BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

DISTRIBUTED ACOUSTIC SENSING SYSTEM DATA ANALYSIS APPLIED FOR PERIMETER PROTECTION

ANALÝZA DAT Z OPTICKÉHO DISTRIBUOVANÉHO AKUSTICKÉHO SENZORICKÉHO SYSTÉMU PRO OCHRANU PERIMETRU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. Jakub Senčák

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Adrián Tomašov

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Information Security**

Department of Telecommunications

Student: Bc. Jakub Senčák ***ID:*** 196504

Year of study: 2 ***Academic year:*** 2022/23

TITLE OF THESIS:

Distributed acoustic sensing system data analysis applied for perimeter protection

INSTRUCTION:

The thesis focuses on distributed acoustic sensing (DAS) system data in HDF5 format. The semestral part of the thesis studies DAS principles and the output format of the deployed system, followed by an implementation of conversion from data to audio signal (in WAV format). This part also investigates the possibilities of real-time data analysis. The diploma thesis proposes a graphical user interface displaying real-time data with various chart manipulation methods. The final program is tested with a real DAS system deployed onto the faculty optical sensoric polygon.

RECOMMENDED LITERATURE:

[1] PARKER, Tom; SHATALIN, Sergey; FARHADIROUSHAN, Mahmoud. Distributed Acoustic Sensing–a new tool for seismic applications. first break, 2014, 32.2.

[2] COLLETTE, Andrew. Python and HDF5: unlocking scientific data. " O'Reilly Media, Inc.", 2013.

Date of project specification: 6.2.2023 ***Deadline for submission:*** 19.5.2023

Supervisor: Ing. Adrián Tomašov

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

This work focuses on fiber optic sensing using distributed acoustic sensing. The goal was to create a multiplatform application for data visualization obtained by an Optasense ODH-F interrogator. This work was motivated by the lack of open-source visualization software for fiber optic sensing. Distributed acoustic sensing was explained with light scattering effects such as Rayleigh, Raman, and Mandelsam-Brillouin scattering. A web application was implemented. There are two parts, back-end was written in Python, and the frontend was written using the Svelte framework. The results of this work can be used as a basis for creating visualization software for distributed acoustic sensing.

KEYWORDS

fiber optics, distributed acoustic sensing, optical time domain reflectometry, strain and vibration measurement, visualization software, web application, Svelte framework

ABSTRAKT

Táto práca sa zameriava na snímanie optických vlákien pomocou distribuovaného akustického snímania. Cieľom bolo vytvoriť multiplatformovú aplikáciu na vizualizáciu údajov získaných zariadení Optasense ODH-F. Táto práca bola motivovaná nedostatkom aplikácií s otvoreným zdrojovým kódom na vizualizáciu dát získaných pri snímaní pomocou optických vlákien. Práca vysvetľuje distribuované akustické snímanie a efekty rozptylu svetla, ako je Rayleighov, Ramanov a Mandelsamov-Brillouinov rozptyl. Výsledkom práce je webová aplikácia. Aplikácia sa skladá z dvoch hlavných častí - server napísaný v jazyku Python a klientskej časti implementovanej vo frameworku Svelte. Výsledky tejto práce možno použiť ako základ pre vytvorenie vizualizačného softvéru pre distribuované akustické snímanie.

KĽÚČOVÉ SLOVÁ

optické vlákna, distribuované akustické snímanie, optická reflektometria v časovej oblasti, meranie vibrácií, vizualizačný softvér, webová aplikácia, Svelte framework

ROZŠÍRENÝ ABSTRAKT

Optické vlákna sa v dnešnej dobe používajú hlavne na prenos informácií a to medzi datacentrami alebo až k nám domov. Spolu s vývojom optických vlákien a laserových diód začínajú vznikať aj ďalšie odvetvia. Vzniká optická reflektometria a jej rôzne metódy ako OTDR (optická reflektometria v časovej oblasti) a OFDR (optická reflektometria vo frekvenčnej oblasti). Tie sa používajú na meranie rôznych defektov vlákien - praskliny, zlomy, pretrhnutia a zvary. Meranie funguje na princípe vyslania svetelného pulzu do optického vlákna a pri prechode vláknom sa svetlo rôzne láme a odráža od centier rozptylu (z angl. scattering center). Centrá rozptylu môžu byť rôzne nečistoty, nerovnomernosti vo vlákne ako aj samotné molekuly a atómy materiálu, z ktorého je vlákno vyrobené.

Podľa toho, od akých centier rozptylu sa svetlo odráža, rozlišujeme rôzne efekty odrazu svetla. Mie efekt je spôsobený odrazom od nečistôt vo vlákne väčších ako je vlnová dĺžka signálu z laserovej diódy. Rayleighov odraz je zase spôsobený odrazmi od atómov a molekúl a všeobecne od centier menších než je vlnová dĺžka svetelného paprsku. Ramanov odraz je odrazom od kryštalickej mriežky materiálu, z ktorého je materiál vyrobený. Mandelsam-Brillouinov odraz je odraz od vibrácií atómov a molekúl v materiáli.

Distribuované akustické snímanie *DAS* (z anglického Distributed Acoustic Sensing) umožňuje využiť tieto odrazy na merania rôznych externých veličín. Zaujímavosťou je, že tieto merania sa môžu vykonávať distribuovane. To znamená, že na viacerých miestach na celej dĺžke vlákna dochádza k meraniam. Toho sa dosiahne meraním časového rozdielu medzi vyslaním svetelného pulzu do vlákna a časom, keď svetelný odraz dorazí do fotodetektora. Merajú sa rôzne zmeny vo vlastnostiach vyslaného svetelného pulzu, ako zvýšená alebo znížená vlnová dĺžka, útlm výkonu amplitúdy a zmeny vo fáze odrazeného signálu oproti signálu vyslaného. Základnou vlastnosťou na to aby mohol byť odrazový efekt použitý pre distribuované snímanie je rovnomerné rozloženie odrazových centier po celej dĺžke vlákna.

Rôzne odrazové efekty sa používajú na rôzne účely. Napríklad na meranie okolitej teploty sa hodí najlepšie Mandelsam-Brillouinov odraz, pretože vzniká pri odrazoch od vibrujúcich atómov. Vo všeobecnosti DAS systémy nachádzajú použitie v detekcii vibrácií, ťahu a detekcii pohybu. Napríklad v optických gyroskopoch a akcelerometroch, kde ich najväčšou výhodou je vysoká presnosť a minimálny drift. Aktívne sa používajú v detekcii a lokalizácii zemetrasení, ochrane perimetru, aktívnom monitorovaní budov, stavieb a mostov. Ďalšie zaujímavé využitia sú v biosenzoroch a v chémii pri zisťovaní rôznych vlastností chemikálií. Výhodou optických vlákien je ich vysoká odolnosť voči nepriaznivým externým javom a nízka ovplyvniteľnosť elektromagnetickým šumom. To z nich robí ideálne médium pre nebezpečné prostredia, ako sú vysokorádioaktívne prostredia, nebezpečné chemi-

kálie a vysoké napätia. Ďalšou výhodou je malý rozmer optického vlákna. Preto sa využívajú ako súčasť monitorovania vrtov v ropnom priemysle. Možnosti použitia tejto technológie sú naozaj široké.

Zariadenie schopné merať signál v optických vláknach sa nazýva DAS interrogator. Asi najbližší preklad do slovenčiny je "vyšetrovateľ". Tento názov sa veľmi nehodí preto budeme ďalej používať *DAS systém* alebo len zariadenie. V našom prípade boli merania vykonávané na zariadení OptaSense ODH-F. Zariadenie sa pripojí na optické vlákno z jednej strany. Použité optické vlákno môže byť buď špeciálne vlákno určené na merania alebo už existujúca optická infraštruktúra. Zariadenie umožňuje rôzne nastavenia merania, vzorkovacie frekvencie, nastavenie svetelného pulzu, nastavenia GPS lokácie a ďalšie.

