseline noise level has been calculated regardless, and this threshold is thus easily available. The second approach used for determining the threshold is to set the threshold as a multiple of the standard deviation in the noise levels. This approach

has the advantage that it could lead to a better sensitivity in frequency bands where there is a relatively high but consistent noise level due, e.g. due to the presence of a nearby noise source in a particular frequency band. Additionally, this approach has the potential to allow a more consistent threshold value for different conditions, as it scales more closely with the peakiness of the signal.

After determining the trigger points for the relevant infrasound signal, each of the triggers is matched against both the estimated arrival window, and the reference window. Based on these matches, the detection is classified as one of six categories. If the total number of triggers is greater than 2, the signal is assumed to be too peaky to draw any meaningful conclusions regarding actual meteor detections, since a meteor does not typically generate this many peaks. As such, this could indicate that either the threshold is set too low, or the environment is too noisy to conclusively detect meteor entries. Additionally, if the total duration of the triggers exceeds the duration of the detection window, this once again indicates that the threshold is too low, since the trigger never actually released once it had started. If all triggers took place within the estimated arrival window, this detection is assumed to be a proper detection of a meteor infrasound signature. Conversely, if the triggers all took place within the preceding reference window, this is counted as an confirmed false positive, indicating that the detection algorithm detected something which was not a meteor. If no peaks were detected, the signal is classified as a non-detection. Lastly, if the detected peaks did not meet any of the above criteria, either by having a peak in both the arrival and the reference window, or by having detections which cross the edges of the detection window, the signal is classified as inconclusive.

For each of the threshold defining algorithms, the results are tested both with fixed threshold multipliers, and with a dynamic threshold. For a dynamic threshold, the detection algorithm is re-run with slowly increasing multipliers until the result can be classified as either a detection, a confirmed false positive, or a non-detection. A global overview of this detection algorithm can be seen in figure 3.3.

3.2. Noise reduction

This section covers the research into noise reduction in infrasound setups. Subsection 3.2.1 covers the design and characteristics of the test setup used for this investigation. Subsection 3.2.2 will cover the methodology used for analysing the data generated by the test setup and estimating their effectiveness.

3.2.1. Test setup

During the experiment, three different setups were tested for their noise characteristics. First, a baseline scenario was tested, with the bare sensor without any noise reduction methods. The specifics of this setup, and the setup considerations are discussed in section 3.2.1. Next, a setup with a porous hose attachment was tested. This setup is discussed in more detail in section 3.2.1. Thirdly, a fabric dome setup was tested. The design of this dome and the test setup is discussed in section 3.2.1.

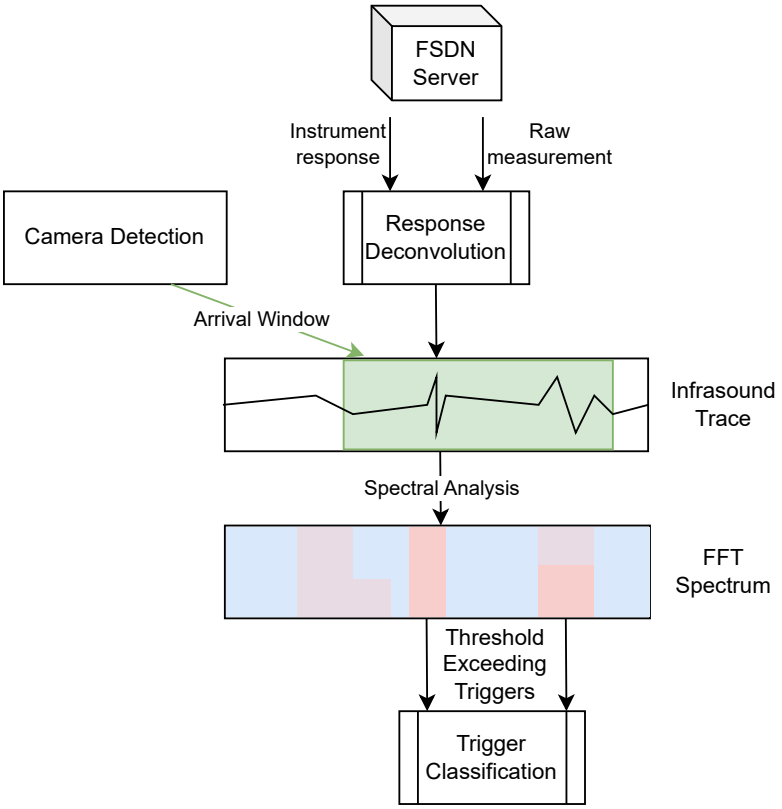


Figure 3.3: A schematic overview of the detection algorithm employed in this research

Basic sensor mounting

The sensor used for the experiment is a Raspberry Boom infrasound sensor manufactured by the company Raspberry Shake. This sensor consists of a single microbarometer with a sampling frequency of 100 Hz which connected to a Raspberry Pi 4B. The microbarometer used in the sensor measures the pressure differential between two ports. One port is connected directly to the outside air (through a short length of tubing) and is responsible for providing the relevant sound pressure variations. The other port provides the reference air pressure. For this purpose, it is connected to the outside air on the same external port on the enclosure, but has an additional mechanical filter between the outside port and the sensor port. This mechanical filter comes in two different variations: A filter with a 1 s response, and a filter with a 20 s response. This filter allows the sensor to maintain a average zero pressure balance between the two ports, thereby keeping the measurements in the sensitive region, while still allowing the detection of signals with periods exceeding the filters' period. For this research, the default 1 s mechanical filter was used.

The sensor including the Raspberry Pi are all located in an IP67 rated enclosure. This enclosure was mounted on the TU Delft Rooftop Lab as shown in figure 3.4. The sensor was attached to the grid using zip ties. Due to the low power requirements of the entire setup,

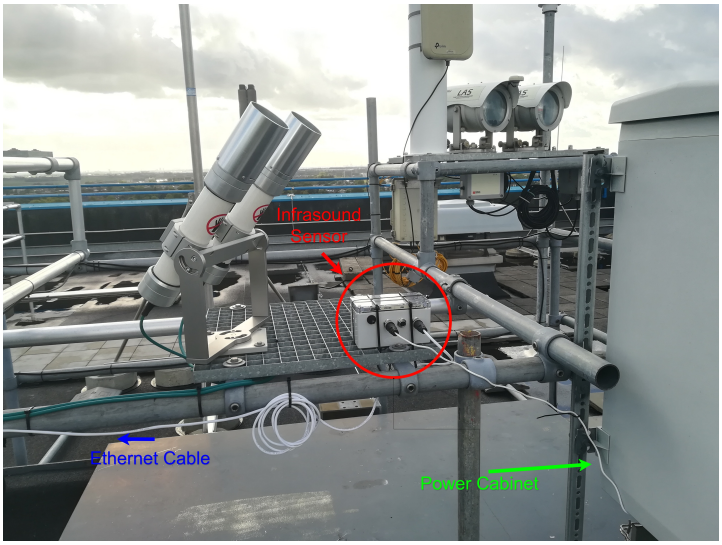


Figure 3.4: The bare infrasound sensor located on the TU Delft Rooftop Lab

it can be fully powered using the standard Raspberry Pi 5V 3A power adapter. Additionally, the sensor is connected to a firewalled internet network, allowing the sensor to upload its measurement data directly to the Raspberry Shake FSDN data network Raspberry Shake, S.A., 2016. Alternatively the data can be retrieved from the sensor through SSH through a different computer that exists within the same network.

Besides the infrasound sensor, the Rooftop Lab of the TU Delft is also equipped with an Anemometer which measures the wind speed and direction on the roof. The sensor provides wind data in the form of a three component wind vector at a slightly variable sampling rate of approximately 30 Hz. This wind data can be used to correlate the noise measurements with the environmental conditions to get a better understanding of the impact of the test setup in these differing conditions.

Porous hose design

Whereas previous studies have typically used large arrays of porous hoses for filtering, requires significant amounts of free space around the sensor to lay out each hose in its own direction. For example, the sensors in use at the IMS network use pipe rosettes with diameters between 18 m to 70 m (Christie & Campus, 2009). Since the goal of this research is to assess the viability of a denser network of infrasound sensors to be deployed alongside existing camera-based networks, it cannot be assumed that such space will be available at desired all infrasound station locations, if any at all. This includes the location of this experiment, the rooftop lab of the TU Delft Aerospace faculty, which has a width of only approximately 15 m, with a significant portion of the space already being reserved for other instruments. As such for this experiment, a singular porous hose is used instead of a large array, as has been found to be a potentially suitable alternative to full multi-hose rosettes (Robinson *et al.*, 2020; Walker & Hedlin, 2009).

The porous hose used for this experiment is a Rain Bird drip hose segment, with a length of 3.0 m. Along the length of this hose, 2.5 mm holes are spaced at 30.5 cm intervals, for a total

of 10 holes along the full tube. The hose has an inner diameter of 1.27 cm. The far end of the hose was closed off with a plug to ensure the wind pressure at this point would not overpower the desired filtering effect of the much smaller holes along the hose. A close-up of the holes in the drip hose can be seen in figure 3.5. In this figure, it is also possible to see a part of the reinforcement of the holes, which usually reduces the chance of blockage of the hole due to e.g. plant growth.

Figure 3.5: A close-up view of the holes in the drip hose used in the porous hose filter. (Rain Bird, 2025)

Since the hose cannot be directly mounted to the Raspberry Boom infrasound sensor due to the size difference of the hose with the sensor port, the hose was connected through an adapter to a length of 75 cm of flexible rubber tubing with an inner diameter of 5 mm which was then connected to the sensor port. Besides functioning as an adapter, this piece of tubing also functions to provide additional flexibility in the mounting orientation of the porous hose.

The porous hose is positioned along a curved path. running away from the sensor. This curved path is used both for practical mounting limitations, and in an attempt to limit the directionality of the noise filtering capabilities of the hose. The final setup of the infrasound sensor with the hose attachment can be seen in figure 3.6

Fabric dome design

The final approach tested in this research is the use of a porous fabric dome. Compared to the porous hose setup, this design has more stringent requirements on the available space close to the sensor. Since the dome must fully enclose the sensor and cover a large enough area surrounding it, any obstacles close to the sensor will limit the maximum size available for the porous dome. Due to the mounting location of the infrasound sensor on the rooftop lab, these limitations consisted of the an unrelated instrument located on the left of the sensor, and a raised rail located on the right of the sensor, as can be seen in figure 3.7.

Due to these limitations, the final design of the porous dome consisted of a hexagonal cone shape. The hexagonal base has an outer diameter of 42 cm, and a height 25 cm. While a larger dome would be beneficial to the filtering of low frequencies, this was not possible due

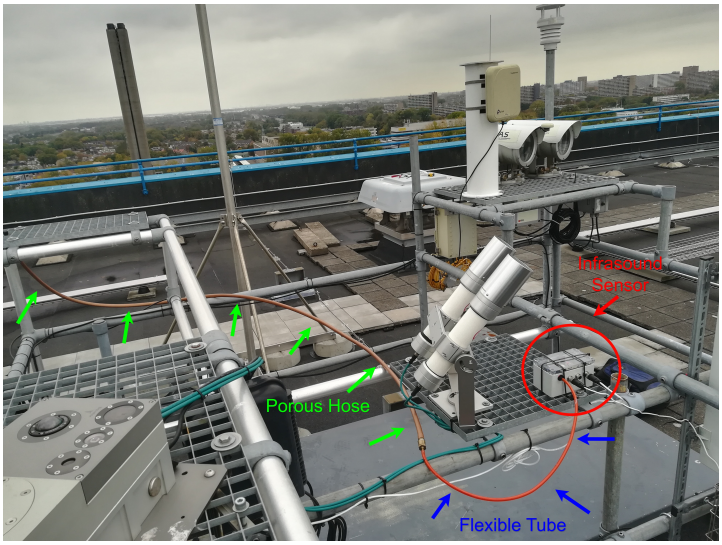


Figure 3.6: The infrasound sensor with porous hose attachment

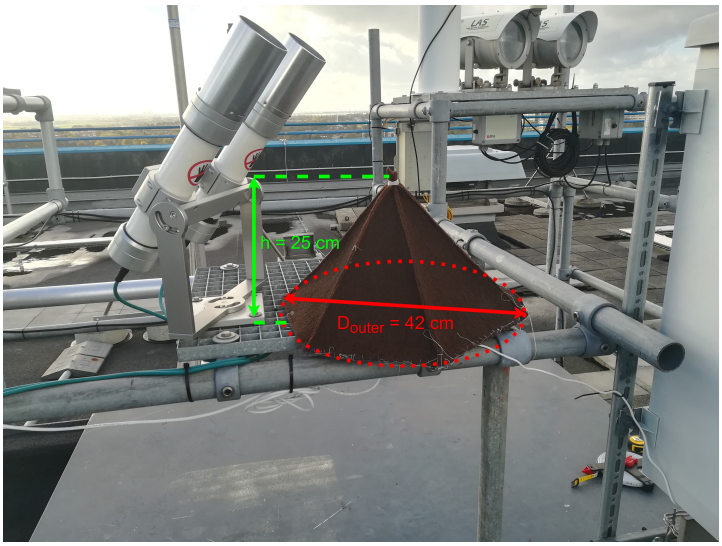


Figure 3.7: The infrasound sensor with the fabric dome filter mounted over the sensor.

to a combination of space and material constraints. The cone was constructed from from a synthetic isolation fabric with a porosity of approximately 80%. The hexagonal dome shape was used due to its similarity to a fully conical dome, thereby providing a level of directional independence in its filtering relative to the wind direction, while being significantly easier to construct that a conical dome (Assink, 2025). Each side of the dome was cut separately from a sheet, and are tightly stitched together to prevent any excessively large holes in the dome. The vertical edges of the cone are reinforced with metal rods to help the cone maintain its shape.

To accommodate mounting the dome over the sensor, a rectangular hole is cut in the bottom panel, tightly fitting around the enclosure of the infrasound sensor. The cables were routed through a single cutout, which was stitched shut around the cables. This dome was then secured to the mounting frame of the sensor using additional thread stitchings. The final fabric dome setup can be seen in figure 3.7

Signal impact validation

Whereas the measurements on the rooftop using the setups described in sections 3.2.1 to 3.2.1 give insight into the noise reduction of the different filters, these measurements do not give the full picture. To determine the effectiveness of the setup, the noise reduction must be compared to the change in signal strength caused by the filters. As such, each of the filters are recreated in a controlled indoor environment to determine their impact on the detection of coherent signals.

To test the impact of the filters on signal detection, a Pioneer XV-DV303 audio system was used to generate a persistent sinusoidal tone at frequencies of 20 Hz, 30 Hz, and 40 Hz, controlled by an Asus TUF A16 laptop. Although it could have provided additional insights to analyse the impact of the filters for lower frequency signals, this was not possible due to limitations of the sound production setup, which struggled to produce any meaningful signal below 20 Hz. Analysing higher frequencies was however deemed irrelevant as the sensor cannot detect any signals above 50 Hz due to the sampling rate of 100 samples per second. The infrasound sensor is placed at a distance of 5 m from the speakers / subwoofer and set up to record the sound level both with and without the filters attached. Each of the control tones was played and recorded for a duration of at least 60 seconds to minimise the impact of peaks in background noise levels due to e.g. a wind gust or passing vehicle. For this setup, an indoor version of the Raspberry Boom sensor was used due to availability. This indoor version is functionally equivalent to the outdoor version used in the Rooftop Lab experiment, except that the enclosure does not have an IP67 rating against water and dust ingress.

3.2.2. Analysis

This section will describe the procedure used for analysing the infrasound measurement produced during the experiment into actionable results.

Similar to the data preprocessing explained in section 3.1.2, the infrasound sensors used during the experiments generate measurements in the form of a unitless sequence of pressure measurements. As such, the same pre-processing method is used to turn this data into a trace of actual pressure measurements.

An important parameter in the analysis of the noise reduction of the differing setups is their effectiveness at different frequencies. As such, the temporal pressure trace must be converted to a noise spectrum in the frequency domain. For this purpose a power spectral density (PSD) is created of the pressure trace. This PSD is made using the mlab psd function provided by the matplotlib python library Hunter, 2007. For the calculation of the PSD, the trace is split into segments of between 1 and 10 minutes. These segments are then subdivided into stretches of 256 data points, on which the FFT is applied. All stretches within these 1 to 10 minute windows are then averaged by the matplotlib psd function to get the average power spectral density within that time window. The FFT length of 256 data points is used as this being a power of 2 allows for the most efficient calculation of the FFT, and by having a trace duration of more than 2 seconds (due to the sensor having a 100 Hz sampling rate) ensures that the entire sensitive range of the sensor is included in the final PSD spectrum. Additionally, a Hanning window is

applied to the the data to reduce spectral leakage between the different frequency bands.

The wind data measured by the anemometer consists of a timestamp, a wind vector consisting of 3 scalar components, the environment temperature, and the status information of the anemometer. To reduce the noise in the wind measurements, each of the individual components is averaged over the same 1 to 10 minute window used for the noise measurements. The average wind speed is then computed by taking the length of the vector formed by the averaged components. Additionally, the horizontal wind angle is computed using the two horizontal components of the wind data.

By filtering both the wind data and the noise measurements over the same time window, it becomes possible to correlate the noise levels of the sensors with the differing environmental conditions.

3.3. Network analysis

The third aspect of the investigation focusses on the size and layout of a potential infrasound network. Since then main target of such an extension of an existing network with infrasound sensors is the FRIPON network, this investigation will make use of the same fireball detection database used for the investigation into the detection of meteor infrasound as explained in section 3.1. Additionally, many of the important parameters for such a network, such as the distance between the sensor and the fireball, and the angle of the infrasound wave relative to the meteor's trajectory are also important parameters in the detection and thus will be calculated using the procedures laid out in section 3.1.3. Using these parameters, the efficiency of various network layouts can be estimated for this region of the world. This part of the investigation will focus on the potential of an infrasound network within the local branch of the FRIPON network, called DOERAK.

The investigation will look at the potential of implementing a small scale infrasound network on just the Dutch part of the FRIPON network. For this investigation, the focus will be on determining where such infrasound sensors should optimally be placed to provide the greatest scientific potential. Since for such a network the focus would be on further research within the Netherlands, the scope of the fireball database must be limited to just those fireballs for which a Dutch FRIPON camera was involved since this would be a limitation which would be encountered regarding accessing recent fireball data for Dutch researchers. For this investigation, a constraint has been set that the infrasound network would initially consist of three infrasound sensors, one of which would be mounted at Delft University of Technology, one sensor would be mounted at a residential location within Tilburg, and the third sensor being a free variable regarding its final placement.

4

Results and Discussion

The investigations explained in chapter 3 have produced multiple interesting results regarding the applicability of large scale infrasound networks for meteor detection, and the setup of the sensors in such a network. This chapter will therefore present the relevant data produced by these investigations. First section 4.1 will cover the results from the investigation into detection algorithms when applied to publicly available infrasound measurements. Next, section 4.2 will cover the results from the experiments regarding the reduction of background noise using different filtering setups.

4.1. Detection performance

This section will cover the results regarding the detection of meteor infrasound signals using publicly available data from the FRIPON and Raspberry Shake network. For this analysis, a total of 263 infrasound traces are available. A trace in this context corresponds to a continuous infrasound signal measured during a period of interest. As such, it is possible for a single meteor entry to lead to multiple traces if multiple infrasound stations were active and within range during the relevant measurement window. Each of these traces collected from the Raspberry Shake network has been analysed and classified using the procedure covered in section 3.1.4. Table 4.1 shows the results from the detection threshold approaches at different multiplicative threshold factors.

As can be seen in table 4.1, the majority of the infrasound measurements do not include any detectable infrasound trace, with even the dynamic approaches only finding any potentially matching infrasound signatures in just 7.6% of all measurements for the baseline threshold, and performing even worse when using the standard deviation for determining the threshold at just 5.7%. Additionally, it can be seen that the ratio of detections compared to false positives, where an infrasound signature was detected only in the reference window, is approximately 2:1. Because of this, it is likely that the actual percentage of infrasound traces containing actual meteor signatures detectable using such a dynamic threshold value is closer to $3\% - 4\%$. For

Table 4.1: Detection performance for the automatic detection algorithm with different detection thresholds

Method	Peaky	Over-sensitive	Detection	False positive	Non-detection	Inconclusive
Dynamic baseline	-	-	20	8	235	-
Dynamic standard deviation	-	-	15	7	241	-
Baseline (2.5)	0	263	0	0	0	0
Baseline (10)	26	237	0	0	0	0
Baseline (20)	102	58	6	3	93	1
Baseline (50)	33	17	5	1	205	2
Baseline (100)	18	7	4	3	228	3
Standard dev (2.5)	0	263	0	0	0	0
Standard dev (10)	15	248	0	0	0	0
Standard dev (20)	99	53	4	2	103	2
Standard dev (50)	31	14	2	2	212	2
Standard dev (100)	18	3	3	3	233	3

reference, the waveforms and trigger points for each of the detections by the dynamic baseline approach can be found in appendix A.

When using a static threshold value, the detectability of potential infrasound traces becomes significantly worse than the dynamic threshold algorithms. For threshold factors of 10 or lower, meaning that the signal must exceed either 10x the baseline noise level or deviate from the mean noise level by at least 10x the standard deviation depending on the method used, neither method is able to distinguish any small individual peaks in the data. Instead, the total duration of the triggers exceeds the arrival window length in almost every case. In the cases where the duration doesn't exceed this time length, instead the number of detected peaks is high enough (> 2), indicating that it would be effectively impossible to truly assign any of these particular peaks to anything other than background noise.

For threshold factors of 20 or higher, no obvious ”optimal” factor seems to be present. Although both methods have their highest detection count at a factor of 20, at these factors a significant subset of the data still suffers from an overly sensitive detection at this value. Raising the threshold factor to a higher value on the other hand seems to mainly increase the non-detections, with some detections being reclassified as non-detections. In fact, when looking at which events were actually detected, it can be seen that the counted events at a factor of 20 are a fully distinct set compared to those detected at a factor of 50, which apart from a single overlapping detection for the baseline approach is itself a fully different set from the detections at a factor of 100.

Based on these detections, it can be concluded that when trying to detect infrasound signatures in publicly available data such as those from the Raspberry Shake network, it is infeasible to set a universal threshold factor which can be used for all analysis. The most important driving factors behind this varying noise performance are likely the unknowns in the setup of the used infrasound sensors. Since the infrasound sensors in the Raspberry Shake network are owned and operated by private individuals, there is little to no information regarding the setup of the sensor. Based on the available data, it cannot be determined where the sensor is mounted (inside or outside of a building) and what type of wind noise filter is used or if they are used at all. Additionally, no information is available regarding the environment of the sensor, including the wind surrounding the sensor and potential infrasound sources close to the sensor. As such, using publicly available infrasound data from such networks for the detection of meteor entries will likely remain unfeasible for the foreseeable future.

When comparing the baseline approach to the standard deviation approach, it can also be noted that the performance for each of these approaches is fairly similar at equivalent threshold factors. This indicates that within the noise data, the standard deviation of the measurements is likely similar in magnitude to the average noise level in at least some of the frequency bands. Overall, the baseline approach seems to be slightly more responsive to detecting potentially interesting infrasound signatures, although the relatively low number of total detections and unknowns of the infrasound data make it hard to determine exactly by how much.

A comparison of what could be achievable regarding the detections of meteor infrasound using a strictly defined sensor layout can be found by looking at the results of Silber and Brown, 2014. During this investigation, meteor infrasound was detected by combining camera based detections from the Southern Ontario Meteor Network (SOMN) with infrasound measurements performed at the Elginfield Infrasound Array. This investigation covers approximately 5 years of meteor detections by the SOMN network, with the infrasound array being located in a central location, well surrounded by cameras from the SOMN network. During this time, Silber and Brown, 2014 detected a total of 90 recognisable infrasound signatures originating from 71 different meteor entry events. When compared to the approach using public infrasound data from the Raspberry Shake network as done in this investigation, the total data set spans effectively the same duration, and this investigation utilises stations more spread out over the area of interest, thereby in theory allowing for an equal if not improved detection count. Nonetheless, even in the best case scenario, the Raspberry Shake data only allowed for an estimated 12 true positive detections in this infrasound data.

Additionally, in the investigation, Silber and Brown, 2014 further classified the detected infrasound traces through the number of N-waves visible in the infrasound signature. For comparison, a preview of the waveforms detected as a positive detection using the dynamic baseline method can be seen in figure 4.1. Higher resolution versions of these traces can be found in appendix A. When looking at the results, only traces l, k, and m can be found to have a clear single-wave N-wave form visible in the infrasound trace, making up approximately 15% of the detections. When compared with the results from Silber and Brown, 2014, who found that approximately 75% of the traces were clear single N-wave signals, this indicates that applying

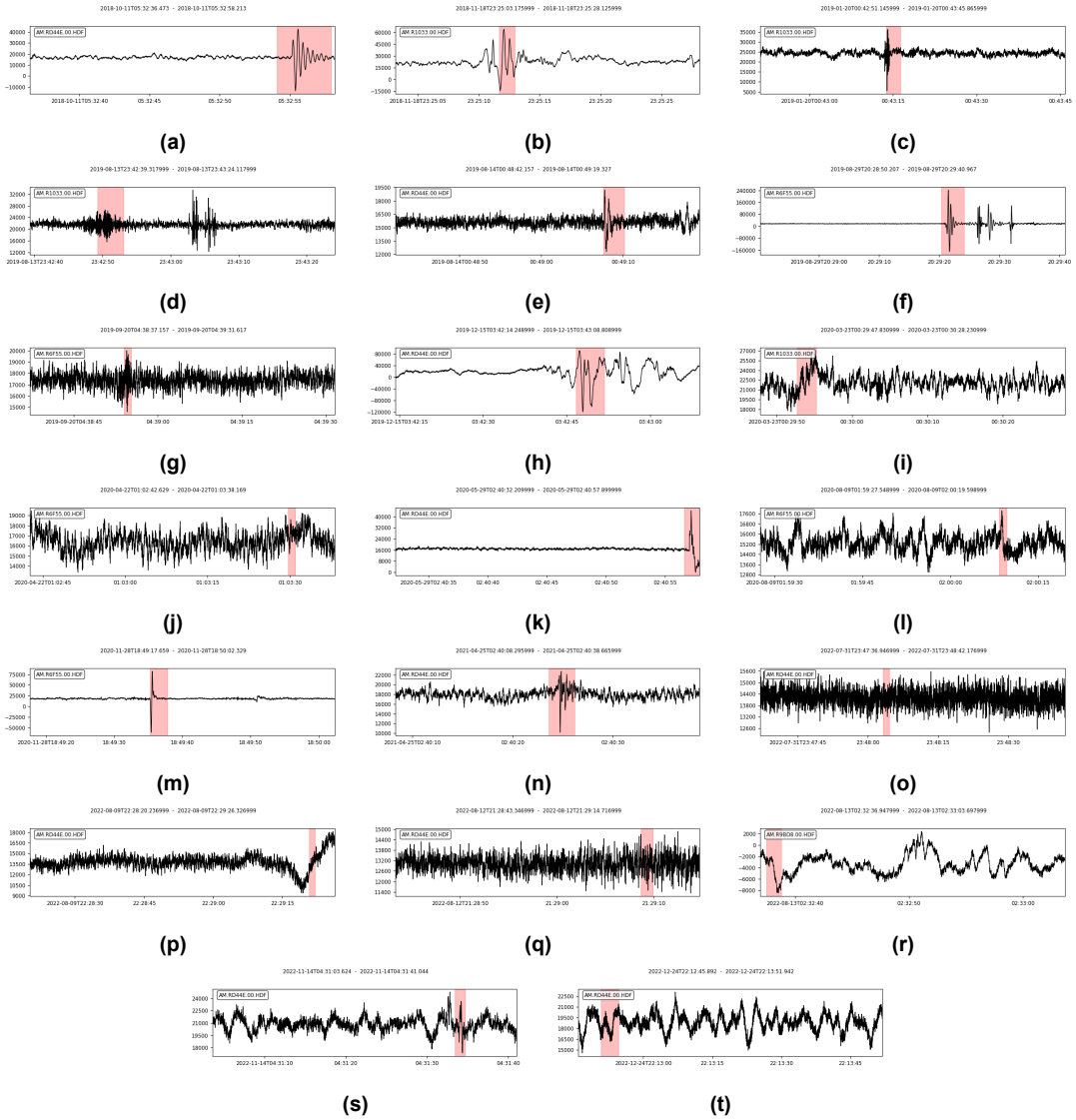


Figure 4.1: Infrasound peak detections using a dynamic threshold relative to the base noise level

similar classifications to these detections is potentially less feasible due to the low quality of many of these detections. This also limits the usefulness of this public data in developing improved detection algorithms, as many of the features such an algorithm might look for are not present in the data. If a more advanced algorithm were to be applied to the infrasound data generated by the Raspberry Shake network, such as an algorithm that explicitly looks for an N-wave like shape in the time domain, it is likely that this algorithm would achieve even fewer detections than the rudimentary threshold algorithms applied in this investigation. As such, at that point any conclusions drawn from a comparison with this threshold algorithm would be lost, as the detection count for both true and false positives would be too low to hold any statistical significance.

For the development of more advanced automatic algorithms, the dataset used by Silber and Brown, 2014 has since been released as a public dataset (Silber *et al.*, 2025). The timing of the public release of this dataset however did not align with the schedule set out for this

thesis. As such, this avenue has not been explored further in this thesis.

4.2. Wind noise reduction performance

For the investigation into the effectiveness of different filtering methods, there are two distinct experiments which need to be analysed. The first experiment to be analysed consists of the signal impact analysis using the indoor test setup, which will be covered in section 4.2.1. The second experiment, concerning the noise levels of the different setups, will be covered in section 4.2.2

4.2.1. Signal impact analysis

This section will cover the results of the signal impact analysis performed on the different setups using known infrasound signals. As explained in section 3.2.1, the setups were each tested with a tone produced at 20, 30, and 40 Hz. Due to the alignment of time schedules not allowing all three setups to be tested simultaneously, the experiment has been performed twice, each time covering two setups. Each of the experiment runs contained the bare sensor as a reference, and either the hose attachment or the fabric dome cover.

Figure 4.2 shows the power spectrum measured by the infrasound sensor during each of the different frequencies. As can be seen in this figure, during the 20 Hz test tone, another distinct peak is visible in the power spectrum around 40 Hz. This peak is likely caused on the signal generation side, either by the laptop generating the initial tone, or by the amplifier in the stereo system due to nonlinearities in the amplifier. Since neither of these devices is made specifically for producing perfect tones at such low frequencies, something which is also noticeable in the fact that producing signals below 20 Hz did not produce any measurable response, it is not unreasonable to experience some artifacts at the harmonics of this 20 Hz tone. Additionally, due to the artifacting showing up with similar strength for both sensor setups, it is unlikely that this is caused by the filter.

Although not visible in the data of figure 4.2, it should be noted that the power spectra of the 30 Hz tone and 40 Hz tone for the bare sensor setup in this first run were measured over a shorter time span. Due to an issue with the Raspberry Shake infrasound sensor, the measured streams had missing data gaps of 11.5 and 11.0 seconds respectively. Despite these gaps, the gathered data should still be sufficient to draw conclusions regarding the effectiveness of the filters due to each run lasting at least 60 seconds, and the FFT's in the power spectrum estimation requiring only 2.5 seconds of consecutive measurements to achieve the full resolution.

Figure 4.3 shows the power spectrum measured during the second validation run comparing the bare sensor with the dome cover. Similarly to the first measurement run with the hose filter, a 40 Hz peak is once again visible in the 20 Hz measurement. Apart from this, no problems were encountered during this test.

Table 4.2 shows the total reductions of the signal due to the presence of the wind noise reduction attachment during each of the respective test runs, where a negative value indicates a reduction of the signal, and a positive value indicates an increase in the strength of the signal.

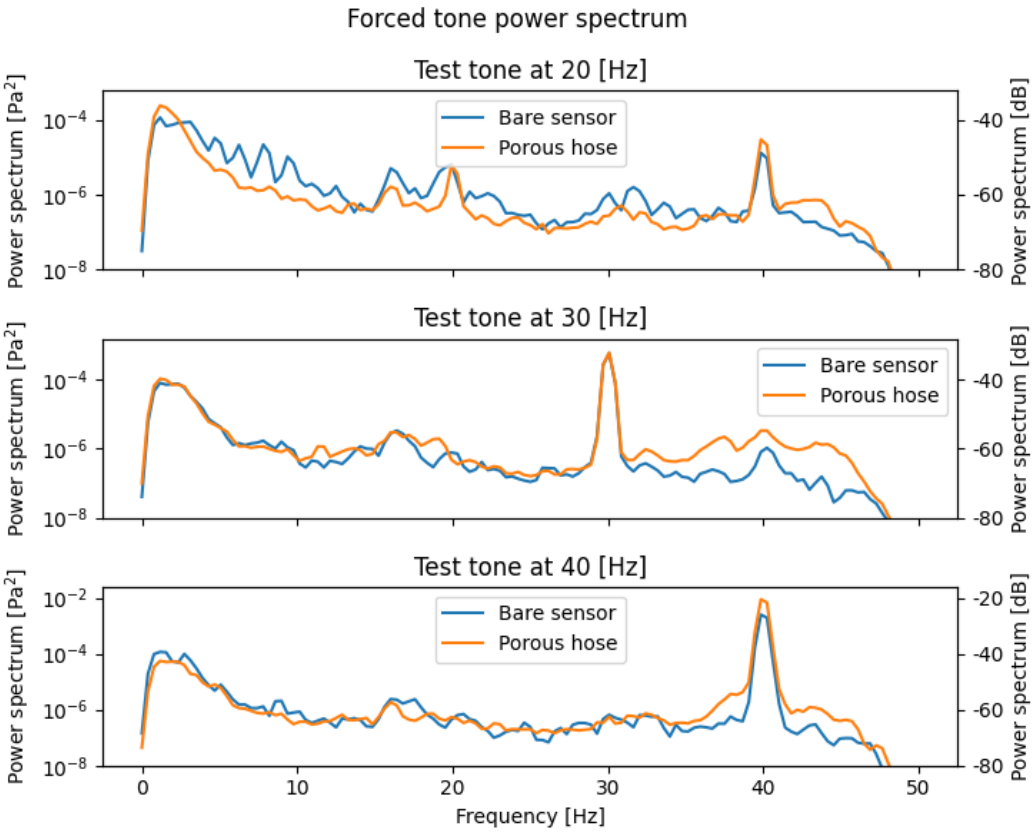


Figure 4.2: Power spectrum measured during a forced tone for the bare sensor and porous hose setups

As can be seen from this data, at the lower end of the measurable frequency range both filters perform fairly similar in that neither filter significantly alters the sensor response. Although the porous hose shows a minorly higher reduction of the signal in the 20 Hz band, this can be explained from the full spectrum for this test in figure 4.2, where the bare sensor showed a significantly less consistent baseline measurement than any of the other test runs for both this setup, and the dome setup shown as shown in figure 4.3. As such, this inconsistent baseline is likely caused by the environment being more noisy during this test due to activity outside of the test setup, rather than the behaviour of the sensor itself. Nonetheless, the actual peak value for this 20 Hz tone during the porous hose run can be seen to still coincide closely with the peak of the bare sensor, supporting the conclusion that the small level of attenuation visible in table 4.2 is due to measurement error rather than due to the hose attachment.

At the 40 Hz peak, a more distinct difference is visible. At this frequency both filters showed an increase in the response compared to the bare sensor. This increase would indicate that the filter amplifies the forced tone rather than attenuating it. Even so, the increase in amplitude for the fabric dome sensor was small at an increase of just $+1.3$ dB. This magnitude is still similar to the magnitude difference measured at the 20 Hz and 30 Hz tones, and can thus likely be attributed simply to measurement noise rather than being a caused by the dome itself. Additionally, even if this effect is caused by the fabric dome, its impact on any actual infrasound measurements is unlikely to significantly affect the detection capabilities of the

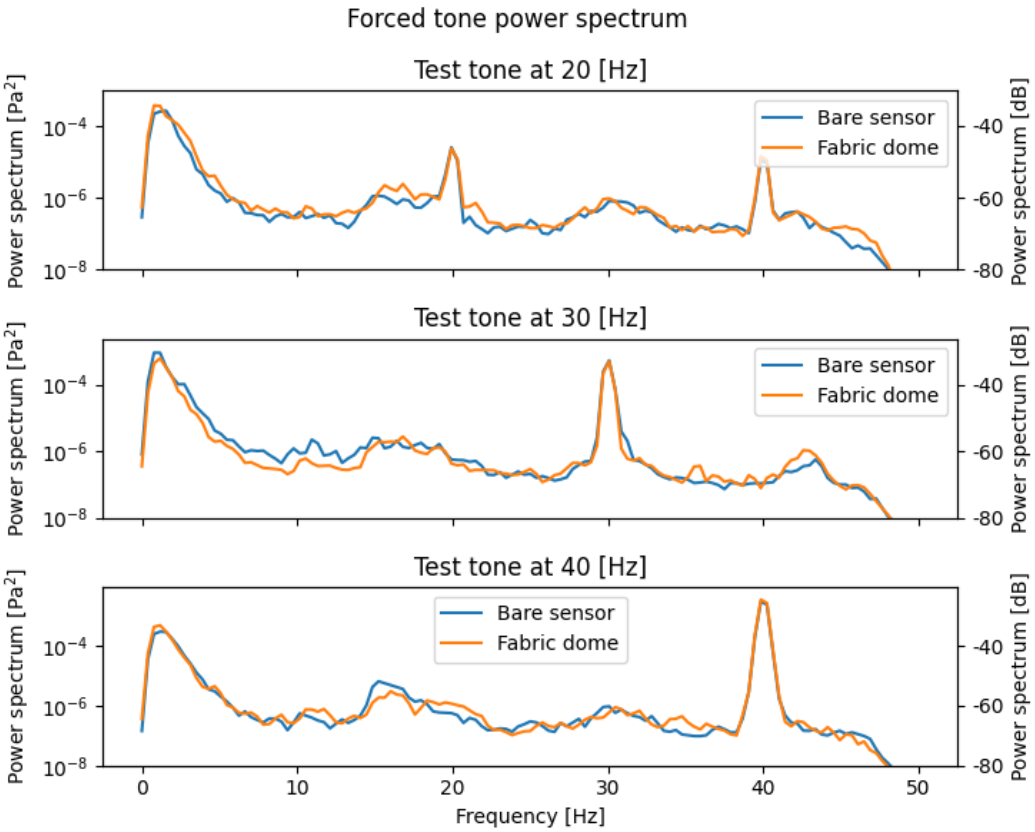


Figure 4.3: Power spectrum measured during a forced tone for the bare sensor and fabric dome setups

sensor as it is close to the noise caused by the sensor variance visible in the bare sensor test. The porous hose setup on the other hand shows a more significant increase in the magnitude of the sensor at an increase of 12.5 dB. An interesting aspect of this increase is that rather than being focussed only on the 40 Hz peak in the relevant test run, this increase is also visible in the 30 Hz test and to a lesser extent in the 20 Hz test as can be seen in figure 4.2. Additionally, the increase in PSD due to the attached hose is visible in frequencies starting at 30 Hz for the test run with the 30 Hz tone, and 35 Hz in the test with the 40 Hz tone. This increased response is then sustained up until the upper limit of the sensor measurement capabilities where the frequency pre-filter of 45-50 Hz begins to cut in. This indicates that at higher frequencies the hose will significantly affect the signal which could complicate its use for detecting these higher frequencies which would normally correspond to smaller meteors.

Table 4.2: PSD reduction for the porous hose and fabric tent wind noise filter setups relative to the bare sensor, achieved in the relevant frequencies during their respective test runs

Setup	**Peak ratio (20 Hz)**	**Peak ratio (30 Hz)**	**Peak ratio (40 Hz)**
Porous hose	-2.7 dB	$+0.4$ dB	$+12.5$ dB
Fabric dome	-0.4 dB	-0.704 dB	$+1.3$ dB

4.2.2. Noise reduction

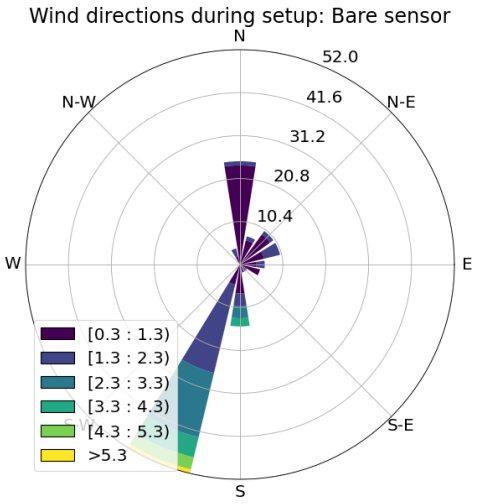
This section will cover the actual noise reduction achieved by the various setups. First, section 4.2.2 will cover the wind data measured by the anemometer on the Rooftop Lab. Next section 4.2.2 will look at the relation between the ambient wind speed and the measured noise level. Next, section 4.2.2 will analyse the impact of the wind direction on the two filtered setups.

Wind measurements

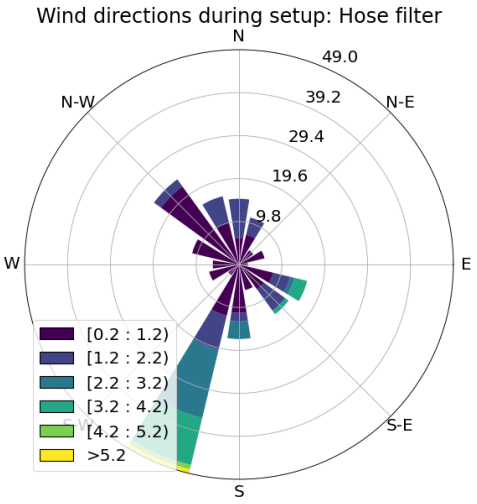
As stated in section 3.2.1, an important parameter in the noise generated by the infrasound sensor is the wind present at the setup. As such, before any comparison of the data can be made, first the results of the anemometer are to be analysed. Analysis of the wind data shows that the wind measurements of the anemometer show a significant level of noise and variability. In low wind conditions, where the time-averaged wind has a magnitude of $V_{wind} < 1.0$ m s^{-1}, the individual wind components show a typical a standard deviation of as much as 1.5 m s^{-1} for the horizontal components of the wind vector, and up to approximately 1.0 m s^{-1} for the vertical wind component measured by the anemometer. As the wind increases, the standard deviation in the wind measurements similarly increases, reaching a standard deviation of up to 3.0 m s^{-1} during average wind speeds of $1 - 2$ m s^{-1}, and 3.5 m s^{-1} at wind conditions of $V_{wind} > 2$ m s^{-1}. The vertical component shows a similar but less pronounced increase in its standard deviation, reaching 2.0 m s^{-1} and 2.5 m s^{-1} standard deviation for these wind domains respectively. As such, to achieve a reasonable estimate of the actual wind speeds at the sensor setup, an averaging duration of 10 min was chosen for each of the data points. This should provide a reasonably stable wind estimation, while still allowing for enough data points to draw conclusions regarding the efficiency of the different setups in various wind conditions. A scatter plot of the average wind speed versus the standard deviation can be found in Appendix B.

To understand the prevalent wind directions during this test figure 4.4 shows the wind directions that were measured on the Rooftop Lab of the TU Delft during the experiment. As can be seen in this figure, during every run a significant portion of the wind came from the south-west direction, ranging from almost wind free conditions up to winds of 5 m s^{-1}. As such, this wind direction provides a solid foundation for performing a full cross-comparison of the noise measured by each of the setups. Besides this south-west prevalence, both the porous hose setup and the fabric dome setup encountered a range of wind speed coming from the east-south-east. Although for the fabric dome setup the amount of measurements where these winds were present is fairly limited, the fact that this direction has wind speeds spanning effectively the same range as those encountered in the south-west peak make this wind direction a good candidate for a secondary comparison between the porous hose and the fabric dome. The bare sensor however not have this peak, and thus can not be included in this secondary comparison.

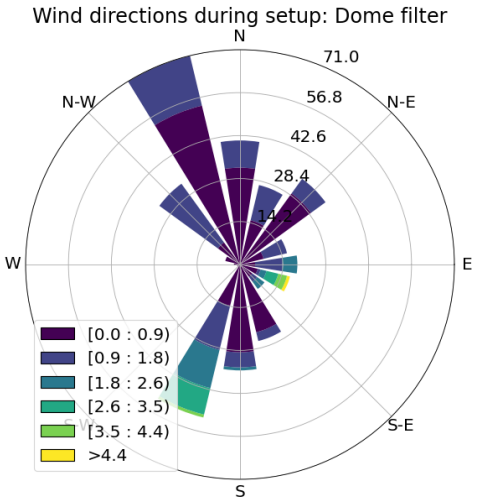
Wind speed impact



(a) Windrose plot of the period where the bare sensor setup was active



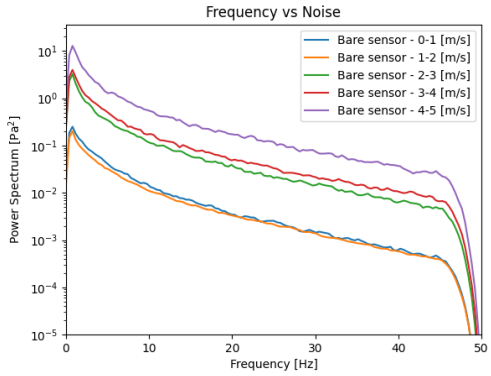
(b) Windrose plot of the period where the porous hose attachment was connected to the infrasound sensor



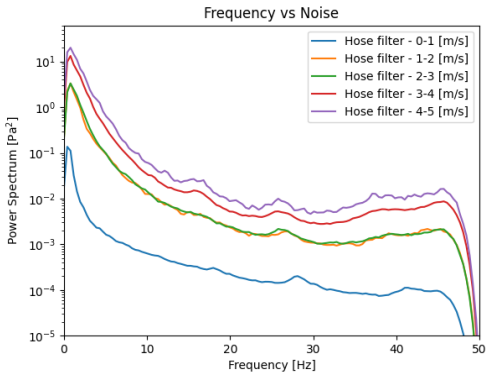
(c) Windrose plot of the period where the fabric dome attachment was placed over the infrasound sensor

Figure 4.4: Windrose plots of the wind experienced during the various measurement periods for each of the three setups

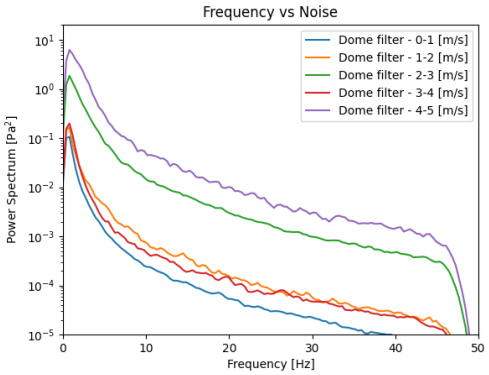
The first aspect to be analysed is the impact of wind speed on the noise level in the system. For this analysis, the data is filtered to only include the measurements with the wind in the south-western peak as described in section 4.2.2. Figure 4.5 shows the power spectra measured during the rooftop experiment for each of the different setups at various wind speeds.



(a) Power spectrum measured for the bare sensor without any filter attachment as a result of different wind speeds



(b) Power spectrum measured for the sensor with the porous hose attachment connected to the infrasound sensor as a result of different wind speeds



(c) Power spectrum measured for the sensor with the fabric dome attachment placed over the infrasound sensor as a result of different wind speeds

Figure 4.5: Power spectra recorded during various wind speeds for each of the infrasound sensor setups

As can be seen in figure 4.5, each of the measurements show a common trend where an increasing wind speed will cause a significant increase in the noise levels measured by the sensor. An interesting observation in figure 4.5a is that the power spectra for $0 - 1$ m s^{-1} effectively coincides with the power measured for the $1 - 2$ m s^{-1} speed range. There are two potential explanations that can be given for this phenomenon. The first possible explanation is rooted in the fact that, as discussed in section 4.2.2, the wind measurements themselves are fairly noisy. As such, it is not unreasonable to assume that for some runs with a medium wind speed ($1 < V_w ind < 2$) the wind data has averaged out to be closer to 0 than it should have been if a perfect wind sensor was used. This would cause the zero-wind measurement to contain a few noisier samples, which in turn would increase the magnitude of the PSD. Similarly, some near-zero-wind measurements might have a wind average that due to this noise

increased to just above the 1 m s^{-1} threshold, incorrectly reducing the average for the $1-2$ m s^{-1} bin to be close to the $0-1$ m s^{-1} bin. This effect could have an especially strong impact if either of the two bins has significantly less measurements than the other due to the wind speeds falling primarily in the in one of the two bands. In such cases, the erroneously classified measurements could relatively easily overpower the correctly classified measurements by virtue of there being significantly more measurements which can erroneously be placed into this bin.