Výstupom meraní je súbor vo formáte HDF5 (*Hierarchical Data Format v5*), ktorý umožňuje ukladať dáta v podobe podobnej Linuxovému súborovému systému. HDF5 súbor je založený na modele HDF5, ktorý definuje základné súčasti súboru a jeho štruktúry, ale samotná štruktúra a rozloženie stanovuje systém alebo vývojár. Tieto dáta zo zariadenia je možné v reálnom čase zobrazovať pomocou aplikácie OptaSense OS6. Aplikácia umožňuje zobrazovať horný pohľad na sledovanú oblasť a udalosti, ktoré sa detekujú na jednotlivých miestach okolo vlákna.

Cieľom tejto práce bolo vytvoriť aplikáciu na zobrazovanie dát z DAS systému, ktorá bude multiplatformná, keďže jediný softvér na zobrazovanie dát z tohoto systému je proprietárny a nie je možné ho upravovať. Z tohto dôvodu sme vybrali dizajn aplikácie ako webovú aplikáciu. Aplikáciu sme rozdelili na dve hlavné časti a to na klienta, ktorý bude spustený v prehliadači a serverovú časť, ktorá posiela dáta do klientskej časti.

Klientská časť aplikácie je postavená na frameworku Svelte, ktorý kompiluje celý projekt do čistého JavaScriptu. Pre porovnanie, frameworky ako React a Vue pracujú s virtuálnou reprezentáciou webových objektov, zatiaľ čo Svelte všetky komponenty kompiluje do jedného súboru, ktorý potom vykonáva aktualizácie webovej aplikácie. Týmto spôsobom sa obetuje kompilačný čas za čas pri behu aplikácie. Výsledkom je teda rýchla aplikácia s vysokou reaktivitou. Svelte aplikácie sú takto oveľa rýchlejšie oproti konkurencii. Svelte framework tiež umožňuje pomerne rýchlu implementáciu a nezaťažuje vývojára so zložitými konceptami aktualizácie elementov.

Vizualizácia dát je implementovaná ako tepelná mapa (z anglického heatmap), ktorá sa vykresľuje do Canvas elementu. Hodnoty sú reprezentované v tepelnej mape pomocou farieb. Farby združujeme do farebných máp podľa toho, ako sa menia hodnoty s farbou. Poznáme štyri základné druhy farebných máp. *Sekvenčné* sú buď jednofarebné alebo viacfarebné, pri tomto type zmeny v saturácii alebo svetlosti farby reprezentujú zmeny hodnôt a používajú sa napríklad pri radení. *Rozchádza-*

júce sa mapy obyčajne začínajú v strede jednou farbou a do minima a maxima sa rozchádzajú dve rôzne farby. Používajú sa, ak sú hodnoty okolo jednej strednej hodnoty. *Cyklické* mapy začínajú a končia v rovnakej farbe. Posledným typom sú *kvalitatívne* farebné mapy, ktoré zobrazujú rôzne farby a zobrazujú informácie bez jednoznačného poradia. Užívateľ má možnosť si vybrať farebnú mapu podľa svojej subjektívnej preferencie.

Webová aplikácia umožňuje vybrať dátový súbor otvoriť ho a prehrávať dáta z neho, tak ako boli zaznamenané DAS systémom. Umožňuje tiež pozastaviť prehrávanie a znova ho spustiť, nastavovať rýchlosť prehrávania a exportovať zobrazené dáta do formátu PNG.

Na komunikáciu klient-server sme navrhli jednoduchý protokol. Komunikácia prebieha pomocou WebSocketov tak, že si klient so serverom vymieňajú bezstavové informácie.

Serverová časť aplikácie zabezpečuje čítanie a spracovanie súborov. A po tom, čo si užívateľ zvolí súbor, ktorý chce zobrazovať v klientskej časti, sa súbor načíta a to tak, aby príliš nezaťažoval operačnú pamäť počítača. Spustí sa predspracovanie dát, pretože nespracované dáta by nezobrazovali informácie pochopiteľným spôsobom a navyše by ich bolo príliš veľa. Predspracovanie vytiahne potrebné informácie z datasetu a tie sa následne uložia do *numpy* súboru, ktorý má už aj prijateľnú veľkosť aj vhodné dáta. Tento súbor sa potom číta a posiela do klientskej časti aplikácie, kde sa dáta zobrazujú.

Výsledkom práce je teda webová aplikácia na vizualizáciu dát zo systému DAS na kontrolu perimetru. Aplikáciu sme otestovali na dátach, ktoré sme zaznamenali pri behaní po chodníku neďaleko zakopaného optického vlákna v perimetri neďaleko školy. Dáta boli nahrané so vzorkovacou frekvenciou 20 kHz. Dáta sme spracovali a zobrazili v nami implementovanej aplikácii.

Pri tom, ako som robil na tejto práci, som sa naučil ako funguje systém DAS, rôzne optické javy, ktoré nastávajú pri odraze svetla v optických vláknach a naštudoval si rôzne použitia tejto technológie v praxi. Pri implementovaní webovej aplikácie som sa naučil framework Svelte a programovať v jazyku JavaScript. Pri implementácii serverovej časti som sa zase naučil používať moduly a knižnice pre dizajn asynchrónnej aplikácie, komunikácie pomocou WebSocketov, používať objekty typu generátor a prácu s HDF5 súbormi.

Táto práca môže slúžiť ako vzor pri implementácii vizualizácií nad dátami z DAS sytémov. Je tiež základom pre prácu s dátami z DAS systému a práci s HDF5 súbormi. Ďalšia práca na tejto aplikácii bude zahŕňať zlepšovanie vizualizačného algoritmu a pridávanie ďalších funkcionalít, ako je zoom, výber dát a zvýrazňovanie udalostí. Nad dátami tiež môže bežať aj analýza pomocou umelej inteligencie, ktorá môže kategorizovať udalosti, ktoré sa stali pozdĺž optického vlákna.

SENČÁK, Jakub. *Distributed acoustic sensing system data analysis applied for perimeter protection*. Brno: Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 82 p. Master's Thesis. Advised by Ing. Adrián Tomášov

Author's Declaration

Author:	Bc. Jakub Senčák
Author's ID:	196504
Paper type:	Master's Thesis
Academic year:	2022/23
Topic:	Distributed acoustic sensing system data analysis applied for perimeter protection

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I want to thank my supervisor Ing. Adrián Tomášov for leading me during my struggles with this work, for his time during our consultations, and for lots of patience.

Contents

List of Figures

List of Tables

Introduction

With the discovery of laser diodes and optical fibers, the data transmission lines increased the throughput drastically. It enabled fast and reliable communication between data centers and later into our homes. Optical reflectometry enabled to study the inner structure of the fiber itself, the joints, cracks, and imperfections in the fiber. Further research discovered that optical fibers can also be used as sensors. Analyzing the signal coming back from the fiber using optical reflectometry to measure different external properties. And further, by clever timing, it is possible to measure these properties in multiple places along the wire, thus creating distributed sensing.

The work explains the topic of DAS (Distributed Acoustic Sensing), which uses optical fiber as a sensor array. Light pulses are sent from the light source through the fiber. The light is reflected and scattered on imperfections in the fiber and is reflected back to the light source, where its properties are measured, like changes in frequency and phase. If there is a strain on the fiber or the fiber is subjected to vibrations of any type, it is possible to interpret them as an audio signal or some movement. DAS is used for applications such as perimeter monitoring, earthquake detection and localization, traffic monitoring and incident detection, and many more. One of the uses is the possibility to hear, interpret the data as an audio signal, and use it as a microphone, which makes optical fiber a big security vulnerability. Especially dangerous is that the attackers do not need access to the server room or the devices but can connect to the fiber anywhere. There is also the issue of detecting these kinds of attacks because DAS does not impact existing communication on the fiber.

The main goal of this work is to implement an application that takes the data in HDF5 file format and converts it to WAV audio file format. The second goal is to design an application for displaying the data in a waterfall graph. The design includes studying existing technology and technology capable of displaying the data in real-time.

The first chapter explains fiber optic sensing in general and introduces different light scattering effects, like Rayleigh and Raman scattering. It also explains the principles in the field of optical reflectometry and explains how the DAS system works, and the methods for measuring the strain on the fiber. An important part is studying the HDF5 file format and especially the output of the DAS system. The second chapter explains the software design of the data visualization application and the important technologies that make it - WebSockets, Svelte framework, and HTML rendering options using Canvas and SVG graphics. The last chapter covers the implementation. It focuses on the most interesting parts of implementing client-server communication, Svelte components, data processing, and data visualization.

1 Fiber-optic sensing

Light has revolutionized data transmission and made high data rates possible using optical fibers and laser diodes. Apart from the intended usage, data transmission lines made from optical fibers have a new use case in sensing. The optical fiber can be used to measure useful external properties thanks to the detection and analysis of different light scattering effects of the interaction between the light and the fiber, as discussed in Section 1.3.

This chapter will discuss fiber optics, optical reflectometry, distributed sensing, and different light scattering effects in the fiber.

1.1 Fiber optic sensors

Fiber optic sensors have three main parts - a *light source*, a *medium* that light passes through, and a *detector*. The principle these sensors use is to generate light at the *source* light (laser diode LD), then passes through the medium, which can be a scanned material or an optical fiber (Section 1.2). The medium affects the light signal and changes signal properties measured at the detector. This way, fiber optic sensors can detect external properties such as vibrations (seismic, acoustic), pressure, acceleration, rotation, and chemical properties.

There are two types of sensors based on the medium used[1]:

- *intrinsic sensors* - the optical fiber is a measuring medium.
- *extrinsic sensors* - use optical fiber to get the signal to and from the actual sensor.

There are two types of optical fiber sensors based on the location of the measurement:

1. **Point** - These sensors measure only at the location of the transducer[2]
2. **Quasi-distributed** - They use many sensors along the fiber to measure
3. **Distributed** - The sensing element is the optical wire. It can measure at thousands of points along the optical wire thanks to different scattering effects, as discussed in Section 1.3. It can use existing telecommunication infrastructure to build the sensing network.

1.1.1 Fiber-optic sensing applications in different fields

The advantage of fiber optic sensing compared to other kinds of sensing is that it is immune to signal interference. The optic fiber is made of glass or transparent plastic.

[1] https://www.rp-photonics.com/fiber_optic_sensors.html

[2] device transforming energy from one form of energy to another form of energy

It can be used in environments that would be dangerous or harsh for other types of sensors. It is good to mention environments such as flammable, explosive, harsh chemicals, high voltage, or environments that would create electromagnetic noise. Thanks to these properties, fiber-optic sensing has many different applications.

The applications include:

- *Fiber-optic gyroscopes* - rotation measurement thanks to the Sagnac effect. They can replace older ring-laser technology [31].
- *Fiber-optic accelerometers* - vibration measurements with added electromagnetic interference immunity [30].
- *Fiber-optic bio-sensors* - thanks to glass fibers' chemical and thermal stability, these sensors are perfect for measuring harsh chemicals. Measurements can be done in hard-to-get or small spaces. They also measure on small sample volumes [8].
- *Vibration detection* - seismic, acoustic, and even underwater.
  - `Seismology` - measuring and locating earthquakes [10].
  - `Building monitoring` - bridge monitoring for changes such as cracks [1].
  - `Perimeter protection` - detecting and localizing intrusion into an area, more in Section 1.1.2.
  - `Location detection` - fiber-optic sensors can detect traffic and vehicles in cities and on highways or locate trains along train tracks [10].
  - `Fiber-optic hydrophones` - under-water detection systems for seismic monitoring [11].

1.1.2 Perimeter security

In the early 2000s, perimeter protection using fiber optics was based on breaking or cutting the fiber, triggering an alarm. This is good enough for one-time use because after the wire is broken, there is no choice but to replace or repair the wire. This system can not tell the location when using a single wire. Newer systems used *Sagnac effect*, which uses a Sagnac interferometer and a closed loop made of fiber optic wires, for example, 3x3[3] wire system. Sagnac interferometer detects changes in the phase of light, and thanks to signal processing and calculating the time difference between amplitudes, the position of an intruder is calculated. Such an interferometer has a conversion unit from optical to electric signal. The electric signal is then sampled using a fast A/D converter with high sampling rates. The accuracy of such a system is 20-50 meters which is more than sufficient for perimeter protection [2].

[3]three by three

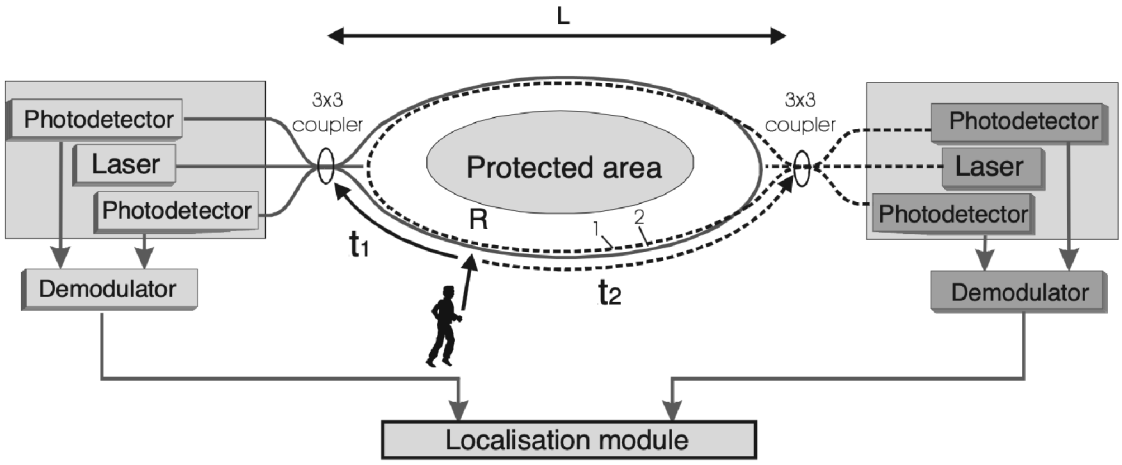

Fig. 1.1: 3x3 system for perimeter protection using two Sagnac interferometers [2].

Thanks to research and technological advancements, new devices based on *Distributed Acoustic Sensing* (DAS) systems are used. These systems use scattering effects that happen in the fiber during the passage of photons through the fiber's medium. These effects are then analyzed at the source of light. The light bounces from imperfections in the fiber and is propagated backward as scattering (*backscattering*). Nothing special happens when the fiber is not moving, but when the fiber is affected in any way, for example, by vibrations of an intruder or just by voice alone, the back-scattering changes. These changes can then be analyzed and categorized as an intrusion.

1.2 Optical fibers

As optical sensing uses existing fiber optic transmission lines, it is important to account for different kinds of optical fibers, materials, and production methods. Optical fibers consist of three main elements *core*, *cladding*, and *coating*. Materials from which core and cladding are made are plastic or glass Si_2O_3.

All-glass fiber dopants, such as GeO_2, P_2O_5, B_2O_3, can be added to all-glass fibers to adjust the refractive index. The core usually has a higher refractive index than cladding by about 1 %. Lowering the refractive index can be done by doping fluorine, which is done in the core when the refracting index is too high and needs to be lowered[4]. The radius of core ranges from 3.7 µm to 200 µm and radius of cladding is up-to 140 µm [5].

Plastic optics uses organic material in the form of polymers - chains. Materials used are acrylic, polycarbonate, polystyrene, or liquid silicone. The core of plastic

[4]https://www.rp-photonics.com/fiber_core.html

fiber has a popular diameter of 980 µm.

Although the purpose of the coating is simply protection, the fiber would be very fragile without it. It is usually without special color but can be painted to ease the identification of individual fibers. There are multiple layers of coating, at least primary and secondary. The primary coating is softer to allow the bending of the fiber. Secondary is harder to protect inner layers. Materials such as acrylate, silicone, polyimide,e or carbon are used depending on the application of the optical fiber. for example. For example, acrylate has limited temperature resistance; in this case, silicone is better as it is heat resistant up to 200 °C[5]. For more extreme applications, Polyimide is used as it can withstand temperatures up to 350 °C, and it is also resistant to chemicals and abrasion [5].