An alternative explanation for this behaviour is that potentially during the measurement period the wind speed remained close to 1 m s^{-1} for an extended period of time. This would bias both results to display noise levels that are more representative for just this on speed, rather than the average speed of the bin.

A similar behaviour is visible in the data for this wind direction for the porous hose attachment. In this case however, the overlapping bins are the $1-2$ m s^{-1} bin with the $2-3$ m s^{-1} bin.

The case with the fabric dome filter visible in figure 4.5c is more peculiar. Whereas in the other two cases, the overlapping bins were consecutive bins, making it more reasonable for them to coincide, in this measurement the $3-4$ m s^{-1} bin overlaps with the significantly lower-wind $1-2$ m s^{-1} bin. This effect is too significant to simply be explained by either of the two mechanisms listed above. As can be seen in the V_wind vs standard deviation plot of this sensor in Appendix B, the $3-4$ m s^{-1} bins contains numerous measurements with a significantly lower standard deviation than the majority of other measurements. Indeed filtering the data for only those data points where the wind measurement had a standard deviation $|std| >= 2.5$ removes this data point entirely from figure 4.5c, indicating that all data points used for this line were derived from this low-variance measurement series. What exactly caused this measurement to have such a low standard deviation is unknown. Nonetheless, there are two mechanisms which could explain this behaviour. In the first place, it is possible that during the particular day where this wind measurement was made the wind speed was just significantly more steady than during any of the other days. This explanation would be supported by the fact that in the data in Appendix B it can be seen that there are two distinct standard deviation bands for a particular wind speed depending on the wind direction. Nonetheless, these data points do not seem to fall within either of these two noise bands, instead appearing significantly lower than any of these. Therefore, an alternative explanation must also be considered that perhaps the noise measured in all other measurements is not actually caused by the wind but rather by an external mechanism such as interference from other sensors. If this noise is caused by electrical interference by data or power lines running to other experiments on the rooftop lab, it is possible that during this particular day this other experiment was turned off, thereby allowing the wind sensor to operate more accurately. If this is the case, this could imply that the actual wind speed used for the other traces of figure 4.5c is likely an underestimation of the actual wind speed, and therefore if this data series were to be corrected for this wind speed discrepancy should actually be included in a different bin. Regardless of whether this is the case however, this should not affect the conclusions from this investigation since this wind speed underestimation would be present in almost every other measurement, thus maintaining

the scaling of the infrasound noise level with the wind speed.

Overall, each filter shows a comparable wind speed impact to the overall magnitude of the spectrum. For each of the setups, when the wind speed increases from the $0-1$ m s^{-1} bin to the $4-5$ m s^{-1}, the magnitude of the noise increases by approximately 20 dB in the PSD. Overall however, the dome filter shows a seemingly slightly higher impact of increasing wind speeds, with the $2-3$ m s^{-1} bin already achieving a close to 20 dB increase compared to the low wind speed scenarios, although the $4-5$ m s^{-1} is only higher than this bin by approximately 3 dB, thus potentially remaining within the margin of error.

An investigation into the impact of wind speed at an IMS station in Bolivia has been performed by Woodward *et al.*, 2005. In this investigation, the wind speeds measured ranged from 3 m s$^{-}1$ up to 7 m s$^{-}1$, measuring frequencies ranging from 0.04 Hz to 4 Hz. Although this frequency and wind speed range differ somewhat from the measurements performed in this experiment, a comparison can still be made for the overlapping range from 3 m s$^{-}1$ to 5 m s$^{-}1$ at frequencies between 1 and 4 Hz. In this investigation, Woodward *et al.*, 2005 found that for a windspeed increase from 3 m s$^{-}1$ to 5 m s$^{-}1$, this corresponded to an increase in the power spectral density of approximately 10 dB at frequencies around 1 Hz. When compared with the results shown in figure 4.5, it can be found that the porous hose and fabric dome setups show a somewhat smaller difference between the $2-3$ m s$^{-}1$ bracket and the $4-5$ m s$^{-}1$ bracket, each only showing a difference in wind noise pressure of approximately 8 dB. in this region. This could imply that the wind speed measured on the rooftop lab is rather an overestimation of the actual wind speed, rather than an underestimation as was previously stated. At the same time, the small range of overlap between these experiments in both frequency and measured wind speed make it difficult to conclude this with any significant certainty, especially as can be seen in figure 4.5 that the difference in amplitude seems to come more in-line with the anticipated 10 dB at higher frequencies. If this is compared with the results from Woodward *et al.*, 2005, it can be seen that the wind speed impact is mostly independent of the frequency within the measured domain. As such, it is also possible that this difference is merely caused by the impact of the wind noise filters becoming more uncertain at these frequencies. Additionally, if rather than looking at the specific wind speed lines, the impact is interpolated between the $0-1$ to $4-5$ m s$^{-}1$ lines, an impact more closely matching the expected 10 dB can be found.

Wind direction impact

For the wind directionality impact analysis, only those measurements are included where the wind speed laid within the $2-3$ m s^{-1} bin. This ensures that a proper comparison can be made between each of the results, as the impacts of differing wind speeds are minimised. The results of this analysis can be seen in figure 4.6

As can be seen in figure 4.6, both the porous hose filter and the fabric dome have significantly lower noise levels when the wind comes from the east when compared to a south-western wind direction. Since the shape of the dome filter is close to fully rotationally symmetrical, with the hexagonal shape only having comparatively minor deviations from a purely conical dome, it is unlikely that this difference is fully caused by the filter itself. It is more likely that at least a significant part of this noise difference can be attributed to the interaction of the wind with

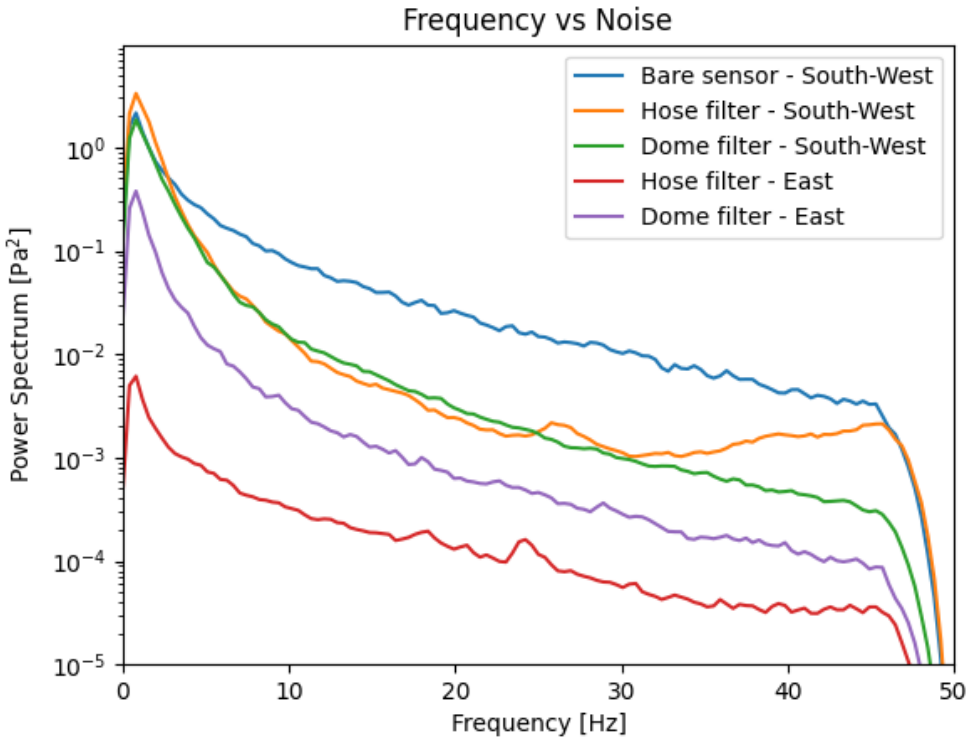


Figure 4.6: Noise level versus wind direction for each of the setups with $2 < V_{wind} < 3$ m s^{-1}

objects in the nearby surroundings of the infrasound sensor, such as the optical instrument visible on the left in figures 3.4 through to 3.7, or even turbulence around the building as a whole.

What can also be observed from figure 4.6 is that whereas the fabric dome has a difference between the two cases of approximately 7 dB, the porous hose has a difference in its noise reduction capabilities by approximately 15 dB relative to a south-western wind. This indicates that in the case of the porous hose, there is likely an additional mechanism which further reduces the noise levels for an eastern wind. For this phenomenon, two relevant mechanisms have been identified. Firstly, due to the length of the porous hose it covers a significantly larger area atop the rooftop lab. Due to the close proximity of the infrasound sensor to other nearby equipment, the positioning of the hose might mean that for certain wind directions its openings are located in a less disturbed part of the airflow due to there being less obstacles which introduce turbulence into the flow.

A contradicting process for this behaviour is related to the orientation of the hose. Although the curve of the hose was intended to reduce the directionality of the porous hose filter, the final mounting of the hose did result in a primary orientation of the hose running in the north-south direction. This means that for the south-western wind, the wind direction is mostly aligned with this primary hose direction, whereas for the eastern wind would be oriented almost orthogonally to the majority of the hose. As found by Walker and Hedlin, 2009, the length scales for wind generated noise along the wind direction are smaller than those in the cross-wind directions

by approximately a factor 3. This means that for the eastern wind, the hose would cover less distinct noise regions, thus expecting potentially higher noise levels for this wind direction. As such,this effect is likely not to affect the porous hose too strongly, with the turbulence of nearby objects having a significantly larger impact on the measured noise.

Filter performance comparison

The performance of all three setups at fixed conditions of $2 < V_{wind} < 3$ from a south-western direction can be seen in figure 4.7. In this figure, the marked area surrounding each line corresponds to the area marking the 5% and 95% noise levels in each of the frequency bands.

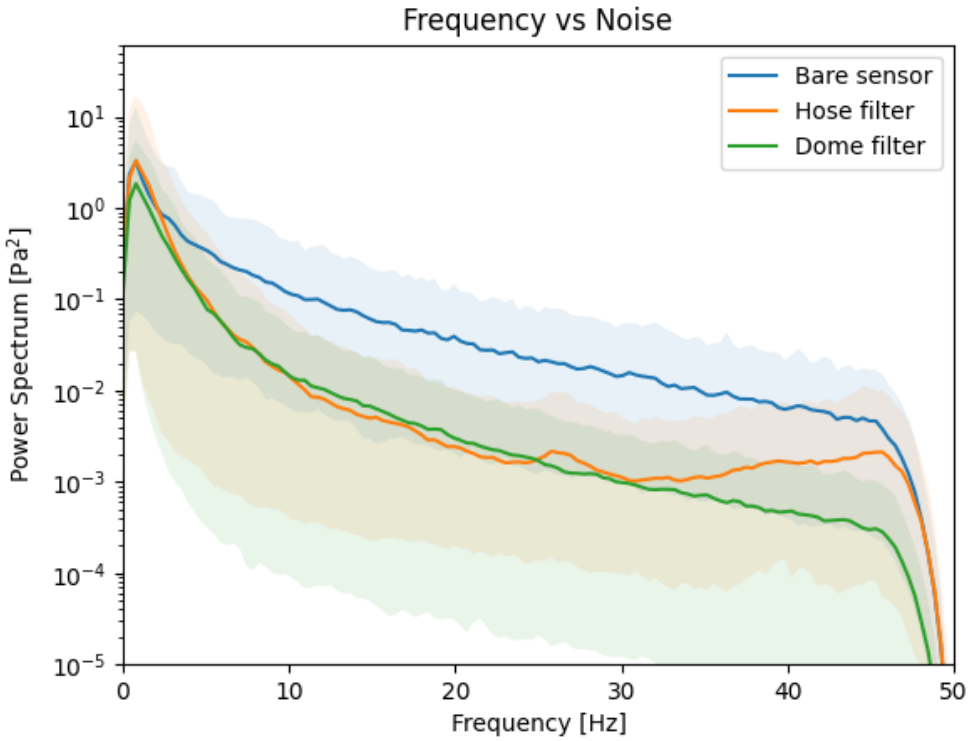


Figure 4.7: Comparison of the overall performance of each of the setups with $2 < V_{wind} < 3$ m s^{-1} coming from the south-west with 5% and 95% noise level ranges

As can be seen in figure 4.7, both filtering setups have similar performance with their average wind noise reduction capabilities for frequencies below 30 Hz. Nonetheless, the fabric dome filter is able to achieve a lower 5% noise level, indicating that in optimal conditions, this filter might be able to outperform the porous hose by some margin. At lower frequencies, both filters have a similar loss in their performance, both starting to reduce in their effectiveness around 10 Hz, and providing little to no reduction in the infrasound noise levels below approximately 2 Hz band. This lower end for the effectiveness of the fabric dome matches estimates based on theory, which would put the lower range of the filter at approximately 8 Hz (Assink, 2025).

For higher frequencies above 30 Hz, the porous hose filter seems to perform significantly worse than the fabric dome. This drop-off in the performance of the porous hose seen in this data matches the results observed in the signal impact analysis of section 4.2.1. As such, it

might be that in practical detections of signals in this frequency range, the hose would not perform as much worse as would be expected based on just the results seen in 4.7, as it is possible that the actual signal could achieve an amplification similar to that of the noise. Whether this is truly the case however might in turn depend on the underlying mechanism and the shape of the to be detected signal. If this amplification of noise in these higher frequencies relies on the formation of partially standing waves within the hose, the amplification achieved for a single infrasonic wave might be significantly lower as the standing wave would not have the opportunity to build up over time.

An additional interesting observation in this data, which can also be seen in the data for the wind speed analysis in figure 4.5b is that the porous hose shows a notable peak around the 26 Hz mark. Given a speed of sound in air at a temperature of 20° C of 343 [ms^{-1}], this yields that this peak occurs at a wavelength of 13.2 m. Given the length of the hose segment has a length of just over 3 m, that puts this peak right at the lowest possible frequency where a standing wave could form in this tube if it were closed off at one side, and open at the other end. This requirement of having an open end does not match with the actual design of the hose, which on one side is connected to the sensor through a small tube, and is closed off on the other end. Nonetheless, it is possible that at the sensor side the change in diameter or the sensor itself could lead to this side acting partially like an open end for the formation of standing waves within the porous hose. As such, it is possible that this frequency peak is in fact generated by a standing wave forming inside the tube. One observation which does undermine this theory is that in figure 4.5b it can be seen that as the wind speed increases, the frequency of the peak shifts slightly towards lower frequencies. If this peak were to be generated by a standing wave, it would be expected that the peak would remain at a rather more constant frequency.

Comparing the performance obtained for these filters with those found by Noble *et al.*, 2014, it can be found that for frequencies above 10 Hz, the performance of both filters is comparable to the performance of full hose rosettes and larger fabric domes respectively, each achieving reductions in the PSD between 10 and 15 dB. Where the limitations of the smaller filters do become more apparent is below this 10 Hz threshold. In the experiment performed by Noble *et al.*, 2014, the filters maintained remained effective at reducing the noise levels by at least 10 dB down to frequencies of 0.8 Hz. The smaller filters used in this experiment both dropped off in performance in this regime, with the reduction reaching 0 dB around $1-2$ Hz. An interesting observation that can be made is that, similar to the peak seen in the hose filter around 26 Hz in figure 4.7, the measurements by Noble *et al.*, 2014 show a similar peak around 11 Hz. Since the hose system used in that experiment used longer hose segments of 20 feet each, this peak could very well be caused by the same mechanism that caused the bump in the noise level for the single porous hose.

Overall, it can be concluded that including some form of a wind noise reduction filter can dramatically improve the noise characteristics of the sensor. The porous hose and fabric dome both achieved similar performance in reducing the noise characteristics of the wind measurements, significantly outperforming the bare sensor in noise levels over the entire spectrum. The fabric dome performed more consistently independent from the wind direction,

although the porous hose did achieve a higher level of noise filtering in some conditions. If space permits a dome filter would likely be a better choice for the detection of meteor generated infrasound since this filter does not introduce any significant artifacts into the measurements and remains effective into higher frequencies where the porous hose performance starts to drop off.

4.3. Network analysis

The first step in determining the requirements for a Dutch infrasound network to complement the Dutch component of the FRIPON network, also referred to as the DOERAK network, is to determine the goal of this network. If the goal of the network is to test the potential detection of meteors based purely on infrasound measurements, an important aspect of this sensor network is that the meteor entry needs to be detected by multiple infrasound stations. Such multiple detections could, similar to a multiple detection as is currently used for camera networks, help constrain properties of the meteor entry such as the entry trajectory and meteor energy. If on the other hand the goal of the network is to better understand the infrasound signature generated by meteors, the focus of the infrasound network would shift towards having comprehensive coverage of at least a single sensor for a large area rather than achieving multiple detections. Alternatively, if the focus is on improving the understanding of the relation between shocks in different shock angles relative to the entry, a middle ground needs to be found where the network is wide enough to provide coverage over multiple angles, but dense enough as to allow for multiple detections of the same entry.

For the layout of an infrasound network complementary to the existing DOERAK system, current plans are to create an initial network consisting of three infrasound stations spread over the Netherlands. Due to operational constraints, the locations for two of these stations have been established as a residential location in Tilburg and at Delft University of Technology. Although the latter of these locations hasn't been fully set in stone yet, for purposes of this investigation it will be treated as a given. The third station could be placed at one of five candidate locations throughout the Netherlands. These locations have been marked in figure 4.8 with blue markers, with the fixed station locations marked with red markers.

To determine the estimated usefulness of placing the infrasound sensor at any of these locations, an estimation of likely future meteor entries has to be made. Since it is difficult to predict where such an entry event will take place, it is assumed that the future entry events will follow a similar pattern as historical events in the region. As such, the meteor entry database described in section 3.1.1 has been filtered to only those entries in which a DOERAK camera participated in the detection of the fireball. This constraint should provide a decent estimate of future data availability, since only those events where a Dutch camera was contained will be easily accessible by Dutch researchers which might want to make use of this network. The final fireballs included in this dataset are visible in figure 4.8 as red lines, with a semi-transparent ring at the trailing end of the meteor, for a total of 40 historical fireballs. Although the actual detections of the network in the future might change due to the recent addition of cameras especially in the northern parts of the Netherlands, the actual impact of this will be difficult to

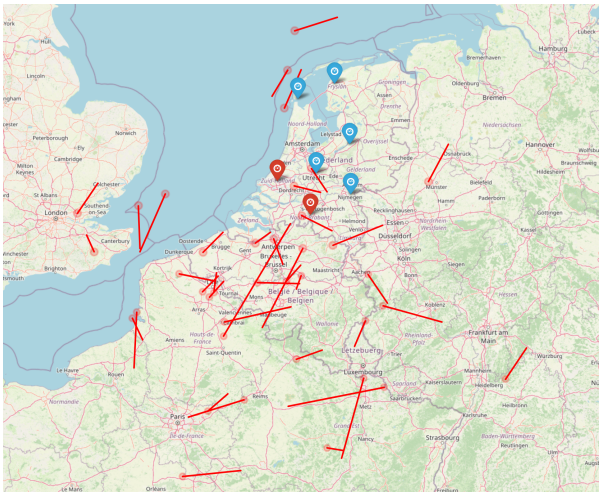


Figure 4.8: Meteor entries in which a Dutch FRIPON station took part in the detection with the FRIPON network, with marked locations of potential infrasound stations

predict since multiple detections also rely on the network density surrounding the camera.

For an indication of the potential future detection rate above the Netherlands, it is possible to look at a previous study performed on the efficiency of the FRIPON network (Colas *et al.*, 2020). In this study, Colas *et al.*, 2020 found an estimated detectable meteor flux for meteors >1 cm in size of 1250e-6 yr^{-1} km^{-2}. Given the surface area of the Netherlands of approximately 41.5e3 km^2, this would yield an estimated meteor flux of around 52 meteors flying overhead each year. Given the time span covered in this investigation is approximately 5 years, that would indicate that during this period, this part of the network has been operating at $< 20\%$ coverage. Additionally, during the investigation by Colas *et al.*, 2020, an additional border of 120 km was added around the outside boundaries of the relevant area due to meteors passing close by the relevant area also potentially being detectable by the network. If this correction is applied, the network efficiency drops significantly lower still, reach only around 10%. Here it should however be noted that, being on the Northern edge of the FRIPON network, the efficiency of the Northernmost stations will likely remain somewhat below the theoretical maximum since a meteor will need a multiple-detection to be positively identified, which is statistically less likely to occur if the station is on the edge of the network. Nonetheless, it can be concluded that despite best efforts, this analysis of potential infrasound station performance is rudamentary at best.

Figure 4.9 and figure 4.10 contain the amount of meteor entries grouped by the total number of detections for each of the candidate locations for a maximum detection range of 200 km and 300 km respectively. As can be seen from this data, at a maximum detection range of 300 km, any of the potential locations for a third infrasound station provide similar coverage percentages with the only exception being Franeker due to its far northern location. With the new cameras coming online in the northern part of the Netherlands, it can however be expected that this lower detection count will be partially offset as more detections might occur in this area. Still, the camera density in this area remains significantly lower than that along the southern edge of the Netherlands, thus likely maintaining a lower effectiveness than an infrasound station along

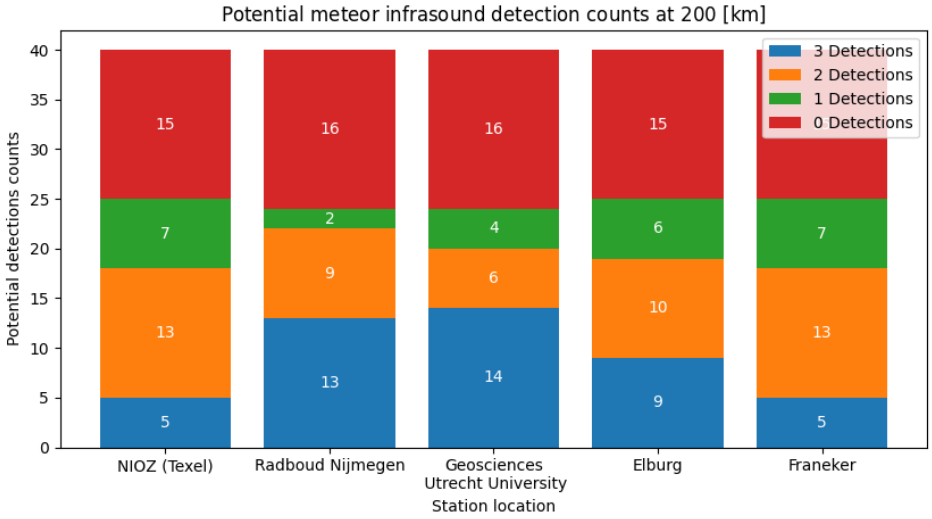


Figure 4.9: Potential detection counts for different infrasound locations at a maximum detection range of 200 km

this southern edge.

If the effective detection range is reduced to just 200 km to match the stricter definition for short range propagation given by Edwards, 2009, the trend of a lower efficiency extends south to also affect the NIOZ (Texel) and Elburg candidate locations. In any of these cases however, the impact of putting the camera at a Northern station is only relevant if higher detection counts are necessary. If a detection count of 2 is enough for the research task at hand, the location of the third station is relatively irrelevant since both the Delft and Tilburg locations are at a similar distance from the bulk of meteor entries which are centered over Belgium.