1.3 Light scattering effects in fiber optics

Light precisely photons traveling through a medium - atmosphere, glasses, glass optical fiber, or any other medium can bounce from what is called *scattering centers* in the medium. *Scattering centers* are any non-uniformities in the medium such as vacancy defects (missing atoms in otherwise uniform structure), foreign particles, bubbles, trapped gas molecules, fractures, micro-cracks, any changes in refractive index, density fluctuations, manufacturing imperfections, and others [6]. Scattering centers create different kinds of scattering, as will be discussed in the next sections. They include *Mie scattering* (Section 1.3.1), *Rayleigh scattering* (Section 1.3.2), *Raman scattering* (Section 1.3.3) or *Brillouin scattering* (Section 1.3.4). For comparison, Figure 1.2 shows all of these different scattering effects:

- 1 - input radiation from a laser diode.
- 2 - Rayleigh scattering
- 3 and 4 - Brillouin scattering lines
- 5 and 6 - Raman scattering

1.3.1 Mie scattering

Mie scattering is an optical phenomenon happening when light traveling through a medium bounces from *scattering centers* the same length or bigger than the wavelength of the light. Mie scattering applies to any spherical particles located in the medium. This also applies to the smaller particles. But we distinguish Mie scattering for particles bigger than the wavelength of light for clarity. For particles smaller than the wavelength of light we distinguish a special case of Mie scattering, and we call it *Rayleigh scattering*, which will be discussed in the next Section 1.3.2. That said, there are differences between these two types. The scattered light's amplitudes

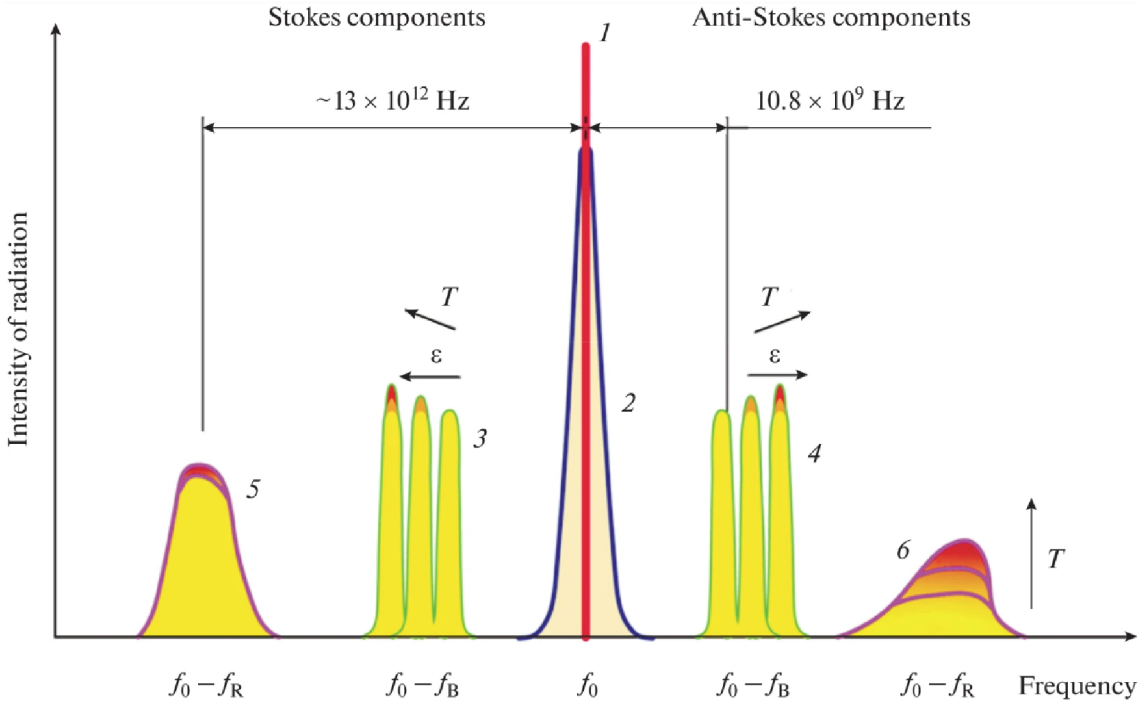

Fig. 1.2: Comparison of different scattering effects [36].

are stronger for forward scattering in Mie scattering[5]. Large defects are usually not uniformly distributed along the fiber, which prevents it from being used in DAS systems [10].

1.3.2 Rayleigh scattering

Rayleigh scattering is an optical phenomenon named after British physicist Lord Rayleigh. Light is scattered from scattering centers much smaller than the wavelength of the light, for example, individual molecules or atoms. This is opposite to Mie scattering 1.3.1, where light is scattering from larger scattering centers. The difference is that amplitudes are the same for forward and backscattering in Rayleigh scattering[6]. Compared to other scattering processes, Rayleigh is *linear* scattering process whereas *Raman* and *Brillouin* scatterings are *nonlinear*[7].

Only a small portion of the back-scattered light returns to the source - most of it leaves the fiber on the sides. Rayleigh scattering is used in DAS, as discussed in Section 2.4.

[5]https://www.rp-photonics.com/rayleigh_scattering.html

[6]https://www.rp-photonics.com/rayleigh_scattering.html

[7]Nonlinear light effects occur when the output intensity does not increase proportionally to the input intensity; for example doubling the optical input intensities does not result in double the output intensity. These nonlinear effects tend to weaken significantly at low optical intensities.

When solidified in a medium, not all scattering centers cause Rayleigh scattering. At a wavelength of about 0.95 µm (microns), glass optical fibers have a high attenuation band caused by scattering and absorption by hydroxide ions [6]. Silica glass is an amorphous material with random density fluctuations due to its irregular microscopic structure. This can be limited by an annealing process but can not be removed completely[8].

1.3.3 Raman scattering

The effect photons have when interacting with the crystal lattice of glass is called *Raman scattering*. A transparent optical medium, such as glass, has a crystal lattice. The lattice is naturally vibrating, causing a delayed nonlinear response to the light passing through it. The photon traveling through the medium experiences a loss in energy due to interactions with the medium. This is also called *inelastic scattering*. This is further explained in the next Section 1.3.4.

Raman scattering can be measured by sending two light waves with different wavelengths through the optical medium. The signal with longer wavelength experiences optical amplification at the expense of the one with a shorter wavelength. This is used in Raman lasers, Raman amplifiers, or Raman spectroscopy[9].

1.3.4 Brillouin scattering

Also known as *Mandelsam-Brillouin scattering*, it was first described by Raman in the 1920s. Brillouin scattering is a scattering effect created when light traveling through a medium is scattered during interaction with thermal vibrations of these molecules. As described earlier, it is very similar to Raman scattering. The intensity is much lower than Rayleigh scattering; see Section 1.3.2. But at the same time, much stronger than Raman scattering. For comparison, see Figure 1.2.

The difference between Raman and Brillouin scattering is in the type of interactions with vibrations of the crystal lattice and molecules. To describe the fundamental quanta of lattice vibrations involved in these interactions, we use the term *phonons*.

There are two types of phonons:

- *acoustic phonons* - associated with backward Brillouin scattering; show linear dispersion relation in bulk.
- *optical phonons* - associated with Raman scattering relate to molecular vibrations; have flat dispersion. The forward Brillouin scattering has similar phonon dispersion called Raman-like scattering.

[8] https://www.rp-photonics.com/rayleigh_scattering.html
[9] https://www.rp-photonics.com/raman_scattering.html

Brillouin scattering in optical fibers primarily occurs in the backward direction, but there can also be some weaker forward Brillouin scattering due to the acoustic waveguide's influence [13].

Brillouin scattering is used in Brillouin spectroscopy. Thanks to its uniform distribution along the fiber, it can also be used in DAS systems, although it is much weaker than Rayleigh scattering [10]. Brillouin *line width* $\Gamma = 1/\tau$ measures material viscosity. In fiber-optic sensing, the Brillouin scattering measures temperature even in distributed manner[10] [13].

[10]For distributed sensing, please see Chapter 2

2 Distributed Sensing

Distributed sensing (in general) is a technology using optical fiber as an array of sensors. It was started in the field of optical reflectometry. There are thousands of virtual sensors along the optical fiber. These sensors are not real devices but rather a clever way of measuring differences in the light signal properties, such as changes in phase. It can measure tension, compression, temperature, vibrations, and other strain impacting the fiber and, consequently, the light passing through it [10].

Distributed sensing can be based on a single scattering effect (Rayleigh, Raman, or Brillouin). These effects can be combined to improve measurement properties, like accuracy and spatial resolution [18].

We distinguish multiple distributed sensing systems depending on what is measured:

- *Distributed Acoustic Sensing* (DAS) - recording sound and vibrations.
- *Distributed Vibration Sensing* (DVS) - vibration detection.
- *Distributed Strain Sensing* (DSS) - twisting, pulling, bending.
- *Distributed Temperature Sensing* (DTS) - temperature measurements.

The work will focus on DAS rather than the other Distributed sensing methods because the DAS is used for the measurements. That said, other Distributed sensing methods are quite similar.

The scattering effects create dispersion effects in the light, which leads to distortion of the light pulse, making it broader. The superposition of neighboring light pulses also limits the transmission frequency. The maximum frequency possible on a transmission line is approximated by Formula 2.1. It takes into account the fact that the light has to travel from the light source to the end of the fiber and back to the starting point at the light source.

$$f_i = \frac{c}{n(2L + 2P + 3D)} \tag{2.1}$$

2.1 Distributed sensing based on Brillouin scattering

Brillouin scattering has been used for distributed sensing since the 1980s when distributed temperature measurement was introduced. Scattering centers for Brillouin scattering are the molecules of the fiber. As they are evenly distributed along the entire length of the optical fiber, it is a perfect candidate for distributed sensing. The Brillouin scattering has a backward and a frontward scattering effect, and both can be used for distributed sensing [13].

Brillouin scattering is mostly used for distributed measurements of temperature in DTS as the scattering effect depends on the temperature vibrations of atoms and molecules in the optic fiber [14].

There are also acoustic waves present in the fiber when vibrations or strain affect the fiber. When an acoustic wave interacts with an optical wave, it creates a scattering effect, which produces a new optical wave with a shifted frequency. This wave is referred to as the *Stokes wave* if it has a lower frequency than the original pump wave (the source wave) and as the *anti-Stokes wave* if it has a higher frequency. It is possible to stimulate this phenomenon, resulting in an exponential amplification of the optical Stokes wave. This process is called *Stimulated Brillouin Scattering* (SBS).

Brillouin Optical Time Domain Reflectometry (BOTDR) is a technique based on *spontaneous Brillouin scattering*. Its biggest advantage is that it only needs access to one end of the fiber. Nonetheless, the spatial resolution of this approach is restricted to approximately 1 m, determined by the phonon lifetime in optical fibers (10 ns) [13].

2.2 Distributed sensing based on Rayleigh scattering

Rayleigh scattering, as discussed in Section 1.3.2, is a great candidate for distributed sensing as it can extract three main properties of light - intensity, phase, and polarization. The biggest advantage of Rayleigh scattering is that it is almost completely free from external physical fields - electromagnetic, microwave, and others. The signal is also quite strong power-wise compared to Brillouin and Raman scattering; see Figure 1.2 for a comparison of the two. In Rayleigh-based distributed sensors, scattering is used to track and reveal propagation effects such as attenuation and gain, phase interference, and polarization variation.

Rayleigh scattering originates from the light reflecting back from the molecules and atoms in the fiber. This differs from the scattering from the crystalline lattice (Raman scattering) and atom vibrations (Brillouin scattering). Rayleigh scattering can sense more than strain and temperature - it senses chemical concentration, pressure, vibrations, ionizing radiation, and relative humidity. Polarization enables sensors to detect changes in the magnetic field, twist, and geometrical layout. Detection of phase changes is crucial for sensing based on Rayleigh scattering.

The back-scattered light can be characterized as the coherent superposition of the light generated from randomly distributed scattering centers in the fiber. The scattering centers create radiation in all directions, but some light travels back to the source, where it can be detected and analyzed. According to Rayleigh's theory, the backscattered light is in phase with the incident light and has the same polarization.

The intensity of the light reflected by the scattering center has random quality as the density of the material changes throughout the fiber. Neglecting the polarization effects and dispersion, the complex envelope *b(t)* of the backscattered light can be described as follows in Equation 2.2. β is the propagation constant of the optic fiber, *a(z)* describes the attenuation accumulated up to z, c_n and z_n are random amplitude and position of the nth scattering center. The τ_n is a group delay introduced by the propagation up to z_n, and factor 2 accounts for the roundtrip propagation; *a(t)* is a wave function of the signal from the light source, the statistics of c_n and z_n are not important in this context [14].

$$b(t) = \sum c_n e^{-2[\alpha(z_n)+j\beta z_n]} a(t - 2\tau_n) \tag{2.2}$$

$$\tau_n = z_{nd}\beta/d\omega \tag{2.3}$$

The measurement involves retrieving the attenuation $a(z)$ by measuring $b(t)$. This creates two modes of measurement either it means to first probe the fiber by continuous wave signals at different frequencies to measure the frequency response and then compare it to the values during the real measurement (frequency domain), or to measure the response of the fiber by analyzing the roundtrip propagation through the fiber (time domain) [14].

2.3 Optical reflectometry

Optical reflectometry is used for measuring optical cable properties; it can detect defects, joints, breaks, or other damage and their location on the wire. It is the basis for *Optical Time Domain Reflectometry* (OTDR) or *Optical Frequency Domain Reflectometry* (OFDR) and other optical sensors such as DAS. A pulse of light is sent from the source, such as a light-emitting diode (LED) or a laser diode (LD). The light travels from the source through the optical fiber in pulses and is reflected from the other side of the wire through connections, breaks, damage, or imperfections in the material. All of these create some back-scattering toward the light source [25].

2.3.1 Optical Time Domain Reflectometry

Phase-Sensitive Optical Time Domain Reflectometry (Φ-OTDR) is the most widely used method. When a strain is applied to the fiber, it causes phase-shift changes in the light signal. Provides high sensitivity, resolution, and sampling rates in the frequency range between hertz and kilohertz. Φ-OTDR has limitations in measuring

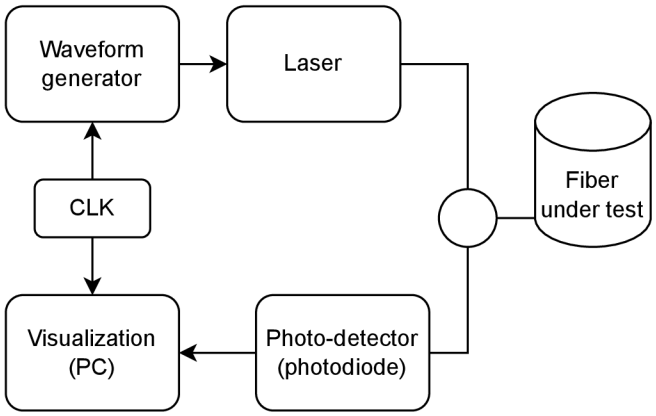

Fig. 2.1: Example of basic OTDR reflectometer.

slowly changing effects and noise components [15]. The optical time domain reflectometer was proposed back in 1976 by Barnoski and Jensen. The principle is to send a light pulse through the fiber to measure the impulse response; see Figure 2.1.

2.3.2 OFDR

OFDR analyzes interference of signal between the initial signal and the back-scattered signal but focuses on the frequency scan. A source of light has to be a laser diode that can be precisely tuned to a certain frequency. The signal given by OFDR contains frequency information that can be processed with the Fourier transformation. The output would be the position of the reflective elements along the fiber length. There are also other methods to measure light properties such as *C-OFDR* - coherent version of OFDR using light with the frequency with linear dependency but having problems with high noise levels, and DSS - Mandelstam-Brillouin [21].

2.4 Distributed Acoustic Sensing

Implementation of DAS system is usually done by OTDR, OFDR, or analysis of other light properties such as polarization and back-scatter correlation. DAS allows the measurement on thousands of points on an optical wire without the need to cut the wire or have multiple sensors distributed along the wire. The measurement mechanism is based on optical reflectometry, the same as in DTS, a variant of OTDR. DAS relies on non-uniformities spread evenly along the fiber. As discussed in Section 1.3.2, the best suited for this application is Rayleigh scattering, which describes light scattering from particles smaller than the wavelength of the light. In this case, the particles are molecules and atoms in the fiber.