Table 4.3: Classification of infrasound detection categories at a maximum detection range of 200 km

Third station location	Bow + ballistic	Known + transient	Ballistic only	Bow only	Other	None
Confirmed stations only	1	5	11	7	0	16
NIOZ (Texel)	3	4	11	6	1	15
Radboud Nijmegen	2	5	11	6	0	16
Geosciences Utrecht University	1	6	11	6	0	16
Elburg	4	4	12	5	0	15
Franeker	3	5	11	6	0	15

If the main research to be answered by the infrasound network is related to the correlation of the different shock regimes, an important criterium is that the network must contain an infrasound station in both the ballistic shock regime of the meteor and in the bow shock region. Tables 4.3 and 4.4 show how many of the meteor entries provide insight into each of the different shock regimes defined by Edwards *et al.*, 2008. As can be seen from these table, adding a third infrasound station relatively close to the existing stations, being either at Utrecht University or at Radboud University in Nijmegen does not increase the amount of cases where

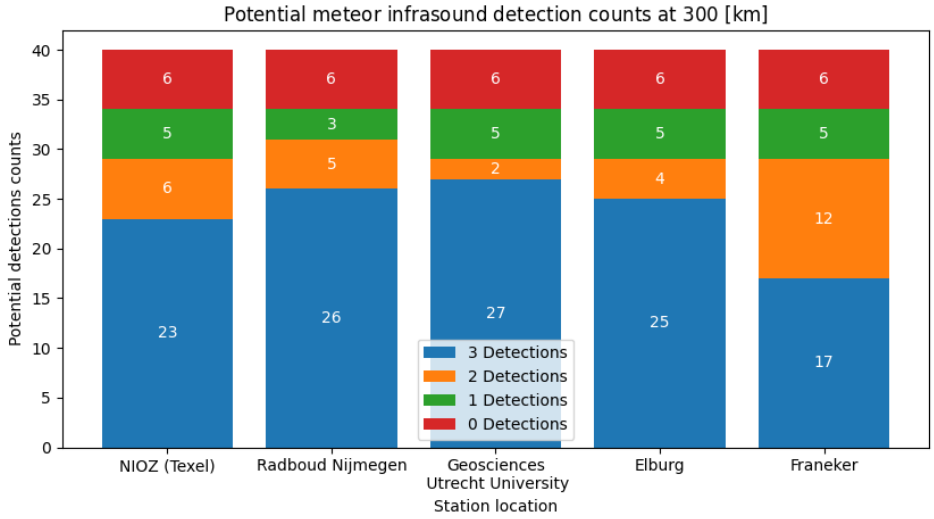


Figure 4.10: Potential detection counts for different infrasound locations at a maximum detection range of 300 km

Table 4.4: Classification of infrasound detection categories at a maximum detection range of 300 km

Third station location	Bow + ballistic	Known + transient	Ballistic only	Bow only	Other	None
Confirmed stations only	1	9	17	7	0	6
NIOZ (Texel)	4	8	16	6	0	6
Radboud Nijmegen	2	9	17	6	0	6
Geosciences Utrecht University	1	9	17	7	0	6
Elburg	4	8	17	5	0	6
Franeker	4	8	16	6	0	6

multiple shock regimes are measured. Since figures 4.9 and 4.10 do show a significant amount of triple-detections for these locations, it must be concluded that these stations would typically measure the same regimes as measured by the two confirmed locations. As such, placing a third station at either of these locations might be useful if the research focus is on correlation of individual signals to a common source, but not when the focus of the research is investigating the shape of the shock in the different regimes.

The potential locations at the NIOZ institute on Texel or the location in Franeker are likely to provide more detections with different shock regimes, but the usefulness of these stations for this purpose is reduced if the detectability of meteor entries does not reach the 300 km limit set for the short range infrasound propagation assumptions. If it proves to only be feasible to make proper detections to approximately 200 km, the additional distance will likely primarily lead to more single-detections at this station rather than being correlated with the detections at Delft University or Tilburg. The potential location in Elburg seems to provide the most reasonable midpoint out of all available options. The Elburg location would provide a similar level of multi-regime shock detections, while also being close enough to maintain a high level of multiple detections.

5

Conclusion and Recommendations

In this report, it has been investigated how an infrasound sensor network could be set up to improve the detection capabilities of meteor detection networks. For this purpose, three aspects of such infrasound detection networks have been investigated to answer the sub-questions derived in chapter 1. This chapter will first answer these research questionsand link these outcomes back to the main research question in section 5.1. After that, section 5.2 will give recommendations for future research.

5.1. Research questions

This section will answer each of the research questions posed in chapter 1.

How can meteor generated infrasound signature be automatically detected from infrasound sensors?

For the detection of meteor generated infrasound signatures in data collected by the Raspberry Shake FSDN network, a pipeline was developed to perform this in an orderly function. Actually identifying a meaningful infrasound signature in this data proved to be more difficult. Using a dynamic threshold detection algorithm on the power spectra measured by the sensor, the estimated successful detection rates lie between 3% - 4% of the sound traces used in the analysis. If a static threshold factor is used, the number of successful detections drops closer to 1% for the optimal fixed threshold, and even lower if an incorrect threshold factor is chosen. Additionally, the approach had a false detection rate similar to the successful detection rate around the same 3% - 4% region, which makes it difficult to conclusively state which of the detections were indeed generated by meteors and which were caused by other sources.

The biggest limiting factors for improving the detection are the quality of the infrasound data and the uncertainty regarding the setup and environment of the infrasound sensors within this network. Since the environmental conditions are unknown, it isn't possible to adjust the detections thresholds to the noise level. Attempts to make the algorithm automatically adjust to the variance in the signal by setting the threshold relative to the standard deviation failed to

provide any improvements in the detection, and in fact seem to slightly reduce the detection performance of the algorithm.

Based on these results, although this does not mean that the algorithm is necessarily a bad fit for the problem at hand, this investigation does show that public infrasound data is unlikely to be useful in detecting the smaller meteor events which networks like FRIPON focus on.

How can the noise characteristics for infrasound sensors be optimised with a reasonable sensor footprint?

For the reduction of the wind noise in the infrasound data measured with a Raspberry Shake infrasound sensor, two different filtering approaches have been tested. The first filter consisted of a porous hose with a length of 3 m which was connected to the measurement port on the infrasound sensor. The second filter was a fabric dome with a diameter of 25 cm placed over the entire filter assembly. Both of these filtered setups, as well as a baseline setup without any wind noise filtering have been deployed at the Rooftop Lab of the Aerospace Engineering faculty at Delft University of Technology. Additionally, the impact of each of the filters on the detectability of a coherent signal was analysed in a controlled indoor test.

In the signal impact analysis, it was found that for low frequencies, neither filter significantly impacted the response of the sensor to a coherent signal, detecting similar signal levels both with and without the filter. For higher frequencies exceeding a frequency of 30 Hz, the porous hose sensor actually showed an amplified response, indicating that at these frequencies there might be additional constructive interference in the porous hose filter. The fabric dome did not show any such artifacts, maintaining a consistent response at each of the tested frequencies.

The analysis of the noise performance on the rooftop lab showed that both filters achieved similar performance, reducing the noise levels compared to a bare sensor by between 10 and 15 dB for higher frequencies (above $8 - 10$ Hz) matching larger filters. For lower frequencies, the limited size of each of the filters became more apparent, with their performance dropping to close to 0 dB around $1 - 2$ Hz.

The effect of wind speed on the filtering performance of either filter was also fairly similar, each showing a noise level increase of 20 dB if the wind speeds increase from $0 - 1$ m s^{-1} to the $4 - 5$ m s^{-1} range. The impact of the wind direction showed more difference, with the single porous hose being significantly ore susceptible to changes in wind direction to its filteriing performance.

A significant downside of the porous hose remains that it significantly alters the response of the filter at frequencies above 30 Hz. Similar to the distortion seen in the signal impact analysis, the noise analysis shows the performance of the porous hose filter significantly reducing for frequencies above this range. Additionally, the porous hose shows an extra noise peak around the 26 Hz range, which could be caused by a standing wave forming inside the porous hose.

Since both filters perform similarly in the low frequency range (below 30 Hz) and the generally better performance of the fabric dome at higher frequencies as well as not introducing artifacts into potential infrasound signals, the fabric dome is considered to be the best candidate filter out of the ones tested. The additional advantages, having a more predictable response to different wind directions, counts as an additional benefit of this filter, as this could make it easier

to adapt a detection algorithm based on the environmental properties.

What would an effective integrated infrasound sensor network need to look like to achieve a high coverage?
For a Dutch infrasound network consisting of three infrasound sensors, a multitude of potential sensor locations were considered. Based on existing plans for a Dutch infrasound network, two of the sensors were assumed to be placed at Delft University of Technology and a residential building in Tilburg. For the placement of the third infrasound sensor, the optimal placement depends on the goal of the infrasound network.

If the goal of the combined DOERAK + infrasound network is to determine entry information purely based on infrasound signatures, either Radboud University Nijmegen or Utrecht University would provide the most effective placement. This location for a third sensor would maximise the chance that all three sensors would be able to detect the meteor signature, with a relatively high likelihood that all detection will fall within the same part of the shock domain.

If the goal of the infrasound network is to achieve the highest number of meteor entries with at least one infrasound detection, the optimal location for the third station would be Franeker. Although historic data shows relatively little difference between station locations for achieving the highest potential coverage, given the recently installed DOERAK cameras in the northern part of the Netherlands, it can be anticipated that the meteor frequency in this area will increase, thereby giving this station an edge over the other locations.

If the goal of the infrasound network is a hybrid between these two, or the goal is to perform more research on the behaviour of the shock wave in different parts of the shock regime, the optimal placement would be the potential location in Elburg. The distance of this station relative to the fixed stations increases the likelihood that this station will be in a different part of the shock regime, while still being close enough as to achieve a high number of correlated infrasound detections with these two stations. Additionally, its somewhat northern location would allow it to potentially also detect infrasound signals from meteors detected by the expanded northern DOERAK cameras.

How can the detection capabilities of meteor detection networks be improved through infrasound infrasound measurements.
Based on the answers to the sub-questions above, it is now possible to answer the main research question of this investigation. Given the current state of meteor signature detections, it is unlikely that infrasound measurements will be able to conclusively detect meteors on their own. Although the prospects might improve if the sensors are part of a more controlled network with better noise filtering, the extent of this improvement is as of yet unknown.

For optimal infrasound data, the sensors in an infrasound network will need to be equipped with some form of wind noise filter. The optimal filter for this use case would be a dome filter, since this type of filter provides the most reliable reduction in environmental noise while leaving the system untouched. For the frequency ranges in which the Raspberry Shake infrasound sensor operates, the fabric dome with a diameter of 25 cm provided a good response for the majority of the sensitive range of the sensor. As such, a relatively small sensor might already

be enough to achieve a low enough noise in the frequencies of interest.

If a three sensor network is to be introduced into the DOERAK fireball detection network, the most versatile option out of the available options would consist of an infrasound sensor at the Delft University of Technology, a sensor at a residential location in Tilburg, and a sensor in Elburg. This layout provides the widest range of possibilities for future research.

When such a network is rolled out, it will likely still take a significant amount of time before the sensors can truly be used for aiding in the detection of meteors. The fine-tuning of the detection parameters would first require a dataset of known meteor entries and their infrasound signatures to determine if the noise level reduction achieved with the infrasound sensors is low enough for reliable detection, and if potentially more advanced detection algorithms can be developed to distinguish better between false positives and true positives.

5.2. Recommendations

For future research into the combination of infrasound detections with camera based fireball detection networks, it will be vital to be able to more closely correlate visible detections with the infrasound signatures. With the arrival window approach used in this report, the final arrival windows commonly had durations close to or exceeding a full minute. Having such wide windows makes it harder to conclusively proof whether a signal came from the meteor event in question, or from an unrelated source. If the window can be shortened, this could aid in further investigations into the effects of the meteor properties on the infrasound signals' shape. For this purpose, it could be useful to include higher fidelity atmospheric models of the time of interest, including live ambient temperatures and wind speed and direction. For this purpose, it might be possible to adapt tools such as SUPRACENTER (Edwards & Hildebrand, 2004) to help provide more accurate arrival times. This would in turn also help to finetune potential detection algorithms.

If more data is available, it could also be interesting to look at the potential of more advanced detection algorithms. Although this report mainly investigated the use of simple threshold models, it might be interesting to look at algorithms which look at e.g. the phase of different frequency components in the infrasound signature to more accurately detect meteor infrasound. Such models would however rely on higher quality measurements being available first. As such, the main recommendation for future research is to first look at generating a larger dataset of known meteor entries by deploying an infrasound sensor alongside an existing camera network for an extended period of time.

Before future research starts investigating fully independent meteor detections, it might also be interesting to see if single-station detections of the FRIPON network could include any potential meteor entries. These single detections, while not providing ful trajectory information, could still serve as a trigger which could help provide a starting point for unknown meteor detections to take place.

References

Assink, J. (2025, September). Direct correspondence.

Beyer, R. T. (1974). *Nonlinear acoustics*. US Government Printing Office.

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., & Wassermann, J. (2010). Obspy: A python toolbox for seismology. *Seismological Research Letters*, *81*, 530–533. https://doi.org/10.1785/gssrl.81.3.530

Borovička, J., Popova, O., & Spurný, P. (2019). The maribo cm2 meteorite fall—survival of weak material at high entry speed. *Meteoritics & Planetary Science*, *54*(5), 1024–1041. https://doi.org/https://doi.org/10.1111/maps.13259

Bowman, J. R., Baker, G. E., & Bahavar, M. (2005). Ambient infrasound noise. *Geophysical Research Letters*, *32*(9). https://doi.org/https://doi.org/10.1029/2005GL022486

Brown, P., Weryk, R., Kohut, S., Edwards, W., & Krzeminski, Z. (2010). Development of an all-sky video meteor network in southern ontario, canada the asgard system. *WGN, Journal of the International Meteor Organization, vol. 38, no. 1, p. 25-30*, *38*, 25–30.

Brykina, I., & Tirskiy, G. (2017). Mass loss and light curve of a large meteoroid. analytical solution. *Journal of Applied Mathematics and Mechanics*, *81*(5), 395–408. https://doi.org/https://doi.org/10.1016/j.jappmathmech.2018.03.008

Ceplecha, Z. (1996). Luminous efficiency based on photographic observations of the lost city fireball and implications for the influx of interplanetary bodies onto earth. *Astronomy and Astrophysics, v. 311, p. 329-332*, *311*, 329–332.

Ceplecha, Z., Borovička, J., Elford, W. G., ReVelle, D. O., Hawkes, R. L., Porubčan, V., & Šimek, M. (1998). Meteor phenomena and bodies. *Space Science Reviews*, *84*(3), 327–471. https://doi.org/10.1023/A:1005069928850

Christie, D. R., & Campus, P. (2009). The ims infrasound network: Design and establishment of infrasound stations. In A. Le Pichon, E. Blanc, & A. Hauchecorne (Eds.), *Infrasound monitoring for atmospheric studies* (pp. 29–75). Springer Netherlands. https://doi.org/10.1007/978-1-4020-9508-5_2

Colas, F., Zanda, B., Bouley, S., Jeanne, S., Malgoyre, A., Birlan, M., Blanpain, C., Gattacceca, J., Jorda, L., Lecubin, J., et al. (2020). Fripon: A worldwide network to track incoming meteoroids. *Astronomy & Astrophysics*, *644*, A53.

Drob, D. P., Picone, J. M., & Garcés, M. (2003). Global morphology of infrasound propagation. *Journal of Geophysical Research: Atmospheres*, *108*(D21). https://doi.org/https://doi.org/10.1029/2002JD003307

Edwards, W. N., & Hildebrand, A. R. (2004). Supracenter: Locating fireball terminal bursts in the atmosphere using seismic arrivals. *Meteoritics & Planetary Science*, *39*(9), 1449–1460. https://doi.org/https://doi.org/10.1111/j.1945-5100.2004.tb00121.x

Edwards, W. N. (2009). Meteor generated infrasound: Theory and observation. In A. Le Pichon, E. Blanc, & A. Hauchecorne (Eds.), *Infrasound monitoring for atmospheric studies* (pp. 361–414). Springer Netherlands. https://doi.org/10.1007/978-1-4020-9508-5_12

Edwards, W. N., Brown, P. G., Weryk, R. J., & ReVelle, D. O. (2008). Infrasonic observations of meteoroids: Preliminary results from a coordinated optical-radar-infrasound observing campaign. In J. M. Trigo-Rodríguez, F. J. M. Rietmeijer, J. Llorca, & D. Janches (Eds.), *Advances in meteoroid and meteor science* (pp. 221–229). Springer New York. https://doi.org/10.1007/978-0-387-78419-9_31

ElGabry, M., Korrat, I., Hussein, H., & and, I. H. (2017). Infrasound detection of meteors. *NRIAG Journal of Astronomy and Geophysics*, *6*(1), 68–80. https://doi.org/10.1016/j.nrjag.2017.04.004

Ens, T., Brown, P., Edwards, W., & Silber, E. (2012). Infrasound production by bolides: A global statistical study. *Journal of Atmospheric and Solar-Terrestrial Physics*, *80*, 208–229. https://doi.org/https://doi.org/10.1016/j.jastp.2012.01.018

Grangeon, J., & Lesage, P. (2019). A robust, low-cost and well-calibrated infrasound sensor for volcano monitoring. *Journal of Volcanology and Geothermal Research*, *387*, 106668. https://doi.org/https://doi.org/10.1016/j.jvolgeores.2019.106668

Hajdukova, M., Sterken, V., Wiegert, P., & Kornoš, L. (2020). The challenge of identifying interstellar meteors. *Planetary and Space Science*, *192*, 105060. https://doi.org/https://doi.org/10.1016/j.pss.2020.105060

Hankey, M., Molau, S., & Perlerin, V. (2023). Allsky7 history, status and updates. *Proceedings of the International Meteor Conference, 2022*, 133–135.

Haynes, C. P., & Millet, C. (2013). A sensitivity analysis of meteoric infrasound. *Journal of Geophysical Research: Planets*, *118*(10), 2073–2082. https://doi.org/https://doi.org/10.1002/jgre.20116

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Leventhall, G. (2007). What is infrasound? [Effects of ultrasound and infrasound relevant to human health]. *Progress in Biophysics and Molecular Biology*, *93*(1), 130–137. https://doi.org/https://doi.org/10.1016/j.pbiomolbio.2006.07.006

Loehle, S., Vaubaillon, J., Matlovič, P., & Tóth, J. (2024). Meteorite material luminous efficiencies from ground testing of meteoroid entry. *Icarus*, *407*, 115817.

McLean, R., Alsop, S., & Fleming, J. (2005). Nyquist—overcoming the limitations. *Journal of Sound and Vibration*, *280*(1), 1–20. https://doi.org/https://doi.org/10.1016/j.jsv.2003.11.047

Molau, S., & Rendtel, J. (2009). A comprehensive list of meteor showers obtained from 10 years of observations with the imo video meteor network. *WGN, Journal of the International Meteor Organization*, *37*.

National Oceanic and Atmosperic Administration, National Aeronautics and Space Administration, & United States Air Force. (1976). *Us standard atmosphere, 1976* (Vol. 76). National Oceanic; Atmospheric Administration.

Noble, J. M., Alberts, W. K., Raspet, R., Collier, S. L., & Coleman, M. A. (2014). Infrasound wind noise reduction via porous fabric domes. *The Journal of the Acoustical Society of America*, *135*(4), 2409–2409.

OBERST, J., MOLAU, S., HEINLEIN, D., GRITZNER, C., SCHINDLER, M., SPURNY, P., CEPLECHA, Z., RENDTEL, J., & BETLEM, H. (1998). The “european fireball network”: Current status and future prospects. *Meteoritics & Planetary Science*, *33*(1), 49–56. https://doi.org/https://doi.org/10.1111/j.1945-5100.1998.tb01606.x

Rain Bird. (2025). Rain bird drip hose system.

Raspberry Shake, S.A. (2016). Raspberry shake. https://doi.org/10.7914/SN/AM

ReVelle, D. O. (1976). On meteor-generated infrasound. *Journal of Geophysical Research (1896-1977)*, *81*(7), 1217–1230. https://doi.org/https://doi.org/10.1029/JA081i007p01217

Robinson, I., Robinson, S., & Robinson, N. (2020). Novel infrasound monitor project: Real geophysics research on a budget. *Physics Education*, *55*(5), 055025.

Silber, E. A., Brown, E., Thompson, A. R., & Sawal, V. (2025). A curated dataset of regional meteor events with simultaneous optical and infrasound observations (2006–2011). *Data*, *10*(9). https://doi.org/10.3390/data10090138

Silber, E. A., & Brown, P. G. (2014). Optical observations of meteors generating infrasound—i: Acoustic signal identification and phenomenology. *Journal of Atmospheric and Solar-Terrestrial Physics*, *119*, 116–128. https://doi.org/https://doi.org/10.1016/j.jastp.2014.07.005

Silber, E. A., Brown, P. G., & Krzeminski, Z. (2015). Optical observations of meteors generating infrasound: Weak shock theory and validation. *Journal of Geophysical Research: Planets*, *120*(3), 413–428. https://doi.org/https://doi.org/10.1002/2014JE004680

Spalding, R., Tencer, J., Sweatt, W., Conley, B., Hogan, R., Boslough, M., Gonzales, G., & Spurný, P. (2017). Photoacoustic sounds from meteors. *Scientific Reports*, *7*(1), 41251. https://doi.org/10.1038/srep41251

Subasinghe, D., & Campbell-Brown, M. (2018). Luminous efficiency estimates of meteors. ii. application to canadian automated meteor observatory meteor events. *The Astronomical Journal*, *155*(2), 88.

Subasinghe, D., Campbell-Brown, M. D., & Stokan, E. (2016). Physical characteristics of faint meteors by light curve and high-resolution observations, and the implications for parent bodies. *Monthly Notices of the Royal Astronomical Society*, *457*(2), 1289–1298. https://doi.org/10.1093/mnras/stw019

Vida, D., Šegon, D., Gural, P. S., Brown, P. G., McIntyre, M. J. M., Dijkema, T. J., Pavletić, L., Kukić, P., Mazur, M. J., Eschman, P., Roggemans, P., Merlak, A., & Zubović, D. (2021). The global meteor network – methodology and first results. *Monthly Notices of the Royal Astronomical Society*, *506*(4), 5046–5074. https://doi.org/10.1093/mnras/stab2008

Walker, K. T., & Hedlin, M. A. (2009). A review of wind-noise reduction methodologies. In A. Le Pichon, E. Blanc, & A. Hauchecorne (Eds.), *Infrasound monitoring for atmospheric*

studies (pp. 141–182). Springer Netherlands. https://doi.org/10.1007/978-1-4020-9508-5_5

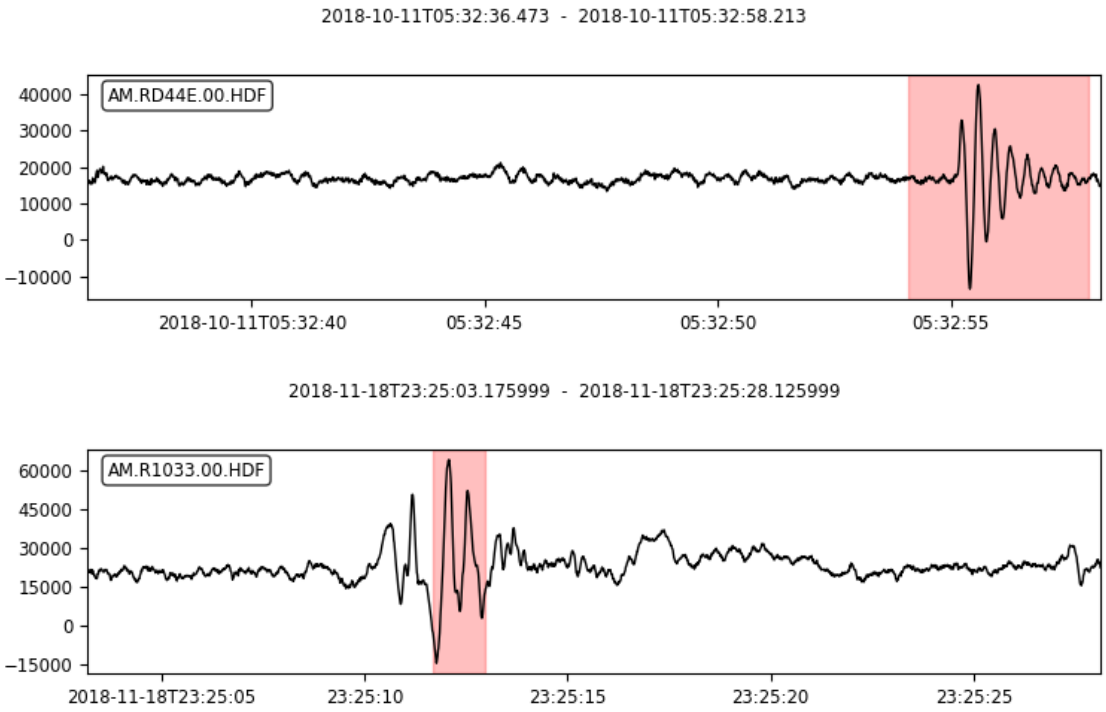
Woodward, R., Israelsson, H., Bondár, I., McLaughlin, K., Bowman, J. R., & Bass, H. (2005). Understanding wind-generated infrasound noise. *Proceedings of the 27th Seismic Research Review: Ground-Based Nuclear Explosion Monitoring Technologies*, 866–875.

Zhang, F., Liu, D., Liu, A., Gang, X., & Li, L. (2020). Theoretical investigation on the infrasonic frequency response of measurement microphones in different venting states. *Measurement*, *162*, 107905. https://doi.org/https://doi.org/10.1016/j.measurement.2020.107905

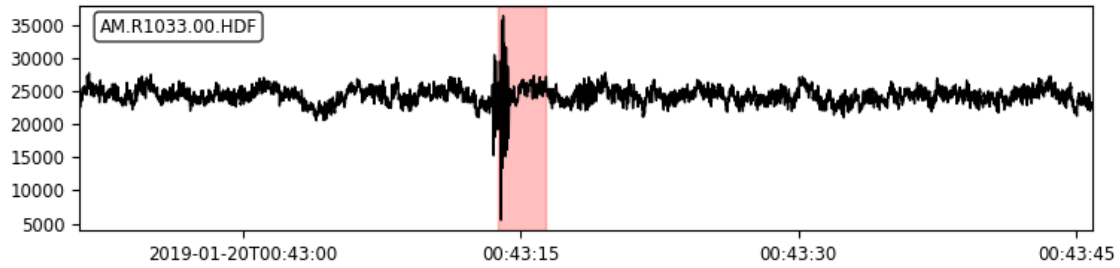
A

Detection peaks

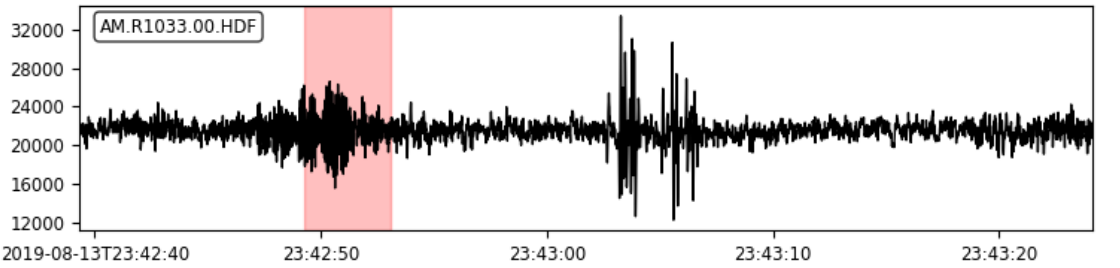
Figure A.1 shows the peaks detected by the automatic detection algorithm explained in section 3.1, using a dynamic threshold relative to the base noise level.



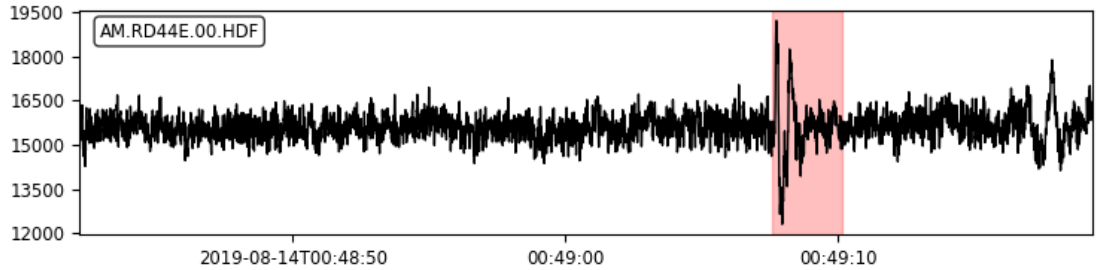
2019-01-20T00:42:51.145999 - 2019-01-20T00:43:45.865999



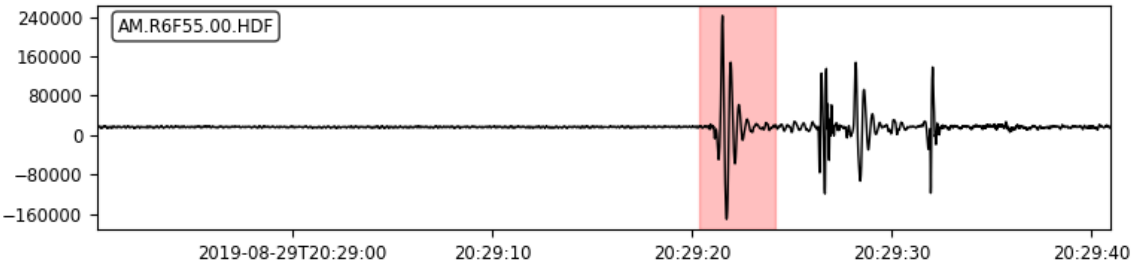
2019-08-13T23:42:39.317999 - 2019-08-13T23:43:24.117999



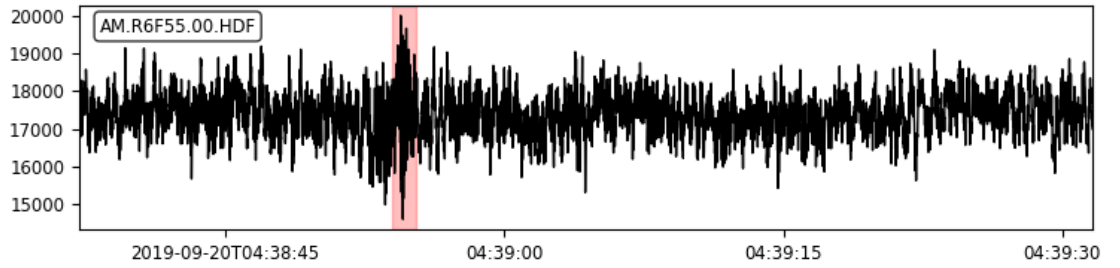
2019-08-14T00:48:42.157 - 2019-08-14T00:49:19.327



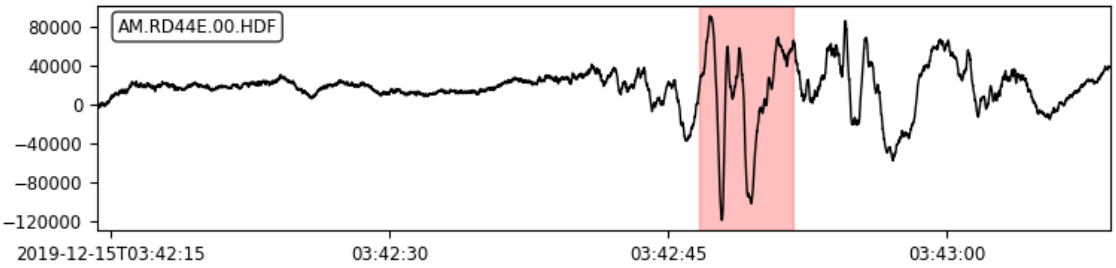
2019-08-29T20:28:50.207 - 2019-08-29T20:29:40.967



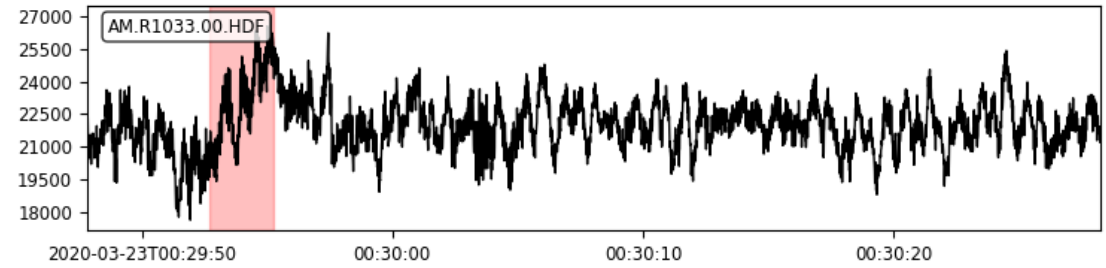
2019-09-20T04:38:37.157 - 2019-09-20T04:39:31.617



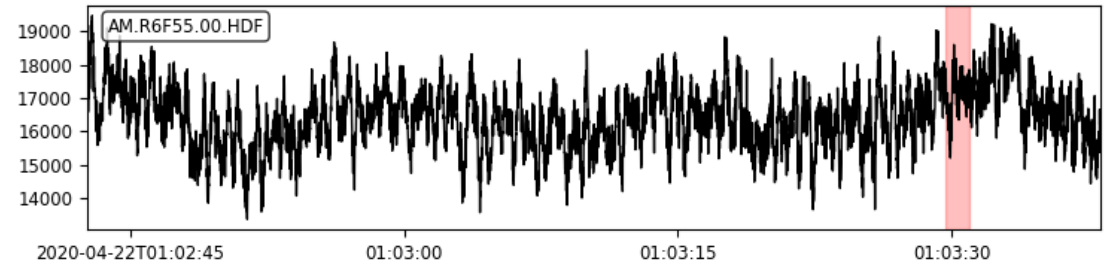
2019-12-15T03:42:14.248999 - 2019-12-15T03:43:08.808999



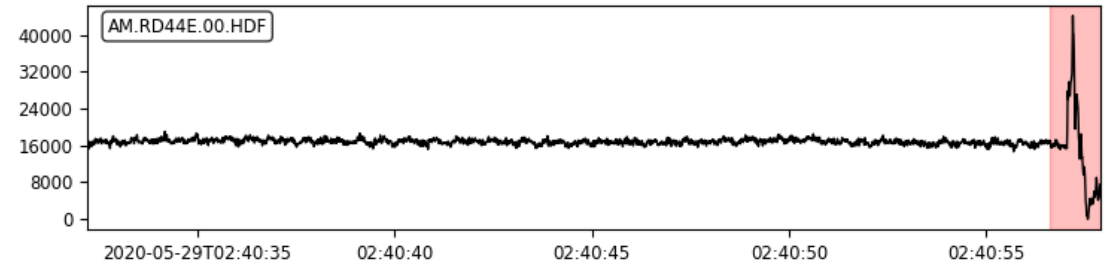
2020-03-23T00:29:47.830999 - 2020-03-23T00:30:28.230999



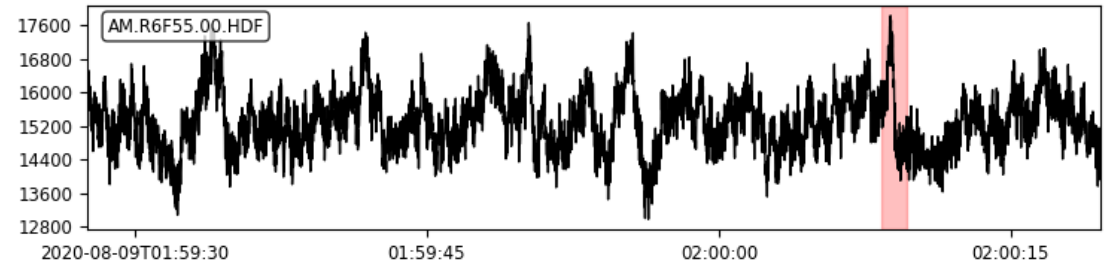
2020-04-22T01:02:42.629 - 2020-04-22T01:03:38.169



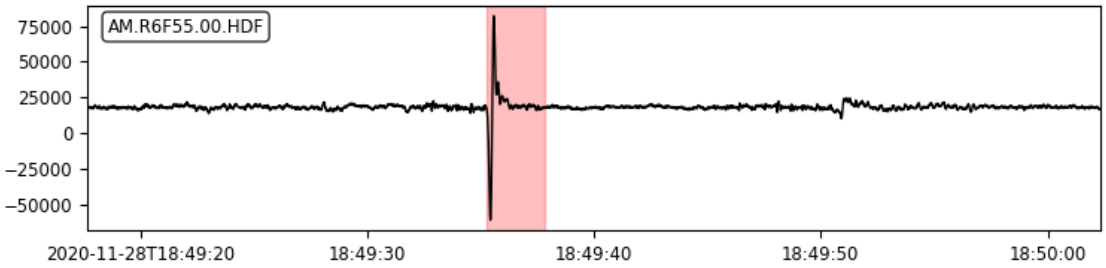
2020-05-29T02:40:32.209999 - 2020-05-29T02:40:57.899999



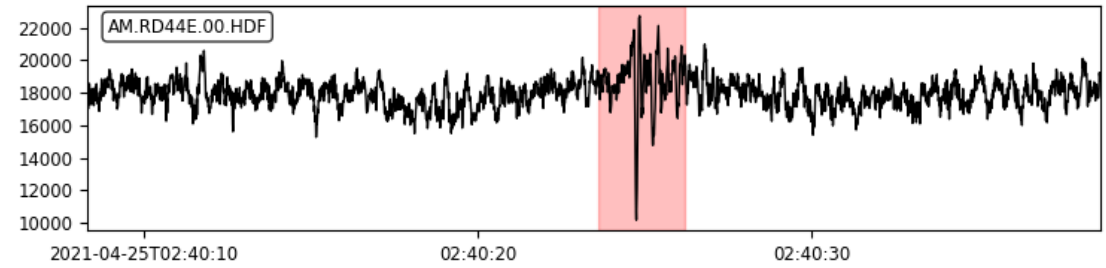
2020-08-09T01:59:27.548999 - 2020-08-09T02:00:19.598999



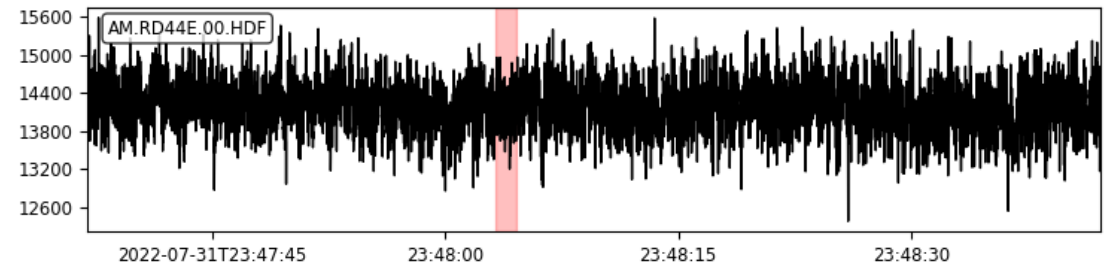
2020-11-28T18:49:17.659 - 2020-11-28T18:50:02.329



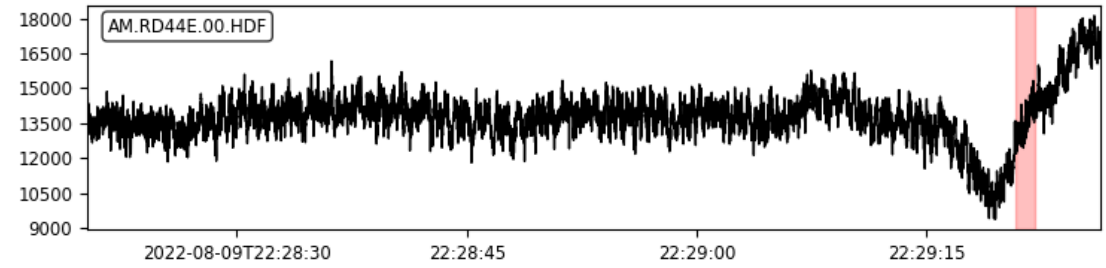
2021-04-25T02:40:08.295999 - 2021-04-25T02:40:38.665999



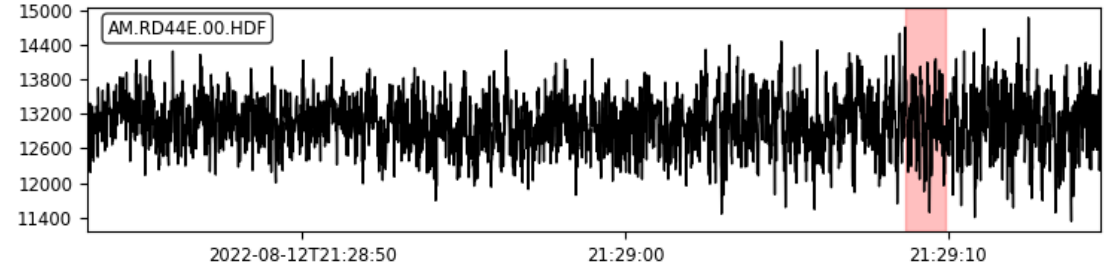
2022-07-31T23:47:36.946999 - 2022-07-31T23:48:42.176999



2022-08-09T22:28:20.236999 - 2022-08-09T22:29:26.326999



2022-08-12T21:28:43.346999 - 2022-08-12T21:29:14.716999



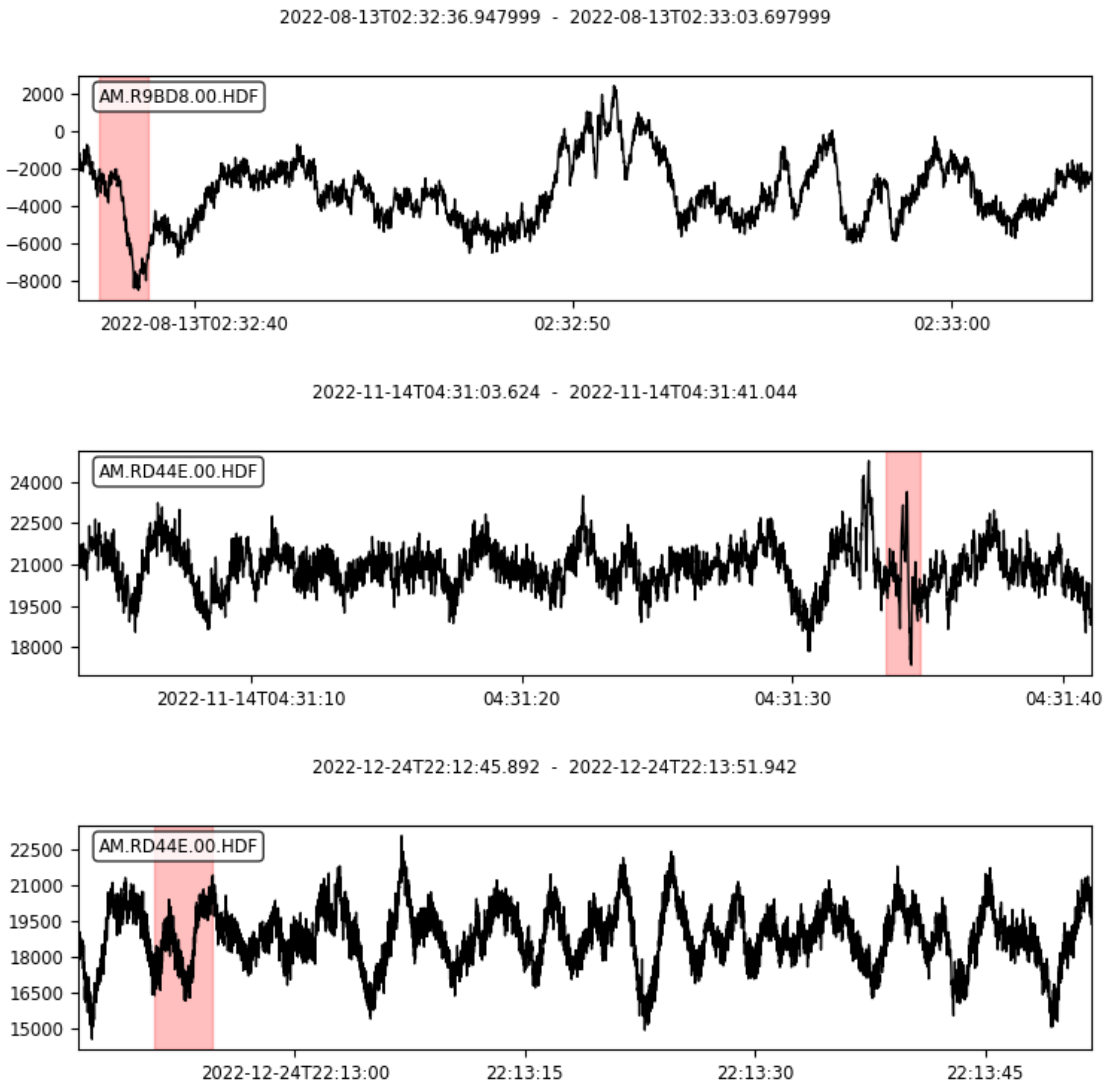


Figure A.1: Infrasound peak detections using a dynamic threshold relative to the base noise level

B

Measurement data

Figures B.1 through figure B.3 show an estimate of the standard deviation of the three wind components during the bare sensor, porous hose, and fabric dome test runs respectively when plotted against the wind strength. The u and v components are the horizontal components of the wind measurements, and w is the vertical component of the wind. The average wind and standard deviation values are computed over time stretches of 10 minutes each.

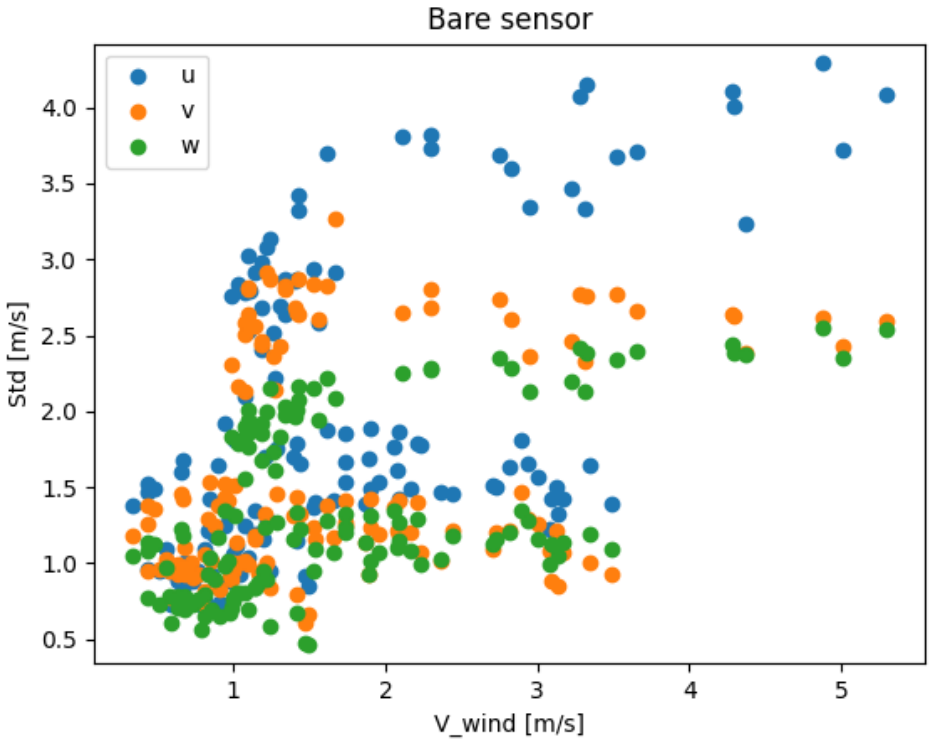


Figure B.1: Wind component standard deviation plotted against the average wind strength for the bare sensor.