Scattering effects suitable for the use in DAS measurements are Brillouin, Raman and Rayleigh

During measurement, pulses of light are sent into the optical fiber. The fiber creates a light scattering (e.g., Rayleigh scattering) in the glass that travels back to the sensing unit (on the same side as the light source), which can be interpreted based on the arrival time as a position on the wire. Back-scattering light from the optical fiber segment is detected at the light source. If a strain or a vibration is applied to the optical fiber, it detects changes in amplitude or phase, which means that the fiber wire segment is externally affected somehow. DAS is used in a wide range of applications, from locating seismic activity, locating trains along the train tracks, as a gyroscope or an accelerometer or even as a microphone [9], [21].

DAS uses optical fiber as many sensors along its length. The fiber is capable of detecting vibrations along it can also detect acoustic properties as they are also vibrations but of sound. These sensors allow for measuring acoustic properties such as frequency, amplitude, and phase [9].

When installing cables, it is crucial to consider both cable design and installation methods. The rigidity of the fiber is a significant factor because stiffer cables can reduce sensitivity. Optimal results are achieved by utilizing a single-mode fiber with minimal protection buried in the ground. The contact with the surroundings is also essential as it can impact the signal-to-noise ratio and alter the frequency content of the recorded signal.

When using an existing fiber optic infrastructure, it is hard to know what parts of the fiber are in contact with, for example, soil or ground, when measuring seismic activity, as good contact is crucial to yield good results. When monitoring boreholes, the wire can be lowered into the hole, but the quality of measurements will vary. For this purpose, mounting the cable to the bore pipe is a better solution. Good contact with the ground or seafloor is very hard to achieve, for example, in underwater applications for measuring underwater seismic activity. When laying the fiber optic

cable, the cable can stretch over crevices and underwater valleys and dips, failing to make contact, which hinders the measurements and has to be accounted for.

There are also special optic fibers being developed for making the connection with the ground uniform. They have a better signal-to-noise ratio and good transmission of external vibrations on the fiber while maintaining mechanical protection [10].

2.4.1 Measurements

The fiber is divided into segments representing a sensor located along the fiber. These sensors are just virtual - they are not real devices. By measuring time differences between the time light was transmitted and the reflected light reached the sensor, interrogator devices can identify what portion of the fiber is affected. Each segment is called *Gauge Length* and represents a sensor. The location of a sensor is in the middle of the gauge length. Sometimes neighboring segments can overlap. The signal is formed between two points on the reflectogram corresponding to the edges of the gauge length. Setting gauge length is crucial to get the correct measurement. If the gauge length is too small, it degrades the signal-to-noise ratio. If too big, it creates signal distortion. Calibration is sometimes necessary to account for insufficient ground contact and to make measurements as precise as possible [10].

2.4.2 Advantages of DAS systems

The possibility of using existing fiber-optic infrastructure makes deploying sensing arrays very easy and cheap. Unused fiber optic wires laid for later activation, when higher bandwidth is required, are called *dark-fibers* and can also be used for distributed sensing. DAS technology can be applied to many different applications with the biggest advantages over the traditional sensors and electric devices in terms of no electromagnetic interference and adverse conditions - radioactivity, harsh chemical environments, high temperature, underwater, and others, see Section 1.1.1. It is hard to imagine the limitations of this new technology [10].

2.4.3 Disadvantages to traditional sensors

When using DAS for seismology, good contact with the ground is necessary. With insufficient contact, the DAS technology can provide unsuitable values. Traditional seismometers plot data instantaneously in comparison to DAS systems that plot data with time delay [10]. The size of fiber-optic sensors such as fiber-optic gyroscopes is still huge compared to *Microelectromechanical Systems* (MEMS) sensors used in today's microelectronics. MEMS can be soldered to PCBs and used in a wide range

of applications, but they also have weaknesses, like poorer precision and low heat resistance.

2.4.4 iDAS

The distributed acoustic sensor is a new addition to distributed optical fiber sensors used in the energy industry and can be used in many applications, for example, in detecting seismic activity. iDAS (intelligent distributed acoustic sensor) is one type of DAS sensor. One of the applications of this sensor is to record an acoustic signal. To determine the signal fidelity, a certain part of the wire is subjected to a known signal, for example, a sine wave. A measurement is made, and the result is compared with the existing recording device. The result suggests that iDAS has very good signal properties. The measured signal shows that almost no measurable crosstalk is exhibited between the two sensing channels on the wire.

The maximum sampling rate can be calculated from the speed of light that travels in glass at a speed of about 200 000 km/s, which corresponds to approximately 10 kHz for 10 km long wire [9].

- Acoustic bandwidth.
- Dynamic range - 120 dB as reported in [12].
- Spatial resolution - about 1 m to 10 m, but up to 25 cm is possible.
- Measurement range - The fiber length can be anywhere from a few hundred meters to more than 100 km.

2.5 OptaSense ODH-F

Data used in this project are obtained from *OptaSense ODH-F Distributed Acoustic Sensing Interrogator*[1](Interrogator). This device is capable of monitoring optical fiber up to 50 km long (in qualitative mode). It allows for sequential monitoring of four cables at the same time. It is used for in-well flow monitoring, pipeline integrity management, and border security.

OptaSense uses *Coherent Φ-OTDR* (C-OTDR), for further explanation see Section 2.3.1

OptaSense comes with *DxS Visualization Software* capable of analyzing and processing output data from the unit. It can show the signal spectrum in a waterfall graph and create analysis, process the signal using *Fast Fourier Transformation* (FFT), and extract data to `.wav` format. It has the limitation that it can only run in the Windows ecosystem and is proprietary software, so scientists cannot change how they work with the data or how it is displayed.

[1] `https://www.optasense.com/technology/odhf/`

2.6 Existing technology for HDF5 data visualization

This section focuses on data processing and visualization using existing software for visualizing scientific data. First is OptaSense OS6 software which is a purpose-made solution for OptaSense devices. Next is the h5web library, written in React, which creates a web page for visualizing the content of HDF5 files.

2.6.1 HDF5 file format

The data captured by the Optasense Interrogator is collected in *Hierarchical Data Format v5* (HDF5) and saved into a file with *.h5* suffix. HDF5 creates a model for managing and storing data. The HDF Group[2] maintains the format and its corresponding software package. Specifically, the HDF5 format is used to store data, but only one of the three parts makes up the full HDF5 model. Parts of the HDF5 model are:

- File format - files ending with .h5
- Data model - specifies the building blocks of the HDF5 file format
- Software - libraries, tools, APIs

HDF5 data model has a folder-like organization, where the folders are called *groups*. This model specifies the format in which the data are stored in the form of *Abstract Data Model* (ADM), which specifies the organization of the data and the types of data. Every HDF5 file has to have a root group *"/"*. Working with groups is very similar to directories on Linux systems; see Section 4.2.1. Each group can have *datasets* that contain raw data, attributes, data types, and other objects. HDF5 file can also specify links to other libraries and tools such as compression and filtering.

HDF5 dataset has connections with other HDF5 objects:

- **attributes** - named data object containing the name and the value
- **datatypes**:
  - Atomic datatypes - time, string, integer, float.
  - Composite datatypes - array, enumeration, compound, variable length.
- **data** - data itself, for example, the result of measurement.
- **dataspace** - the shape of the data.

[2] `https://www.hdfgroup.org/`

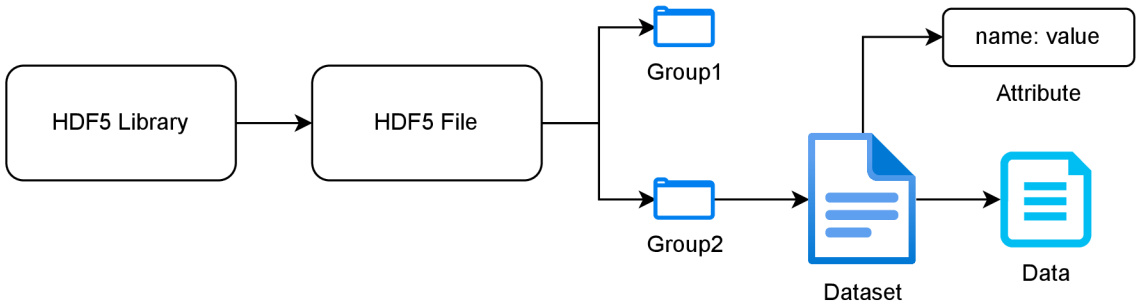

Fig. 2.2: Basic HDF5 file structure.