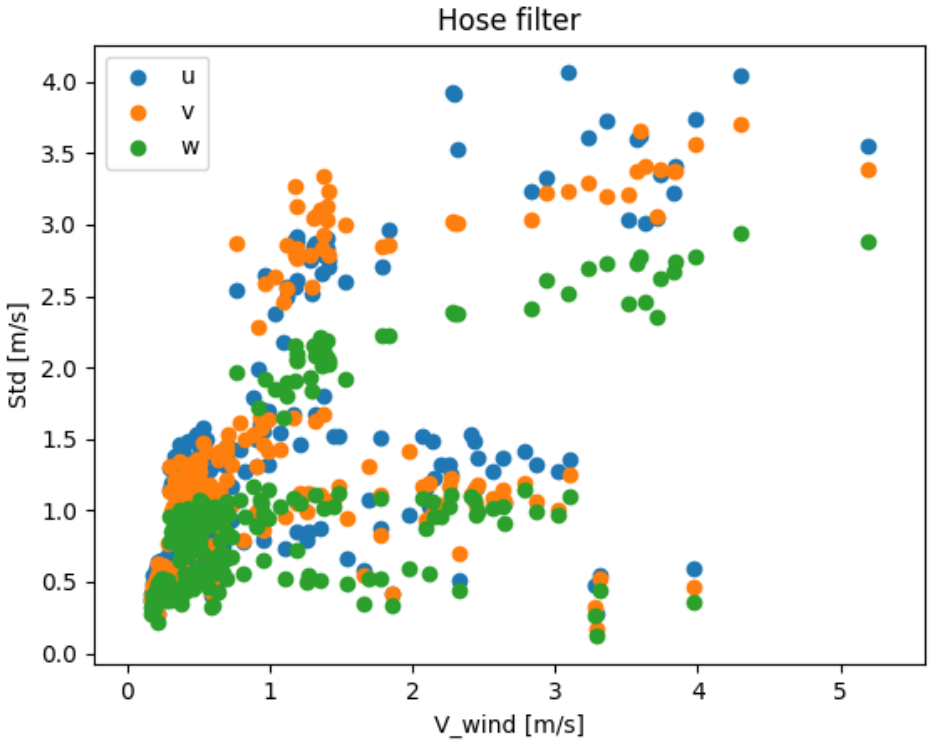


Figure B.2: Wind component standard deviation plotted against the average wind strength for the porous hose filter.

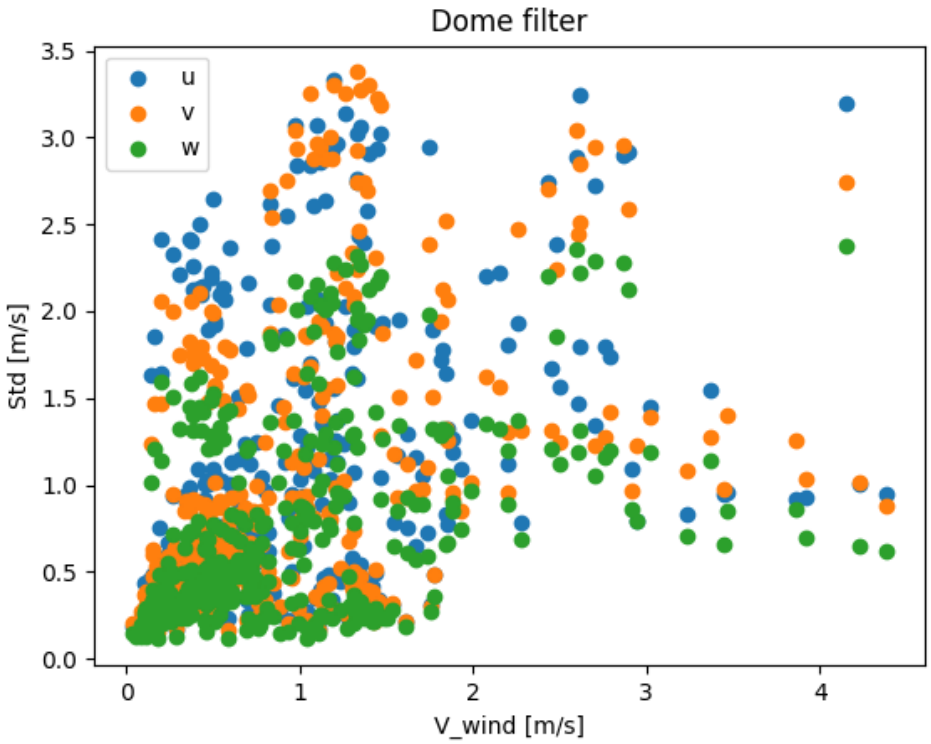


Figure B.3: Wind component standard deviation plotted against the average wind strength for the fabric dome filter.

C

Source code

This chapter contains excerpts containing the relevant parts of the code used to calculate the results found in this thesis.

convert.py

This section contains the code contained within the convert.py file, which was used for coordinate transformations.

```
import pyproj

cartesian_transformer = pyproj.Transformer.from_crs("WGS84", "EPSG:4978") # Transforms from lat, lon, alt to x, y, z (x goes through greenwich, z goes through poles)
to_cartesian = cartesian_transformer.transform
polar_transformer = pyproj.Transformer.from_crs("EPSG:4978", "WGS84") # Transforms from x, y, z to lat, lon, alt
to_polar = polar_transformer.transform
```

Event.py

This section contains the code contained within the Event.py file, which provided an interface to easily access meteor entry event information. This file contains a significant amount of duplicate code due to an updated workflow between different phases of the research.

```
import os
import math
from collections import namedtuple
import obspy
from obspy import UTCDateTime as UTC
import pandas ad pd
import numpy as np
import convert
from functools import cached_property, cache
```

```
10
11 def get_recording(idx, station):
12     for file in os.listdir(f"data/{idx}"):
13         if station not in file:
14             continue
15         else:
16             break
17     stream = obspy.read(f"data/{idx}/{file}")
18     return next(t for t in stream if t.stats.channel == "HDF")
19
20 def get_events(index = None):
21     dat = pd.read_csv("data/Data release.csv")
22     if index is None:
23         return dat
24     else:
25         return dat.iloc[index]
26
27 def nearby_events(lat, lon, alt = 0, max_dist = 300e3, max_dev = 1):
28     """
29     Creates an array of every event that happened within a given radius of the
           specified point.
30     """
31     point = convert.to_cartesian(lat, lon, alt)
32     dat = pd.read_csv("data/Data release.csv")
33     keys = dat.keys()
34     start_keys = [(keys == "Initial Latitude").argmax(), (keys == "Initial
           Longitude").argmax(), (keys == "Initial Altitude").argmax()]
35     end_keys = [(keys == "Final Latitude").argmax(), (keys == "Final Longitude").
           argmax(), (keys == "Final Altitude").argmax()]
36     for i, event in reversed([*enumerate(dat.values)]):
37         start_lat, start_lon, start_alt = [to_float(i) for i in event[start_keys]]
38         end_lat, end_lon, end_alt = [to_float(i) for i in event[end_keys]]
39         start = convert.to_cartesian(start_lat, start_lon, start_alt)
40         end = convert.to_cartesian(end_lat, end_lon, end_alt)
41         dist = get_dist(point, start, end)
42         angle = get_angle(point, start, end)
43         if np.isnan(dist) or dist > max_dist or abs(angle - 90) > max_dev:
44             dat = dat.drop(axis = 0, index = i)
45     return dat
46
47 def get_nearest(point, start, end, dtype = np.float64):
48     """
49     Returns the nearest point near 'point' on the line from start to end
50     """
51     point = np.array(point, dtype = dtype)
52     start = np.array(start, dtype = dtype)
53     end = np.array(end, dtype = dtype)
54     diff = point - start
55     line_length = np.linalg.norm(end - start)
56     assert line_length > 0
57     dir = (end - start) / line_length
```

```
58      t = dir.dot(diff) # The interpolant from start to end
59      # print(f"{t = }, {line_length = }")
60      t = np.clip(t, 0, line_length)
61      p_ref = start + t * dir
62      return p_ref
63
64  def get_nearest_height(point, event):
65      start = get_nearest(point, *(convert.to_cartesian(*p) for p in get_event_loc(
            event)))
66      h1 = convert.to_polar(*start)[2]
67      return h1
68
69  def get_dist(point, start, end, dtype = np.float64):
70      """
71      Returns the distance from point to the line formed by start-stop, with all
            coordinates given in cartesian form
72      """
73      p_ref = get_nearest(point, start, end, dtype)
74      return np.linalg.norm(point - p_ref)
75
76  def normalize(vec: np.ndarray):
77      return vec / np.linalg.norm(vec)
78
79  def get_angle(point, start, end, dtype = np.float64):
80      point = np.array(point, dtype = dtype)
81      start = np.array(start, dtype = dtype)
82      end = np.array(end, dtype = dtype)
83      dir = normalize(end - start)
84      a1 = np.degrees(np.acos(dir.dot(normalize(point - start))))
85      a2 = np.degrees(np.acos(dir.dot(normalize(point - end))))
86      if (a1 - 90) * (a2 - 90) <= 0:
87          return 90.
88      else:
89          return min(a1, a2, key = lambda i: abs(i - 90))
90
91  def get_event_loc(event):
92      """
93      Returns the event location as lat-lon-alt
94      """
95      keys = event.keys()
96      start_keys = ["Initial Latitude", "Initial Longitude", "Initial Altitude"]
97      end_keys = ["Final Latitude", "Final Longitude", "Final Altitude"]
98      start = tupLocation(*[to_float(i) for i in event[start_keys]])
99      end = tupLocation(*[to_float(i) for i in event[end_keys]])
100     return start, end
101
102 def estimate_time(point, event, temp_offset: float = 0., silent: bool = False):
103     """
104     Calculates the time estimate of the arrival of the sound wave
105     """
106     time = UTC(event["Multi-eventx"])
```

```
107     start = get_nearest(point, *(convert.to_cartesian(*p) for p in get_event_loc(
            event)))
108     dist = np.linalg.norm(point - start)
109     h0 = convert.to_polar(*point)[2]
110     h1 = convert.to_polar(*start)[2]
111     angle = np.acos((h1 - h0) / dist)
112     # print(f"{np.degrees(angle) = }")
113     gamma = 1.4
114     R = 287.05
115     if h1 > layers[-1][1] and not silent:
116         print(f"Nearest event altitude {h1/1e3:.1f} [km] exceeds highest layer of
                {layers[-1][1]/1e3:.1f} [km]")
117     for layer in layers:
118         delta_h = np.clip(h1, layer[0], layer[1]) - np.clip(h0, layer[0], layer
                [1])
119         if layer[2] == layer[3]: # Constant V_sound
120             time += delta_h / (np.cos(angle) * np.sqrt(gamma * R * (layer[2] +
                    temp_offset)))
121         else:
122             T0 = layer[2] + temp_offset
123             lapse_rate = (layer[3] - layer[2]) / (layer[1] - layer[0])
124             time += 2 / (lapse_rate * np.cos(angle) * np.sqrt(gamma * R)) * (np.
                    sqrt(T0 + lapse_rate * delta_h) - np.sqrt(T0))
125     return time
126
127 def get_params(station: str, event):
128     eventUTC = UTC(event["Multi-eventx"])
129     station_loc = convert.to_cartesian(*config.get_station_location(station,
            config.inv, eventUTC))
130     event_locs = [convert.to_cartesian(*i) for i in get_event_loc(event)]
131     t_arrival = estimate_time(station_loc, event, silent = True)
132     height = get_nearest_height(station_loc, event)
133     dist = get_dist(station_loc, *event_locs)
134     angle = get_angle(station_loc, *event_locs)
135     return t_arrival, height, dist, angle
136
137 def get_variance(point, event, delta_t: float = 15.):
138     return estimate_time(point, event, +delta_t, True), estimate_time(point, event
            , -delta_t, True)
139
140
141 tupLocation = namedtuple("Location", ["latitude", "longitude", "elevation"])
142
143 class EntryEvent():
144     def __init__(self, start: tupLocation, end: tupLocation, timestamp: UTC = UTC
            (0), name = None, **kwargs):
145         self.name = name
146         self.time = self.timestamp = timestamp
147         self.start = np.array(start, np.float64)
148         self.end = np.array(end, np.float64)
149         self.pdetcount = 0 # Point Detectability Count
```

```
150         self.__dict__ |= kwargs
151 
152     @classmethod
153     def from_row(cls, row):
154         """
155         Allows generating an EntryEvent object from a row as read from the FRIPON
                data release using a pandas dataframe
156         """
157         start_keys = ["Initial Latitude", "Initial Longitude", "Initial Altitude"]
158         end_keys = ["Final Latitude", "Final Longitude", "Final Altitude"]
159         start = tupLocation(*[to_float(i) for i in row[start_keys]])
160         end = tupLocation(*[to_float(i) for i in row[end_keys]])
161 
162         return cls(start, end, UTC(row["Multi-eventx"]), row["idx"])
163 
164     @cached_property
165     def start_xyz(self) -> "xyz":
166         return convert.to_cartesian(*self.start)
167     @cached_property
168     def end_xyz(self) -> "xyz":
169         return convert.to_cartesian(*self.end)
170     @cached_property
171     def heading(self) -> float:
172         """
173         Calculates the initial heading of the entry event
174         """
175         lat1, lon1, _ = np.radians(self.start)
176         lat2, lon2, _ = np.radians(self.end)
177         y = math.sin(lon2 - lon1) * math.cos(lat2)
178         x = math.cos(lat1) * math.sin(lat2) - \
179             math.sin(lat1) * math.cos(lat2) * math.cos(lon2 - lon1)
180         initial_heading = math.degrees(math.atan2(y, x))
181         return initial_heading % 360
182 
183     @cache
184     def get_nearest(self, point_xyz) -> "xyz":
185         if isinstance(point_xyz, Station):
186             point_xyz = point_xyz.xyz
187         return get_nearest(point_xyz, self.start_xyz, self.end_xyz)
188     @cache
189     def get_nearest_height(self, point_xyz) -> "xyz":
190         return convert.to_polar(*self.get_nearest(point_xyz))[2]
191     @cache
192     def get_dist(self, point_xyz) -> float:
193         if isinstance(point_xyz, Station):
194             point_xyz = point_xyz.xyz
195         p_ref = self.get_nearest(point_xyz)
196         return np.linalg.norm(point_xyz - p_ref)
197     @cache
198     def get_angle(self, point_xyz) -> float:
199         if isinstance(point_xyz, Station):
```

```
200             point_xyz = point_xyz.xyz
201         point = np.array(point_xyz, np.float64)
202         dir = normalize(np.array(self.end_xyz) - self.start_xyz)
203         a1 = np.degrees(np.acos(dir.dot(normalize(point - self.start_xyz))))
204         a2 = np.degrees(np.acos(dir.dot(normalize(point - self.end_xyz))))
205         if (a1 - 90) * (a2 - 90) <= 0:
206             return 90.
207         else:
208             return min(a1, a2, key = lambda i: abs(i - 90))
209 
210     @cache
211     def arrival_time(self, point_xyz, temp_offset: float = 0., silent: bool = True
        ) -> UTC:
212         """
213         Calculates the time estimate of the arrival of the sound wave
214         """
215         if isinstance(point_xyz, Station):
216             point_xyz = point_xyz.xyz
217         time = UTC(self.time)
218         start = self.get_nearest(point_xyz)
219         dist = np.linalg.norm(point_xyz - start)
220         h0 = convert.to_polar(*point_xyz)[2]
221         h1 = convert.to_polar(*start)[2]
222         angle = np.acos((h1 - h0) / dist)
223         # print(f"{np.degrees(angle) = }")
224         gamma = 1.4
225         R = 287.05
226         if h1 > layers[-1][1] and not silent:
227             print(f"Nearest event altitude {h1/1e3:.1f} [km] exceeds highest layer
                    of {layers[-1][1]/1e3:.1f} [km]")
228         for layer in layers:
229             delta_h = np.clip(h1, layer[0], layer[1]) - np.clip(h0, layer[0],
                    layer[1])
230             if layer[2] == layer[3]: # Constant V_sound
231                 time += delta_h / (np.cos(angle) * np.sqrt(gamma * R * (layer[2] +
                        temp_offset)))
232             else:
233                 T0 = layer[2] + temp_offset
234                 lapse_rate = (layer[3] - layer[2]) / (layer[1] - layer[0])
235                 time += 2 / (lapse_rate * np.cos(angle) * np.sqrt(gamma * R)) * (
                        np.sqrt(T0 + lapse_rate * delta_h) - np.sqrt(T0))
236         return time
237 
238     @cache
239     def arrival_window(self, point_xyz, delta_t: float = 15., margin: float = 0.,
        silent: bool = True) -> tuple[UTC]:
240         if isinstance(point_xyz, Station):
241             point_xyz = point_xyz.xyz
242         return (self.arrival_time(point_xyz, +delta_t, silent), self.arrival_time(
            point_xyz, -delta_t, silent))
243     window = arrival_window
```

```
244     @cache
245     def pre_window(self, point_xyz, delta_t: float = 15., margin: float = 0.,
            silent: bool = True) -> tuple[UTC]:
246         window = self.arrival_window(point_xyz, delta_t, margin, silent)
247         return window[0] - (window[1] - window[0]), window[0]
248     @cache
249     def window_length(self, point_xyz, delta_t: float = 15., margin: float = 0.,
            silent: bool = True) -> float:
250         window = self.arrival_window(point_xyz, delta_t, margin, silent)
251         return window[1] - window[0]
252 
253     def add_station(self, station, max_dist = 1000e3):
254         if isinstance(station, Station):
255             station_xyz = station.xyz
256         else:
257             station_xyz = convert.to_cartesian(*[*station, 0][:3])
258         dist = self.get_dist(station_xyz)
259         if dist < max_dist:
260             self.pdetcount += 1
261 
262     def if_station(self, station, max_dist = 1000e3):
263         if isinstance(station, Station):
264             station_xyz = station.xyz
265         else:
266             station_xyz = convert.to_cartesian(*[*station, 0][:3])
267         dist = self.get_dist(station_xyz)
268         if dist < max_dist:
269             return self.pdetcount + 1
270         else:
271             return self.pdetcount
272 
273     def __str__(self):
274         if self.name is not None:
275             return f"<EntryEvent: '{self.name}'>"
276         else:
277             return super().__str__()
278     def __repr__(self):
279         if self.name is not None:
280             return f"<EntryEvent: '{self.name}'>"
281         else:
282             return super().__repr__()
283 
284 class Station():
285     def __init__(self, name: str, loc: tuple[float], **kwargs):
286         self.name = name
287         self.location = tupLocation(*loc) if len(loc) == 3 else tupLocation(*loc,
                0)
288         self.__dict__ |= kwargs
289     @cached_property
290     def latlon(self):
291         return self.location[:2]
```

```
292     @cached_property
293     def xyz(self):
294         return convert.to_cartesian(*self.location)
295 
296     def __repr__(self) -> str:
297         return f"<Station: '{self.name}'>"
298     def __str__(self) -> str:
299         return f"<Station: '{self.name}'>"
300 
301 # Layers based on:
302 # https://ntrs.nasa.gov/api/citations/19770003812/downloads/19770003812.pdf
303 layers = [ # Base height [m], Top height [m], Base temp [K], Top temp [K]
304     [0, 11_019, 288.15, 216.65],
305     [11_019, 20_063, 216.65, 216.65],
306     [20_063, 32_162, 216.65, 228.65],
307     [32_162, 47_350, 228.65, 270.65],
308     [47_350, 51_412, 270.65, 270.65],
309     [51_412, 71802, 270.65, 214.65],
310     [71802, 86_000, 214.65, 186.8673],
311     [86_000, 91_000, 186.8673, 186.8673],
312     [91_000, 110_000, 186.8673, 240.],
313     [110_000, 120_000, 240., 360.]
314     ]
```

lib/Detect.py

The Detect.py file contained the implementation of the actual detection of meteors from infrasound traces.

```
1  import numpy as np
2  import scipy as sp
3  from matplotlib import mlab
4  import obspy
5  from obspy import UTCDateTime as UTC
6 
7  import matplotlib.pyplot as plt
8 
9  import Event
10 import convert
11 
12 class TriggerFFT():
13     def __init__(self, t_start, threshold, freqs = None):
14         self.t_start = t_start
15         self.threshold = threshold.copy()
16         self.t_end = None
17         self.freq_range = np.zeros_like(threshold, dtype = float)
18         self.freqs = freqs
19 
20     @property
21     def is_open(self):
22         return self.t_end is None
```

```
23      def close(self, t_end):
24          self.t_end = t_end
25      @property
26      def norm_freq_range(self):
27          return self.freq_range / float(self.t_end - self.t_start)
28      @property
29      def length(self):
30          if self.is_open:
31              return np.inf
32          else:
33              return float(self.t_end - self.t_start)
34      @property
35      def window(self):
36          return (self.t_start, self.t_end)
37
38
39  def detect_fft(trace: obspy.Trace, win_len: int = 256, overlap: float = 128,
        threshold: float = 2.5, avg_len: int = 24) -> list[TriggerFFT]:
40      """
41      Applies automatic anomaly detection in the FFT domain.
42      trace: obspy.Trace = The waveform trace on which to apply the detection.
43      win_len: int = The length of each sampling window in nr of samples. A power of
             2 is most efficient.
44      overlap: int = The overlap between each sampling window with the previous
            window in number of samples.
45      threshold: float = The factor that the signal needs to be over the average to
            count as a trigger.
46      avg_len: int = The number of FFT segments used to determine the average /
            reference value.
47      """
48      active_trigger = None
49      triggers = []
50      data, freqs, ts = mlab.specgram(trace, Fs = trace.stats.sampling_rate, NFFT =
            win_len, noverlap = overlap, window = mlab.window_none)
51      data[0, :] = 0 # Set the mean value to 0
52      t_sample = (win_len - overlap) * trace.stats.delta
53      t = trace.stats.starttime + ts[24]
54      ref = np.sum(data[:, :avg_len], axis = 1)
55      # breakpoint()
56
57      for i in range(avg_len, len(ts)):
58          value = data[:, i]
59          if not active_trigger:
60              _threshold = (threshold / avg_len) * ref
61              if (value > _threshold).any():
62                  active_trigger = TriggerFFT(t, _threshold, freqs)
63                  # print(f"start = {t}", end = "")
64          # NOTE: No Elif, since this also handles updating the freq_range
65          if active_trigger:
66              still_active = (value > active_trigger.threshold).any()
67              if not still_active:
```

```
68                  active_trigger.close(t)
69                  triggers.append(active_trigger)
70                  active_trigger = None
71                  # print(f" | end = {t}")
72              else:
73                  active_trigger.freq_range += (value > active_trigger.threshold) *
                        t_sample
74
75          ref += data[:, i] - data[:, i - avg_len]
76          t += t_sample
77
78      if active_trigger is not None:
79          active_trigger.close(t)
80          triggers.append(active_trigger)
81      return triggers
82
83  def detect_fft_std(trace: obspy.Trace, win_len: int = 256, overlap: float = 128,
        threshold: float = 2.5, avg_len: int = 24) -> list[TriggerFFT]:
84      """
85      Applies automatic anomaly detection in the FFT domain.
86      trace: obspy.Trace = The waveform trace on which to apply the detection.
87      win_len: int = The length of each sampling window in nr of samples. A power of
             2 is most efficient.
88      overlap: int = The overlap between each sampling window with the previous
            window in number of samples.
89      threshold: float = The factor that the signal needs to be over the average to
            count as a trigger.
90      avg_len: int = The number of FFT segments used to determine the average /
            reference value.
91      """
92      active_trigger = None
93      triggers = []
94      data, freqs, ts = mlab.specgram(trace, Fs = trace.stats.sampling_rate, NFFT =
            win_len, noverlap = overlap, window = mlab.window_none)
95      data[0, :] = 0 # Set the mean value to 0
96      t_sample = (win_len - overlap) * trace.stats.delta
97      t = trace.stats.starttime + ts[24]
98      refs = [data[:, i] for i in range(avg_len)]
99      # breakpoint()
100
101     for i in range(avg_len, len(ts)):
102         value = data[:, i]
103         if not active_trigger:
104             _threshold = np.mean(refs, axis = 0) + threshold * np.std(refs, axis =
                    0)
105             if (value > _threshold).any():
106                 active_trigger = TriggerFFT(t, _threshold, freqs)
107                 # print(f"start = {t}", end = "")
108         # NOTE: No Elif, since this also handles updating the freq_range
109         if active_trigger:
110             still_active = (value > active_trigger.threshold).any()
```

```
111             if not still_active:
112                 active_trigger.close(t)
113                 triggers.append(active_trigger)
114                 active_trigger = None
115                 # print(f" | end = {t}")
116             else:
117                 active_trigger.freq_range += (value > active_trigger.threshold) *
                        t_sample
118 
119         refs.pop(0)
120         refs.append(data[:, i])
121         t += t_sample
122 
123     if active_trigger is not None:
124         active_trigger.close(t)
125         triggers.append(active_trigger)
126     return triggers
127 
128 def detect_event(index: int, station: str, plot: bool = False, silent: bool =
        False, *args, **kwargs):
129     inv = obspy.read_inventory("data/inventory.xml")
130     event = Event.get_events(index)
131     trace = Event.get_recording(index, station)
132 
133     triggers = detect_fft(trace, *args, **kwargs)
134 
135     station_loc = convert.to_cartesian(*config.get_station_location(station, inv,
            UTC(event["Multi-eventx"])))
136     t_start = Event.estimate_time(station_loc, event, +20, silent = silent) - 30 #
             Max ref temp: +35 C
137     t_end = Event.estimate_time(station_loc, event, -20, silent = silent) # Min
            ref temp: -5 C
138 
139     count = sum(trigger.t_start >= t_start and trigger.t_end <= t_end and (trigger
            .freq_range[trigger.freqs < 25]).any() for trigger in triggers)
140     count = count if count or not triggers else -1
141 
142     if plot:
143         # Trigger timeline
144         for i, trigger in enumerate(triggers):
145             plt.plot((float(trigger.t_start), float(trigger.t_end)), (i % 1, i %
                    1))
146         plt.axvline(float(t_start))
147         plt.axvline(float(t_end))
148         plt.show()
149 
150         for i, trigger in enumerate(triggers):
151             plt.plot(trigger.norm_freq_range, label = f"{i}")
152         plt.legend()
153         plt.show()
154 
```

```
155     return count, len(triggers)
```