2.6.2 OptaSense OS6

The OptaSense company provides visualization software for their devices called OptaSense OS6[3]. It supports only Windows operating system. OS6 provides features for monitoring areas or land, for example, a compound or an industrial building. This product is tailor-made for OptaSense devices by the OptaSense company. This system has only one window for everything. The primary view is the monitored area; the background picture is the aerial view of the monitored space, as seen in the picture 2.3. The user can open the sidebar on the right side. The sidebar provides multiple different options:

- **Spectrogram** - Raw data visualization.
- **Alerts** - When an action is detected along the wire, it is logged.
- **Notifications** - Notifications about system state.
- **System status** - Overview of all OptaSense units and their state.

There is also a feature that takes raw data from interrogator units and processes them using machine learning. This way, different actions are detected and categorized into different alerts, such as walking, driving cars, etc. In addition, the user can see the activities detected and triggered in the area overview with live monitoring and a timeline at the top of the screen. To easily look at different locations or start a new view, a feature *type to search* lets the user start a search by typing into the view. For example, the user starts writing "water..." as a waterfall, and the program will look for this feature and open the waterfall visualization window. OS6 saves all detected activities, shown in the *Historic timeline* window, which shows all alerts during a specified time range. The animations look very nice, although some look choppy, mostly when showing activities on top of the waterfall view.

It proves that it is possible to create a real-time data visualization from the DAS system. It lacks one important step, which is the ability to be used not only on Windows machines and be multi-platform. And although it provides enough tools

[3] `https://www.optasense.com/technology/os/`

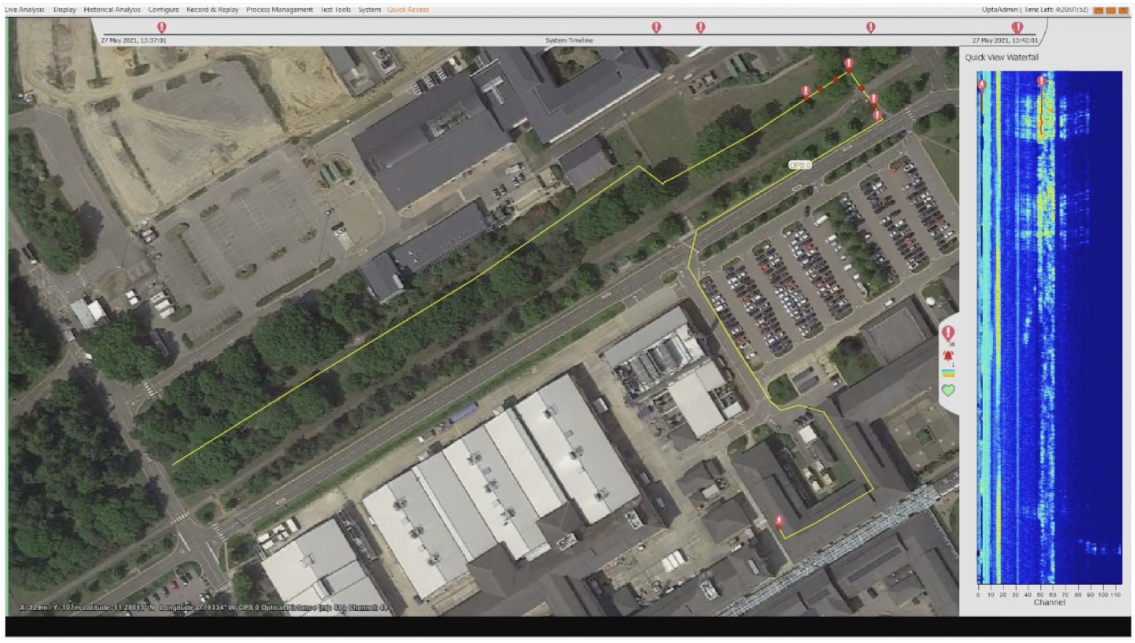

Fig. 2.3: OptaSense OS6 visualization software [27].

for data analysis, it is unsuitable for further scientific work such as custom editing the visualization or exporting data to images and further processing for machine learning and activity categorization.

2.6.3 h5web

The `h5web`[4] library is a set of components written in React[5]. H5web uses existing HDF5 libraries, such as h5wasm (reading HDF5 files in the browser) and h5grove (server for accessing HDF5 files). It displays the contents of the HDF5 file and shows different graphs according to the input from the user. From the presentation of the library by its developer, it is safe to say that although it provides the necessary equipment for opening HDF5 files elegantly and provides advanced graphing techniques, it lacks the ability to receive the data and display them as they were coming from the DAS system. For this purpose, the library would need to add support by creating a new React component capable of such behavior.

[4] `https://h5web.panosc.eu/`

[5] React is a JavaScript library used to create interactive user interface `https://reactjs.org/`

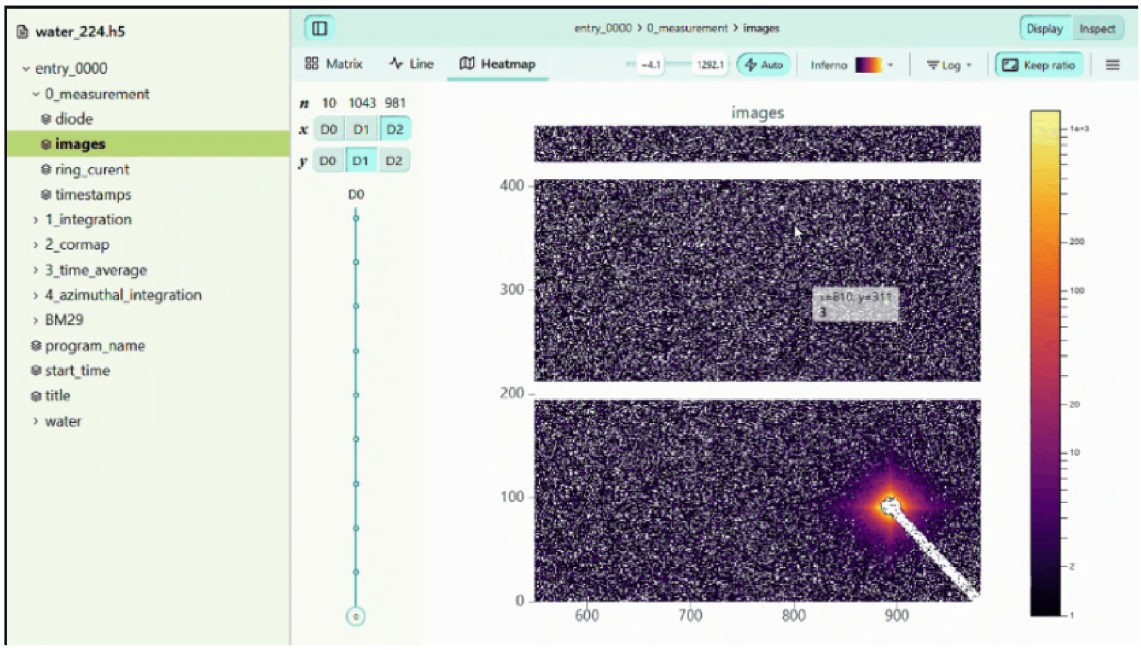

Fig. 2.4: h5web application with an example data visualization.

3 Software design

There are many ways to implement data visualization. Still, choosing the right solution, programming language, or framework is hard, so this chapter first provides information on what this application should do. Second, it studies the existing OptaSense software and other solutions accessible from the internet. Lastly, it explains the software design decisions for implementing this data visualization.

3.1 Software design analysis

Unified Modelling Language (UML) is a design language made to make it easier for developers and system designers to communicate with each other. UML uses different visualization types, graphs, and charts to ease the understanding of complex systems. This way, software design is more understandable and standardized. And when developers, software designers, and management speak the same language - UML language - making the development more efficient. It makes management easier as they do not have to understand the implementation but rather understand the meaning of each part of the system. This way, they can make better time assumptions and time planning more predictable. Software designers can specify many diagrams to explain their ideas to developers to showcase the functionality. Also, the diagrams are far more understandable than pure code or pseudo-code implementation.