DetectCompare.py

The DetectCompare.py file functioned as a wrapper around Detect.py, and provides the logic to turn these detections into actual detection classifications. This code was developed for the investigation into Raspberry Shake data.

```
1  import os
2  from collections import Counter
3  import obspy
4  from obspy.imaging.waveform import WaveformPlotting
5  from obspy import UTCDateTime as UTC
6  import matplotlib.pyplot as plt
7
8  import Event
9  from lib import Detect
10
11 def iter_data2(margin: float = 15.):
12     """
13     Iterates over all data files contained in the data2 folder
14     """
15     inv = obspy.read_inventory("data/inventory.xml")
16     ev_dict = {ev["idx"]: Event.EntryEvent.from_row(ev) for i, ev in Event.
           get_events().iterrows()}
17     for folder in filter(lambda fol: os.path.isdir(f"data2/{fol}") and fol.
           isnumeric(), os.listdir("data2")):
18         event = ev_dict[int(folder)]
19         for file in os.listdir(f"data2/{folder}"):
20             stream = obspy.read(f"data2/{folder}/{file}")
21             trace = next(t for t in stream if t.stats.channel == "HDF")
22             station_name = file[:5]
23             try:
24                 station = Event.Station(station_name, [*inv.get_coordinates(trace.
                       id, trace.stats.starttime).values()][:3])
25             except:
26                 station = Event.Station(station_name, [*inv.get_coordinates(trace.
                       id).values()][:3])
27             if is_full_valid(event, station, trace, margin = margin):
28                 yield event, station, trace
29
30 def detect_dynamic() -> tuple[Counter]:
31     thresholds = []
32     results = []
33     for event, station, trace in iter_data2():
34         threshold = 1.
35         while True:
36             triggers = Detect.detect_fft(trace, threshold = threshold)
37             if (result := classify_result(triggers, event, station)) in ["Peaky",
                   "Over-sensitive", "Inconclusive"]:
38                 threshold *= 1.5
```

```
            else:
                break
        thresholds.append(threshold)
        results.append(result)
        if results[-1] == "Detection":
            wf, fig, ax = make_fig(trace.copy().trim(*event.window(station, margin
                = 15.)), triggers)
            fig.savefig(f"figures/autodetect/auto_ref_{event.name}_{station.name}_
                {threshold:.1f}.png")
    return Counter(thresholds), Counter(results)

def detect_dynamic_std() -> tuple[Counter]:
    thresholds = []
    results = []
    for event, station, trace in iter_data2():
        threshold = 1.
        while True:
            triggers = Detect.detect_fft_std(trace, threshold = threshold)
            if (result := classify_result(triggers, event, station)) in ["Peaky",
                "Over-sensitive", "Inconclusive"]:
                threshold *= 1.5
            else:
                break
        thresholds.append(threshold)
        results.append(result)
        if results[-1] == "Detection":
            wf, fig, ax = make_fig(trace.copy().trim(*event.window(station, margin
                = 15.)), triggers)
            fig.savefig(f"figures/autodetect/auto_std_{event.name}_{station.name}_
                {threshold:.1f}.png")
    return Counter(thresholds), Counter(results)

def detect_static_ref(threshold: float = 2.5) -> Counter:
    results = []
    for event, station, trace in iter_data2():
        triggers = Detect.detect_fft(trace, threshold = threshold)
        results.append(classify_result(triggers, event, station))
        if results[-1] == "Detection":
            wf, fig, ax = make_fig(trace.copy().trim(*event.window(station, margin
                = 15.)), triggers)
            fig.savefig(f"figures/autodetect/ref_{threshold:.1f}_{event.name}_{
                station.name}.png")
    return Counter(results)

def detect_static_std(threshold: float = 2.5) -> Counter:
    results = []
    for event, station, trace in iter_data2():
        triggers = Detect.detect_fft_std(trace, threshold = threshold)
        results.append(classify_result(triggers, event, station))
        if results[-1] == "Detection":
```

```
82              wf, fig, ax = make_fig(trace.copy().trim(*event.window(station, margin
                    = 15.)), triggers)
83              # wf, fig, ax = make_fig(trace, triggers)
84              window = event.window(station, margin = 15)
85              # ax.axvline(window[0], color = "blue")
86              # ax.axvline(window[1], color = "blue")
87              fig.savefig(f"figures/autodetect/std_{threshold:.1f}_{event.name}_{
                    station.name}.png")
88          elif results[-1] == "Inconclusive":
89              # wf, fig, ax = make_fig(trace.copy().trim(*event.window(station,
                    margin = 15.)), triggers)
90              wf, fig, ax = make_fig(trace, triggers)
91              pre_window = event.pre_window(station, margin = 15)
92              window = event.window(station, margin = 15)
93              ax.axvline(pre_window[0], color = "red")
94              ax.axvline(window[0], color = "blue")
95              ax.axvline(window[1], color = "blue")
96              fig.savefig(f"figures/autodetect/inconclusive/std_{threshold:.1f}_{
                    event.name}_{station.name}.png")
97              # plt.show()
98              # breakpoint()
99      return Counter(results)
100
101 def is_full_valid(event: Event.EntryEvent, station: Event.Station, trace: obspy.
        Trace, margin: float = 15.) -> bool:
102     pre_window = event.pre_window(station, margin = margin)
103     window = event.arrival_window(station, margin = margin)
104     return (trace.stats.starttime <= (pre_window[0] - 30)) and (trace.stats.
            endtime >= window[1])
105
106 def classify_triggers(triggers, event, station, margin: float = 15.) -> list[int]:
107     """
108     Returns a list of four integers (a, b, c, d)
109     a: int = The number of triggers which lie fully inside the pre_window
110     b: int = The number of triggers which lie fully inside the window
111     c: int = The number of triggers which are inconclusive (overlapping with one
            of the windows, but not fully inside it)
112     d: int = The number of triggers which are irrelevant (not overlapping any
            window)
113     """
114     pre_window = event.pre_window(station, margin = margin)
115     window = event.window(station, margin = margin)
116     result = [0, 0, 0, 0]
117     for trigger in triggers:
118         if trigger.t_start > pre_window[0] and trigger.t_end < pre_window[1]: #
                Fully inside pre_window
119             result[0] += 1
120         elif trigger.t_start > window[0] and trigger.t_end < window[1]: # Fully
                inside window
121             result[1] += 1
```

```
122         elif trigger.t_start < window[1] and trigger.t_end > pre_window[0]: # Partially inside pre_window or window
123             result[2] += 1
124         else: # In nothing
125             result[3] += 1
126     return result
127 
128 def classify_result(triggers, event, station, margin: float = 15.) -> str:
129     """
130     Returns a classification of the detection as a string. Will be any of the following, with priority in this order.
131     'Peaky' if trigger count > 2
132     'Over-sensitive' if trigger count < 2 and sum(trigger duration) > window length
133     'Detection' if only triggers in window, and trigger count <= 2
134     'False-Positive' if only triggers in pre_window and trigger count <= 2
135     'None' if no triggers at all
136     'Inconclusive' otherwise
137     """
138     window_length = event.window_length(station, margin = margin)
139     if sum(trigger.length for trigger in triggers) > window_length:
140         return 'Over-sensitive'
141     trig_counts = classify_triggers(triggers, event, station, margin)
142     if len(triggers) > 2:
143         return 'Peaky'
144     elif trig_counts[1] and not (trig_counts[0] or trig_counts[2]):
145         return 'Detection'
146     elif trig_counts[0] and not (trig_counts[1] or trig_counts[2]):
147         return 'False-Positive'
148     elif sum(trig_counts[:3]) == 0:
149         return 'None'
150     else:
151         return 'Inconclusive'
152 
153 def make_fig(trace: obspy.Trace, triggers: list[Detect.TriggerFFT]):
154     wf = WaveformPlotting(stream = trace, show = False, block = True)
155     fig = wf.plot_waveform()
156     ax = fig.gca()
157     for trigger in triggers:
158         ax.axvspan(trigger.t_start, trigger.t_end, color = "red", alpha = 0.25)
159     return wf, fig, ax
160 
161 def print_count(counter, method = ""):
162     # print(f"Method & ", end = "")
163     if method:
164         print(rf"\textbf[{method}] & ".replace("[", "{").replace("]", "}"), end = "")
165     print(" & ".join(f"{counter[i]}" for i in ["Peaky", "Over-sensitive", "Detection", "False-Positive", "None", "Inconclusive"]))
166 
167 if __name__ == "__main__":
```

```
    # ================================< Full test >================================
    print("Dynamic baseline")
    print_count(detect_dynamic()[1])
    print("Dynamic STD")
    print_count(detect_dynamic_std()[1])
    for thr in [2.5, 5, 10, 20, 50, 100, 200, 500]:
        print(f"Baseline: {thr}")
        print_count(detect_static_ref(thr), f"Baseline ({thr})")
    for thr in [2.5, 5, 10, 20, 50, 100, 200, 500]:
        print(f"STD: {thr}")
        print_count(detect_static_std(thr), f"Standard dev ({thr})")
```

HomeExperiment.py

The HomeExperiment.py file contains the code for validating the signal response of the different filters with known forcing signals.

```
import math
import numpy as np
import obspy
from obspy import UTCDateTime as UTC
from matplotlib import mlab
import matplotlib.pyplot as plt

filt = (0.5, 1., 45, 50)

def get_streams():
    """
    Returns the trimmed streams as
    [ # Run nr 1 / 2
        [ # Setup a / b
            [ # Streams 20 Hz, 30 Hz, 40 Hz]
        ]
    ]
    """
    margin = 5 # [Seconds]
    inv = obspy.read_inventory("edata/RF442/RF442.xml")
    s = obspy.read("edata/RF442/HDF.D/AM.RF442.00.HDF.D.2025.265")
    s.remove_response(inv, water_level = None, pre_filt = filt, output = "DEF")
    s1_a30 = s.copy().trim(UTC("2025/265:11:50:00") + margin, UTC("
        2025/265:11:52:00") - margin)
    s1_a40 = s.copy().trim(UTC("2025/265:11:52:00") + margin, UTC("
        2025/265:11:53:00") - margin)
    s1_a20 = s.copy().trim(UTC("2025/265:12:08:00") + margin, UTC("
        2025/265:12:09:00") - margin)

    s1_b30 = s.copy().trim(UTC("2025/265:12:00:00") + margin, UTC("
        2025/265:12:02:00") - margin)
    s1_b40 = s.copy().trim(UTC("2025/265:12:02:00") + margin, UTC("
        2025/265:12:03:00") - margin)
```

```
    s1_b20 = s.copy().trim(UTC("2025/265:12:04:00") + margin, UTC("
        2025/265:12:05:00") - margin)

    s2 = obspy.read("edata/RF442/HDF.D/AM.RF442.00.HDF.D.2025.306")
    s2.remove_response(inv, water_level = None, pre_filt = filt, output = "DEF")
    s2_a = s2.copy().trim(UTC("2025/306:13:04:00"), UTC("2025/306:13:07:00"))
    s2_a30 = s2_a.copy().trim(UTC("2025/306:13:04:00") + margin, UTC("
        2025/306:13:05:00") - margin)
    s2_a40 = s2_a.copy().trim(UTC("2025/306:13:05:00") + margin, UTC("
        2025/306:13:06:00") - margin)
    s2_a20 = s2_a.copy().trim(UTC("2025/306:13:06:00") + margin, UTC("
        2025/306:13:07:00") - margin)

    s2_b = s2.copy().trim(UTC("2025/306:13:23:00"), UTC("2025/306:13:26:00"))
    s2_b30 = s2_b.copy().trim(UTC("2025/306:13:23:00") + margin, UTC("
        2025/306:13:24:00") - margin)
    s2_b40 = s2_b.copy().trim(UTC("2025/306:13:24:00") + margin, UTC("
        2025/306:13:25:00") - margin)
    s2_b20 = s2_b.copy().trim(UTC("2025/306:13:25:00") + margin, UTC("
        2025/306:13:26:00") - margin)

    return [
        [(s1_a20, s1_a30, s1_a40), (s1_b20, s1_b30, s1_b40)],
        [(s2_a20, s2_a30, s2_a40), (s2_b20, s2_b30, s2_b40)]
    ]

def plot_comparison(setup: int = 0):
    band_width = 1.
    pa_ref = 1.
    _label = ["Porous hose", "Fabric dome"][setup]
    streams = get_streams()[setup]
    global fig
    fig, axs = plt.subplots(3, 1, sharex = True, figsize = (7.5, 6.0))
    for i, traces in enumerate(zip(*streams)):
        ax = axs[i]
        freq_band = [20, 30, 40][i]
        psd_sums = []
        print(f"{freq_band}")
        for label, trace in zip(["Bare sensor", _label], traces):
            freqs, psd = get_freqs(trace[0])
            ax.semilogy(freqs, psd, label = label)

            freq_mask = ((freq_band - band_width / 2) < freqs) & (freqs < (
                freq_band + band_width / 2))
            psd_sum = psd[freq_mask].sum()
            print(f"{label}: {psd_sum}")
            psd_sums.append(psd_sum)

        print(f"PSD reduction: {10 * math.log(psd_sums[1] / psd_sums[0]):.3f} [dB]
            ")
        ax.set_title(f"Test tone at {freq_band} [Hz]")
```

```
71          ax.set_ylabel("Power spectrum [Pa$^2$]")
72          ax.set_ylim(ymin = 10**-8)
73          ax.legend()
74          db_ax = ax.twinx()
75          db_ax.set_yscale("log")
76          db_ax.set_ylim(ax.get_ylim())
77          db_ax.yaxis.set_major_formatter(lambda i, _: f"{10 * np.log(i / pa_ref) /
                np.log(10):+.0f}")
78          db_ax.set_ylabel("Power spectrum [dB]")
79      ax.set_xlabel("Frequency [Hz]")
80      fig.suptitle("Forced tone power spectrum")
81      plt.tight_layout()
82      fig.savefig(f"figures/home_setup_{setup}.png")
83      plt.show()
84
85  def plot_relative_db():
86      streams = get_streams()
87      fig, axs = plt.subplots(3, 1, sharex = True, figsize = (7.5, 6.0))
88      for i, freq_band in enumerate([20, 30, 40]):
89          for label, setup_streams in zip(["Porous hose", "Fabric dome"], streams):
90              ax = axs[i]
91              freq, psd_ref = get_freqs(setup_streams[0][i][0])
92              freq, psd = get_freqs(setup_streams[1][i][0])
93              db = 10 * np.log(psd / psd_ref) / np.log(10)
94              ax.plot(freq, db, label = label)
95          ax.set_title(f"Test tone at {freq_band} [Hz]")
96          ax.set_ylabel("PSD reduction [dB]")
97          ax.legend()
98          ax.grid()
99      ax.set_xlabel(("Frequency [Hz]"))
100     fig.suptitle("Forced tone PSD reduction")
101     plt.tight_layout()
102     fig.savefig(f"figures/home_setup_db.png")
103     plt.show()
104
105
106 def get_freqs(trace, scale_by_freq = False):
107     psd, freqs = mlab.psd(trace.data, 256, Fs = trace.stats.sampling_rate,
            scale_by_freq = scale_by_freq)
108     return freqs, psd
109
110
111 if __name__ == "__main__":
112     plot_comparison(0)
113     plot_comparison(1)
114
115     plot_relative_db()
```

Rooftop.py

The Rooftop.py file contained the code for analysing the noise reduction performance for the sensors on the TU Delft rooftop lab.

```
import re
import os
import pickle
import math
import itertools

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import mlab
import windrose

import obspy
from obspy import UTCDateTime as UTC

filt = (0.25, 0.5, 45, 50)

def reduce_wind(data: pd.DataFrame, w_len: float = 1. * 60) -> pd.DataFrame:
    """
    data: pd.DataFrame = The data as loaded from wind.{date}.dat
    w_len: window length in seconds for averaging
    """
    t_start = UTC(data.cpu_time[0][:19])
    out = []
    rows = []

    for idx, row in data.iterrows():
        if UTC(row.cpu_time) >= t_start + w_len:
            out.append((t_start, rows))
            t_start += w_len
            rows = []
        if row.status == 0:
            rows.append(row)

    # Append the final row
    out.append((t_start, rows))

    out_df = pd.DataFrame()
    out_df["time"] = [i[0] for i in out]
    out_df["u_avg"] = [sum(j.u for j in i[1]) / len(i[1]) for i in out]
    out_df["u_std"] = [np.std([j.u for j in i[1]]) for i in out]
    out_df["v_avg"] = [sum(j.v for j in i[1]) / len(i[1]) for i in out]
    out_df["v_std"] = [np.std([j.v for j in i[1]]) for i in out]
    out_df["w_avg"] = [sum(j.w for j in i[1]) / len(i[1]) for i in out]
    out_df["w_std"] = [np.std([j.w for j in i[1]]) for i in out]
    out_df["temp_avg"] = [sum(j.stemp for j in i[1]) / len(i[1]) for i in out]

```

```
48      return out_df
49
50  def get_reduced_wind(file: str, w_len: float = 1. * 60) -> pd.DataFrame:
51      dirname, filename = os.path.split(file)
52      red_dirname = os.path.split(dirname)[0] + "/reduced/wind"
53      red_filename = os.path.splitext(filename)[0] + f"-w{w_len:.0f}" + ".csv"
54      new_file = os.path.join(red_dirname, red_filename)
55      if os.path.exists(new_file):
56          new_data = pd.read_csv(new_file, sep = ",")
57      else:
58          data = pd.read_csv(file, sep = ",")
59          new_data = reduce_wind(data, w_len)
60          new_data.to_csv(new_file)
61      return new_data
62
63  def reduce_noise(data: obspy.Stream, w_len: float = 1. * 60) -> dict[str, tuple[np
        .ndarray]]:
64      """
65      Returns a dict containing:
66      key: UTC string
67      value: tuple[
68       np.ndarray: noise (PSD),
69       np.ndarray: freqs
70       ]
71      """
72      trace = data[0]
73      start = UTC(trace.stats.starttime.isoformat()[:16])
74      station = trace.stats.station
75      inventory = obspy.read_inventory(f"edata/{station}/{station}.xml")
76      psds = {}
77      data.remove_response(inventory, water_level = None, pre_filt = filt, output =
            "DEF")
78      while start < trace.stats.starttime:
79          start += w_len
80      while start + w_len < trace.stats.endtime:
81          trace_sect = trace.copy().trim(start, start + w_len)
82          freqs, psd = mlab.psd(trace_sect.data, 256, Fs = trace.stats.sampling_rate
                )
83          psds[start.isoformat()] = (freqs, psd)
84          start += w_len
85      return psds
86
87  def get_reduced_noise(file: str, w_len: float = 1. * 60) -> dict[str, tuple[np.
        ndarray]]:
88      """
89      Returns a tuple containing:
90      1. A list of UTCDateTime objects corresponding to the start of the window
91      2. A numpy array containing the frequencies of the PSD
92      3. A numpy array containing the PSD spectrum
93      """
94      filename = os.path.split(file)[-1]
```

```
95      sensor = filename[3:8]
96      newfile = f"edata/reduced/{sensor}/{filename}-w{w_len:.0f}.pickle"
97      if os.path.exists(newfile):
98          with open(newfile, "rb") as f:
99              return pickle.load(f)
100     stream = obspy.read(file)
101     data = reduce_noise(stream, w_len)
102     with open(newfile, "wb+") as f:
103         pickle.dump(data, f)
104     return data
105
106 def get_noise_data(start_date: UTC, end_date: UTC, w_len: float = 10. * 60,
        scale_by_freq: bool = True) -> tuple[np.ndarray, np.ndarray]:
107     """
108     Returns four arrays:
109     1. V_wind
110     2. A_wind (wind angle)
111     3. Noise_data
112     4. Freqs
113     """
114     w_len_s = f"w{w_len:.0f}"
115     day_range = range(start_date.julday, end_date.julday + 1)
116     V_wind = []
117     A_wind = []
118     stds = []
119     Noise_data = []
120     for file in os.listdir("edata/reduced/wind"):
121         wind_file = f"edata/reduced/wind/{file}"
122         # If the reduced wind file is not reduced to the wanted resolution, skip
123         if not os.path.splitext(file)[0].endswith(w_len_s):
124             continue
125         # If the reduced wind files' date falls outside the requested window, skip
126         wind_date = UTC(file[5:19])
127         if not start_date <= wind_date < end_date:
128             continue
129
130         noise_file = f"edata/reduced/R6D4E/AM.R6D4E.00.HDF.D.{wind_date.year}.{
              wind_date.julday}-{w_len_s}.pickle"
131         if not os.path.exists(noise_file):
132             print(f"Could not find noise data for {wind_date.year}.{wind_date.
                  julday} at {w_len_s}")
133             continue
134
135         wind_data = pd.read_csv(wind_file, sep = ",")
136         with open(noise_file, "rb") as f:
137             noise_data = pickle.load(f)
138
139         for idx, row in wind_data.iterrows():
140             timestr = row["time"][:19]
141             noise, freqs = noise_data.get(timestr, (None, None))
142             if noise is None:
```

```
143                 continue
144
145             V_wind.append(math.sqrt(row["u_avg"]**2 + row["v_avg"]**2 + row["w_avg
                    "]**2))
146             A_wind.append(math.atan2(row["v_avg"], row["u_avg"]))
147             try:
148                 stds.append([row["u_std"], row["v_std"], row["w_std"]])
149             except:
150                 pass
151             Noise_data.append(noise)
152             Freqs = freqs
153
154     if not scale_by_freq:
155         Noise_data = np.array(Noise_data) * (Freqs[1] - Freqs[0])
156     plt.scatter(V_wind, np.array(stds)[:, 0], label = "u")
157     plt.scatter(V_wind, np.array(stds)[:, 1], label = "v")
158     plt.scatter(V_wind, np.array(stds)[:, 2], label = "w")
159     plt.axis("equal")
160     plt.legend()
161     plt.xlabel("V_wind [m/s]")
162     plt.ylabel("Std [m/s]")
163     # plt.savefig("figures/vwind_std.png")
164     # plt.show()
165     return V_wind, A_wind, Noise_data, Freqs #, np.linalg.norm(stds, axis = 1)
166
167
168 if __name__ == "__main__":
169     # ============================< Single Multi-w_len >============================
170     # year = 2025
171     # month = 9
172     # day = 24
173     # hour = 0
174     # file = f"edata/wind/wind.{year}{month:0>2}{day:0>2}{hour:0>2}0000.dat"
175     # w_lens = [15, 30, 60., 600]
176     #
177     # for w_len in w_lens:
178     #     dat = get_reduced_wind(file, w_len)
179     #     print("Plotting")
180     #     plt.plot(dat.u_avg, dat.v_avg, label = f"w_len: {w_len:.0f} [s]")
181     # plt.legend()
182     # plt.show()
183
184     # ============================< Multi Single-w_len >============================
185     # w_len = 5 * 60
186     # for f in os.listdir("edata/wind")[:10]:
187     #     file = f"edata/wind/{f}"
188     #     dat = get_reduced_wind(file, w_len)
189     #     plt.plot(dat.u_avg, dat.v_avg, label = f"{f}")
190     #     print("*", end = "")
191     # plt.axis("equal")
192     # plt.legend()
```

```
    # plt.show()

    # ==============================< Parallel gen >==============================
    # w_len = 10 * 60
    # import concurrent.futures
    # files = [f"edata/wind/{f}" for f in os.listdir("edata/wind")]
    # with concurrent.futures.ProcessPoolExecutor() as executor:
    #     res = executor.map(get_reduced_wind, files, (w_len for i in range(10000)
        ))
    #
    # [*res]

    # ==============================< Single PSD >==============================
    # file = "edata/R6D4E/HDF.D/AM.R6D4E.00.HDF.D.2025.260"
    # get_reduced_noise(file, 10. * 60)

    # ==============================< Parallel PSD >==============================
    # w_len = 10. * 60
    # import concurrent.futures
    # files = [f"edata/R6D4E/HDF.D/{f}" for f in os.listdir("edata/R6D4E/HDF.D")]
    # with concurrent.futures.ProcessPoolExecutor(12) as executor:
    #     res = executor.map(get_reduced_noise, files, (w_len for i in range
        (10000)))
    #
    # [*res]

    # ============================< Noise analysis >============================
    w_len = 10. * 60
    # v1, a1, n1, f1 = get_noise_data(UTC("2025-260"), UTC("2025-266:12:00"),
        w_len)
    # v2, a2, n2, f2 = get_noise_data(UTC("2025-267"), UTC("2025-280:12:00"),
        w_len)
    # v3, a3, n3, f3 = get_noise_data(UTC("2025-281"), UTC("2025-297:12:00"),
        w_len)
    #
    # plt.polar(a1, v1, "o")
    # plt.polar(a2, v2, "o")
    # plt.polar(a3, v3, "o")
    # # ax = windrose.WindroseAxes.from_ax()
    # # ax.bar(np.degrees(a1), v1)
    # plt.show()
    #
    # freq = 5
    # idx = np.argmin(np.abs(f1 - 5))
    # freq = f1[idx]
    # plt.title(f"Noise level at f = {freq} Hz")
    # plt.semilogy(v1, [i[idx] for i in n1], "o", label = "Bare")
    # plt.semilogy(v2, [i[idx] for i in n2], "o", label = "Hose")
    # plt.semilogy(v3, [i[idx] for i in n3], "o", label = "Tent")
    # plt.legend()
    # plt.xlabel("Wind speed (m/s)")
```

```
    # plt.ylabel("Power Spectral Density (Pa^2 Hz$^{-1}$")
    # plt.show()

    # ========================< Noise - Freq analysis >=========================
    # w_len = 10. * 60
    # v_range = (2, 3)
    # a_min, a_max = [-1, 0.2]
    # labels = ["Bare sensor", "Hose filter", "Dome filter"]
    # for (start, end), label in zip(itertools.pairwise([260, 267, 281, 298]), labels):
    #     v, a, n, f = get_noise_data(UTC(f"2025-{start}"), UTC(f"2025-{end - 1}:12:00"), w_len, False)
    #     mask = (v_range[0] < np.array(v)) & (np.array(v) < v_range[1]) & (a_min <= np.array(a)) & (np.array(a) <= a_max)
    #     if not mask.any():
    #         continue
    #     n = np.array(n)[mask]
    #     means = n.mean(axis = 0)
    #     p95 = np.percentile(n, 95, axis = 0)
    #     p05 = np.percentile(n, 5, axis = 0)
    #     std = np.std(n, axis = 0)
    #     plt.semilogy(f, means, label = label)
    #     # plt.fill_between(f, means - std, means + std, alpha = 0.1)
    #     plt.fill_between(f, p05, p95, alpha = 0.1)
    # plt.title("Frequency vs Noise")
    # plt.xlabel("Frequency [Hz]")
    # plt.ylabel("Power Spectrum [Pa$^2$]")
    # plt.gca().set_ylim(ymin = 1e-5)
    # plt.gca().set_xlim(xmin = 0, xmax = 50)
    # plt.legend()
    # plt.savefig("figures/Noise_freq_comparison_all")
    # plt.show()

    # =================< Noise - Freq analysis | Directional >==================
    w_len = 10. * 60
    labels = ["Bare sensor", "Hose filter", "Dome filter"]
    a_bins = [(315, 360), (45, 90)]
    v_range = (2, 3)
    for a_bin in a_bins:
        for (start, end), label in zip(itertools.pairwise([260, 267, 281, 298]), labels):
            v, a, n, f = get_noise_data(UTC(f"2025-{start}"), UTC(f"2025-{end - 1}:12:00"), w_len, False)
            mask = (v_range[0] < np.array(v)) & (np.array(v) < v_range[1]) & (np.degrees(a) % 360 > a_bin[0]) & (np.degrees(a) % 360 < a_bin[1])
            if not mask.any():
                continue
            n = np.array(n)[mask]
            means = n.mean(axis = 0)
            p95 = np.percentile(n, 95, axis = 0)
            p05 = np.percentile(n, 5, axis = 0)
```

```
284             std = np.std(n, axis = 0)
285             wind_dir = ["South-West", "East"][a_bins.index(a_bin)]
286             plt.semilogy(f, means, label = f"{label} - {wind_dir}")
287             # plt.fill_between(f, p05, p95, alpha = 0.1)
288 
289     plt.title("Frequency vs Noise")
290     plt.xlabel("Frequency [Hz]")
291     plt.ylabel("Power Spectrum [Pa$^2$]")
292     plt.gca().set_ylim(ymin = 1e-5)
293     plt.gca().set_xlim(xmin = 0, xmax = 50)
294     plt.legend()
295     plt.savefig("figures/noise_direction.png")
296     plt.show()
297 
298     # ==================< Noise - Freq analysis | Wind speed >==================
299     # w_len = 10. * 60
300     # labels = ["Bare sensor", "Hose filter", "Dome filter"]
301     # v_bins = [*itertools.pairwise([0, 1, 2, 3, 4, 5])]
302     # a_min, a_max = [-1, 0.2]
303     # for (start, end), label in zip(itertools.pairwise([260, 267, 281, 298]),
        labels):
304     #     for v_bin in v_bins:
305     #         # if label != "Dome filter":
306     #         #     continue
307     #         v, a, n, f, std = get_noise_data(UTC(f"2025-{start}"), UTC(f"2025-{
        end - 1}:12:00"), w_len, False)
308     #         mask = (v_bin[0] < np.array(v)) & (np.array(v) < v_bin[1]) & (a_min
        <= np.array(a)) & (np.array(a) <= a_max) & (std >= 2)
309     #         if not mask.any():
310     #             print("Mask empty")
311     #             continue
312     #         n = np.array(n)[mask]
313     #         means = n.mean(axis = 0)
314     #         p95 = np.percentile(n, 95, axis = 0)
315     #         p05 = np.percentile(n, 5, axis = 0)
316     #         # std = np.std(n, axis = 0)
317     #         plt.semilogy(f, means, label = f"{label} - {v_bin[0]}-{v_bin[1]} [m/
        s]")
318     #         # plt.scatter(a, v)
319     #         # plt.fill_between(f, p05, p95, alpha = 0.1)
320     #
321     #     plt.title("Frequency vs Noise")
322     #     plt.xlabel("Frequency [Hz]")
323     #     plt.ylabel("Power Spectrum [Pa$^2$]")
324     #     plt.legend()
325     #     plt.gca().set_ylim(ymin = 1e-5)
326     #     plt.gca().set_xlim(xmin = 0, xmax = 50)
327     #     plt.savefig(f"figures/noise_windspeed_{label.split(' ')[0].lower()}.png
        ")
328     #     plt.show()
329 
```

```
330     # =========< Noise - Freq analysis | Wind speed with angle filter >=========
331     # w_len = 10. * 60
332     # labels = ["Bare sensor", "Hose filter", "Dome filter"]
333     # v_bins = itertools.pairwise([0, 1, 2, 3, 4, 5])
334     # a_min, a_max = [-1, 0.2]
335     # # a_min, a_max = [0.2, 10.5]
336     # # a_min, a_max = [0.3, 1.8]
337     # for v_bin in v_bins:
338     #     for (start, end), label in zip(itertools.pairwise([260, 267, 281, 298]),
            labels):
339     #         if label != "Hose filter":
340     #             continue
341     #         v, a, n, f = get_noise_data(UTC(f"2025-{start}"), UTC(f"2025-{end -
            1}:12:00"), w_len, False)
342     #         mask = (v_bin[0] < np.array(v)) & (np.array(v) < v_bin[1]) & (a_min
            <= np.array(a)) & (np.array(a) <= a_max)
343     #         if not mask.any():
344     #             continue
345     #         n = np.array(n)[mask]
346     #         means = n.mean(axis = 0)
347     #         p95 = np.percentile(n, 95, axis = 0)
348     #         p05 = np.percentile(n, 5, axis = 0)
349     #         std = np.std(n, axis = 0)
350     #         plt.semilogy(f, means, label = f"{label} - {v_bin[0]}-{v_bin[1]} m/s
            ")
351     #         # plt.scatter(np.array(a)[mask], np.array(v)[mask])
352     #         # plt.fill_between(f, p05, p95, alpha = 0.1)
353     #
354     # plt.title("Frequency vs Noise")
355     # plt.xlabel("Frequency [Hz]")
356     # plt.ylabel("Power Spectrum [Pa$^2$]")
357     # plt.legend()
358     # plt.show()
359     #
360     # ============================< Random shit idk >============================
361     # for d1, d2 in itertools.pairwise(range(260, 267)):
362     # for d1, d2 in itertools.pairwise(range(267, 280)):
363     # # for d1, d2 in itertools.pairwise(range(260, 267)):
364     #     try:
365     #         v, a, n, f = get_noise_data(UTC(f"2025-{d1}"), UTC(f"2025-{d2}"),
            w_len)
366     #         plt.polar(a, v, "o", label = f"{d1} - {d2}")
367     #     except: pass
368     # plt.legend()
369     # plt.show()
370
371     # ============================< Wind rose plots >============================
372     # import matplotlib
373     # matplotlib.rcParams.update({'font.size': 18})
374     # for (start, end), label in zip(itertools.pairwise([260, 267, 281, 298]),
            labels):
```

```
375     #     v, a, n, f = get_noise_data(UTC(f"2025-{start}"), UTC(f"2025-{end -
            1}:12:00"), w_len)
376     #     ax = windrose.WindroseAxes.from_ax()
377     #     ax.bar(180 - np.degrees(a), v)
378     #     plt.title(f"Wind directions during setup: {label}")
379     #     ax.legend()
380     #     fig = plt.gcf()
381     #     # fig.set_size_inches(4, 4)
382     #     plt.savefig(f"figures/Windrose_{label.split(' ')[0].lower()}.png")
383     #     plt.show()
```

StationAnalyse.py

The StationAnalyse.py file contained the code for comparing various infrasound station layouts regarding their detectability of various network layouts.

```
1  import folium
2  from obspy import UTCDateTime as UTC
3  import obspy
4  import math
5  import pandas as pd
6  import numpy as np
7  import matplotlib.pyplot as plt
8
9  from lib import config
10 import Event
11 import convert
12 import windrose
13 from collections import Counter
14
15 def entryPlot(startDate: UTC = UTC(0), endDate: UTC = UTC(), include_inactive:
       bool = True):
16     global m
17     m = folium.Map(location = (48.858153, 2.3409807))
18
19     inv = obspy.read_inventory("./data/inventory.xml")
20     stationsLayer = folium.FeatureGroup("Stations").add_to(m)
21     radiusLayer = folium.FeatureGroup("Station Radius Overlay").add_to(m)
22
23     stationLocs = get_fripon_locs(include_inactive)
24     stations = [*stationLocs.keys()]
25
26     eventLayer = folium.FeatureGroup("Events").add_to(m)
27     stationEventLayers = {station: folium.FeatureGroup(f"Events: {station}", show
           = False).add_to(m) for station in stations}
28
29     # stationLocs = {station: config.get_station_location(station, inv, endDate)
           for station in stations}
30
31     for station in stations:
32         location = stationLocs[station]
```

```
        folium.Marker(
            location = location[:2],
            tooltip = station,
            icon = folium.Icon(icon = "glyphicon-dashboard"),
            ).add_to(stationsLayer)
        folium.Circle(
            location = location[:2],
            radius = 290e3, # Approx sqrt(300^2 - 80^2)
            color = "dark_green",
            opacity = 0.7,
            fill_color = "green",
            fill_opacity = 0.1,
            ).add_to(radiusLayer)

        folium.Marker(
            location = location[:2],
            tooltip = station,
            icon = folium.Icon(icon = "glyphicon-dashboard", color = "red"),
            ).add_to(stationEventLayers[station])
        folium.Circle(
            location = location[:2],
            radius = 290e3, # Approx sqrt(300^2 - 80^2)
            color = "dark_green",
            opacity = 0.7,
            fill_color = "green",
            fill_opacity = 0.1,
            ).add_to(stationEventLayers[station])


    events = Event.get_events()
    for i, event in events.iterrows():
        if not isinstance(event["Duration"], str):
            continue
        if startDate <= UTC(event["Multi-eventx"]) <= endDate:
            event_loc = Event.get_event_loc(event)
            event_xyz = [convert.to_cartesian(*i) for i in event_loc]
            folium.PolyLine(
                [i[:2] for i in event_loc],
                ).add_to(eventLayer)

            for station in stations:
                continue
                station_loc = stationLocs[station]
                station_xyz = convert.to_cartesian(*station_loc)
                dist = Event.get_dist(station_xyz, *event_xyz)
                angle = Event.get_angle(station_xyz, *event_xyz)
                folium.PolyLine(
                    [i[:2] for i in event_loc],
                    color = "red" if dist > 300e3 else "blue" if abs(angle - 90) <
                        0.1 else "orange" if abs(angle - 90) < 25 else "red",
```

```
                    opacity = 0.45 if dist > 300e3 or abs(angle - 90) >= 25 else
                        1,
                    ).add_to(stationEventLayers[station])


    folium.LayerControl().add_to(m)
    m.show_in_browser()

def get_fripon_locs(include_inactive: bool = True) -> dict[str, config.tupLocation
    ]:
    df = pd.read_csv("data3/StationSearch.csv")
    out = {}
    for i, row in df.iterrows():
        if include_inactive or row["Statusx"] == "Production":
            out[row["Station codex"]] = config.tupLocation(
                config.to_float(row["Latitudex"]),
                config.to_float(row["Longitudex"]),
                config.to_float(row["Elevationx"]))
    return out

def get_event_status(event, friponLocs = None, threshold = 300e3):
    if friponLocs is None:
        friponLocs = get_fripon_locs()
    bins = [[], [], [], [], []]
    event_loc = Event.get_event_loc(event)
    event_xyz = [convert.to_cartesian(*p) for p in event_loc]

    for station, station_loc in friponLocs.items():
        station_xyz = convert.to_cartesian(*station_loc)
        dist = Event.get_dist(station_xyz, *event_xyz)
        if dist > threshold:
            continue
        angle = Event.get_angle(station_xyz, *event_xyz)
        for i, bin_angle in enumerate([55, 70, 110, 125, 180]):
            if angle < bin_angle:
                bins[i].append(dist)
                break
    return bins

def get_events_statistics(threshold = 300e3, include_inactive: bool = True):
    events = Event.get_events()
    friponLocs = get_fripon_locs(include_inactive)

    out = []
    for i, event in events.iterrows():
        bins = get_event_status(event, friponLocs, threshold)
        out.append(bins)
    return np.array([[len(bin) for bin in bins] for bins in out])

def get_nl_events():
    nl_events = pd.read_csv("data3/MES_NL.csv")
```

```
131     idxs = set(nl_events["IDx"].values)
132     for idx, row in Event.get_events().iterrows():
133         if row["idx"] in idxs:
134             yield Event.EntryEvent.from_row(row)
135 
136 def dir_plot(events, title = ""):
137     ax = windrose.WindroseAxes.from_ax()
138     ax.bar([ev.heading for ev in events], [1 for ev in events])
139     plt.gcf().suptitle(title)
140     plt.show()
141 
142 # ============================< Station evaluation >============================
143 def generate_counts(event: Event.EntryEvent, stations: list[Event.Station],
        max_dist: float = 300e3) -> list[int]:
144     """
145     Generates a list of integers based on the given stations in the form (a, b, c,
            d).
146     a: int = The number of stations in the ballistic regime
147     b: int = The number of stations in the transitionary regime
148     c: int = The number of stations in the bow shock regime
149     d: int = The number of stations in the rear regime
150     """
151     result = [0, 0, 0, 0]
152     for station in stations:
153         dist = event.get_dist(station)
154         if dist > max_dist:
155             continue
156         angle = event.get_angle(station)
157         if angle <= 55:
158             result[2] += 1
159         elif angle <= 70:
160             result[1] += 1
161         elif angle <= 110:
162             result[0] += 1
163         else:
164             result[3] += 1
165     return result
166 
167 def classify_counts_transient(counts: list[int]) -> str:
168     """
169     Classifies the counts with a focus on understanding the transient shock regime
170     Returns any of the following strings
171     'Full range'
172     'Ballistic-transient'
173     'Bow-transient'
174     'None'
175     """
176     if counts[0] and counts[1] and counts[2]:
177         return "Full range"
178     elif counts[0] and counts[1]:
179         return "Ballistic-transient"
```

```
180     elif counts[2] and counts[1]:
181         return "Bow-transient"
182     else:
183         return "None"
184 
185 def classify_counts(counts: list[int]) -> str:
186     """
187     Classifies the counts with a focus on improving the understanding of
            detectability in the "known" regimes
188     """
189     if counts[0] and counts[2]:
190         return "Bow + ballistic"
191     elif (counts[0] or counts[2]) and counts[1]:
192         return "Known + transient"
193     elif counts[0]:
194         return "Ballistic only"
195     elif counts[2]:
196         return "Bow only"
197     elif any(counts):
198         return "Other"
199     else:
200         return "None"
201 
202 def get_counts(max_dist: float = 300e3) -> dict[str, list[int]]:
203     global base_stations, candidate_stations
204     events = [*get_nl_events()]
205     for event in events:
206         for station in base_stations:
207             event.add_station(station, max_dist)
208     all_counts = {}
209     for station in candidate_stations:
210         all_counts[station] = counts = []
211         for event in events:
212             counts.append(event.if_station(station, max_dist))
213 
214     for station, counts in all_counts.items():
215         counter = Counter(counts)
216         print(r"\textbf{" + f"{station.name}" + r"} & " + " & ".join(f"{counter[i
                ]}" for i in range(4)) + r" \\ \hline")
217 
218     return all_counts
219 
220 def analyse_events(stations, max_dist: float = 300e3):
221     result = []
222     for event in get_nl_events():
223         result.append(classify_counts(generate_counts(event, stations, max_dist)))
224     return result
225 
226 # ==============================< Visualisation >================================
227 def print_table(max_dist: float = 300e3):
228     """
```

```
229     Prints a LaTeX table for detection types for each candidate station location.
230     """
231     options = ["Bow + ballistic", "Known + transient", "Ballistic only", "Bow only
            ", "Other", "None"]
232     print(r"\textbf{Station location} & " + " & ".join(r"\textbf{" + f"{option}" +
             r"}" for option in options) + r" \\ \hline")
233 
234     result = analyse_events(base_stations, max_dist)
235     counter = Counter(result)
236     print(r"\textbf{Baseline} & " + " & ".join(f"{counter[j]}" for j in options) +
             r" \\ \hline")
237 
238     for station in candidate_stations:
239         stations = base_stations + [station]
240         result = analyse_events(stations, max_dist)
241         counter = Counter(result)
242         print(r"\textbf{" + station.name + r"} & " + " & ".join(f"{counter[j]}"
                for j in options) + r" \\ \hline")
243 
244 def count_bars(counts: dict[str, list[int]], dist = None):
245     plt.gcf().set_size_inches((8.4, 4.8))
246     stations = [*counts.keys()]
247     s_names = [s.name for s in stations]
248     bottom = np.zeros(len(stations), np.int64)
249     for i in reversed(range(max(max(counter) for counter in counts.values()) + 1))
            :
250         row = np.array([Counter(counts[s])[i] for s in stations])
251         print(row)
252         plt.bar(s_names, row, bottom = bottom, label = f"{i} Detections")
253         for s, b, c in zip(s_names, bottom, row):
254             if c:
255                 pass
256                 plt.text(s, b + c/2, f"{c}", ha = "center", va = "center", c = "
                        white")
257         bottom += row
258     plt.xlabel("Station location")
259     plt.ylabel("Potential detections counts")
260     plt.title(f"Potential meteor infrasound detection counts at ${dist}$ [km]")
261     plt.legend()
262     plt.tight_layout()
263     if dist is not None:
264         plt.savefig(f"figures/DetectionCounts_{dist}.png")
265     plt.show()
266 
267 # ==================================< Config >==================================
268 candidate_locs = {
269     "NIOZ (Texel)": (53.005, 4.7935),
270     "Radboud Nijmegen": (51.822, 5.864),
271     "Geosciences\n Utrecht University": (52.088, 5.167),
272     "Elburg": (52.448, 5.843),
273     "Franeker": (53.190, 5.540),
```

```
274     }
275 base_locs = {
276     "TUD": (51.989, 4.376),
277     "Tilburg": (51.563, 5.042),
278     }
279 
280 candidate_stations = [Event.Station(key, val) for key, val in (candidate_locs).
        items()]
281 base_stations = [Event.Station(key, val) for key, val in (base_locs).items()]
282 stations = [Event.Station(key, val) for key, val in (base_locs | candidate_locs).
        items()]
283 
284 for station in stations:
285     if station.name in base_locs:
286         station.colour = "red"
287 if __name__ == "__main__":
288     ev = next(get_nl_events())
289     nl_evs = [*get_nl_events()]
290     all_evs = [Event.EntryEvent.from_row(row) for idx, row in Event.get_events().
            iterrows()]
291     # dir_plot(all_evs, "All")
292     # dir_plot(nl_evs, "NL")
293     # dist = 300e3
294     # for name, latlon in base_locs.items():
295     #     for ev in nl_evs:
296     #         ev.add_station(latlon, dist)
297     # for i, (name, latlon) in enumerate(candidate_locs.items()):
298     #     counts = Counter([ev.if_station(latlon, dist) for ev in nl_evs])
299     #     plt.bar([j + i/10 for j in counts.keys()], [*counts.values()], label = f
            "{name}", alpha = 0.2)
300     # plt.legend()
301     # plt.show()
302 
303 
304     # friponLocs = get_fripon_locs()
305     # # entryPlot()
306     # # entryPlot(startDate = UTC("2021/01/01"), endDate = UTC("2022/01/01"),
            include_inactive = True)
307     # # events = Event.get_events()
308     # # event = [*events.iterrows()][0][1]
309     # # res = get_event_status(event)
310     # # print(res)
311     # fig, axs = plt.subplots(5, 1, figsize = (7.5, 6))
312     # for thr in [100e3, 150e3, 200e3, 250e3, 300e3]:
313     #     res = get_events_statistics(thr, False)
314     #     for i, ax in enumerate(axs):
315     #         y = [(res[:, i] == j).sum() for j in range(res[:, i].max() + 1)]
316     #         ax.plot(y, label = f"Range: {thr / 1e3:.0f} [km]")
317     # plt.legend()
318     # plt.show()
319     from Visualisation.EventMap import EventMap
```

```
320     # EventMap(stations, nl_evs)
321
322     # for dist in [100e3, 200e3, 300e3]:
323     #     print(f"Maximum distance: {dist / 1e3:.0f} [km]")
324     #     result = get_counts(dist)
325     #     for station, counts in result.items():
326     #         plt.plot([(np.array(counts) >= i).sum() for i in range(4)], label =
        station.name)
327     #     plt.title(f"Detection counts at a maximum distance of {dist / 1e3:.0f} [
        km]")
328     #     plt.xlabel("Minimum detection count")
329     #     plt.ylabel("Fireball events")
330     #     plt.legend()
331     #     plt.show()
332     count_bars(get_counts(200e3), 200)
333     count_bars(get_counts(300e3), 300)
334
335     # print("200 km")
336     # print_table(200e3)
337     #
338     # print("300 km")
339     # print_table(300e3)
```