There are many different UML diagrams. The basic division is to *structural* and *behavioral* diagrams. For this work, we will need three types of diagrams. From the structural diagrams, we will use *Use Case Diagram* and *Class Diagram*. From the behavioral diagrams, we use *Sequence diagram* [32].

3.1.1 Use cases

A *Use Case Diagram* is an UML diagram to form the system or software requirements for a new software program. It is part of *behavioral diagrams* in the UML language. Use case diagrams show the user's point of view of the system, the way they will use the system, with its main functions and features. It consists of the following:

- `Actors` - interacting with software functions; a noun names them; the actor triggers the use cases.
- `Use case` - the features and functions of the system; are specified by a phrase describing the action.

Each of the use cases has to have an actor linked to itself. There are also special relationships that extend and include that are not in the interest of this work but are important in general for Use Case diagrams [32].

The task is to fulfill the usage requirements, as seen in the use case Figure 3.1. Users need to see and view what is happening in their perimeter on their screen. For this purpose, the best data visualization is a waterfall graph, similar to a spectrogram, displayed as the main element. It should have an editable color map to adjust the sensitivity. The waterfall view should be an animated waterfall graph and ideally display real-time data on the screen, similar to the OS6 system; see Section 2.3. In addition, the user should perform selection and zoom on the waterfall graph. The user should be able to edit properties of the graph, like changing the data range and choosing the channels he or she wants to see. The user should be able to export the data to a WAV file by clicking a button and then viewing and playing the audio file. There should also be a waveform display available to show the playing data.

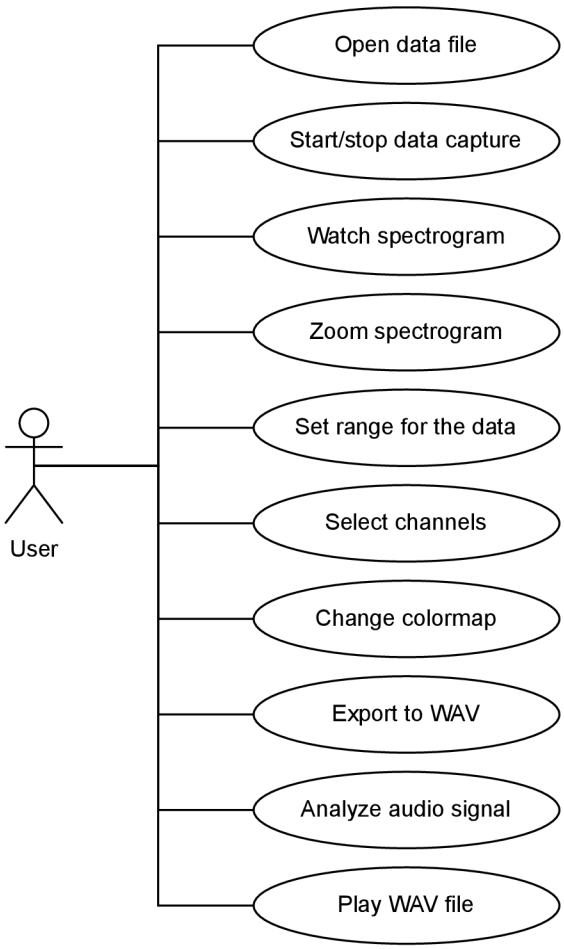

Fig. 3.1: Application use cases.

3.1.2 Application requirements

From the use cases in Section 3.1.1, it is understandable that the application should have certain features. Apart from the given use cases, there are other important requirements:

- Multiplatform - the application should run on any device and still support all features
- Data processing - subsampling data to save data throughput
- Plot editing and animation - changing plot properties
- Reading offline data - possibility to read local files or upload files into the application

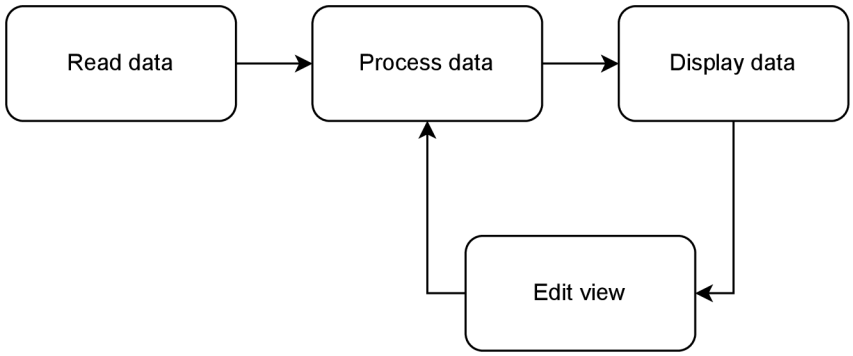

Fig. 3.2: Data flow in the application - reading the data, then processing it and displaying it (optionally) edit the view.

It is necessary to choose the correct programming tools to satisfy all features. The right way to find the right solution in programming is to *divide and conquer*. This means finding all the pieces that will make the application. First, there must be an idea of what will happen with the data. Firstly, it has to be read, processed, and then displayed. The optional step is to edit the data or change the view; this data flow can be seen in Figure 3.2.

From the data flow application, an overview can be made for a web application, as seen in Figure 3.3. The overview illustrates what parts the application will have. The back-end or server application will be responsible for reading HDF5 files and processing data. It will provide some interface or API for the client application to fetch[1] the data. On the client side, the client has to be able to create visualizations of the data and provide a user interface to change application properties.

[1] `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`

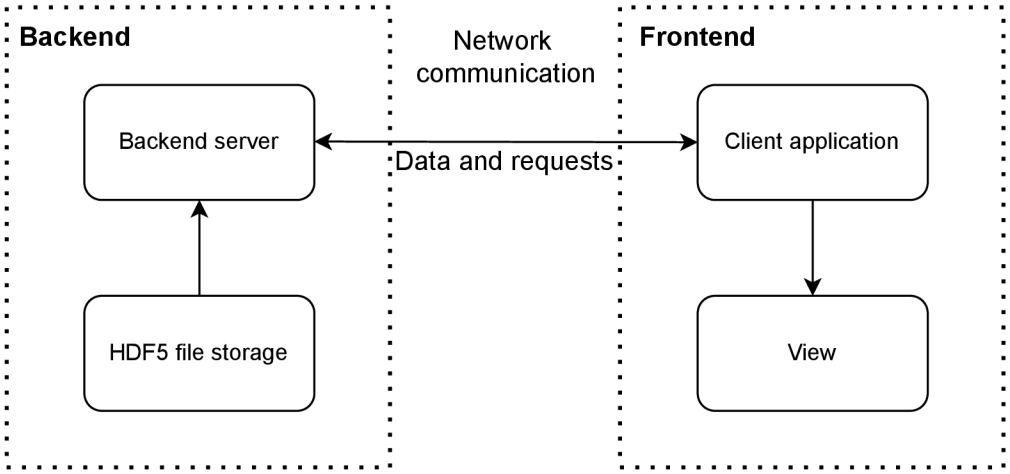

Fig. 3.3: Application overview. The server reads the data from the data storage and sends it to the client application, where it is shown to the user.

3.2 Back-end

The purpose of the back-end part of the application is to read, process, and send the data to the client part of the application, as shown in Figure 3.5. Reading the HDF5 data is done as explained in Section 4.2.1.

3.2.1 Existing REST based servers for HDF5 data access

Representational State Transfer (REST) is a software architecture. More specific is a RESTful API (*Application Programming Interface* (API)), which transfers data between systems on the internet. One of the most common usages is in web services. REST uses stateless messages to get data from some resource. The data can be delivered in many different formats. Very popular are *JavaScript Object Notation* (JSON), *HyperText Markup Language* (HTML). The communication is based on *Hypertext Transfer Protocol* (HTTP) protocol activities:

- `GET` - request a record from the other side,
- `POST` - request to create a record,
- `PUT` - request to update a record,
- `DELETE` - request to delete a record [33]

There is a wide range of REST server implementations; the HDF Group provides documentation for its RESTful API[2]. They have prepared a few RESTful server implementations for their data format. Many more implementations of REST servers, such as the Python Simple HTTP server. Here is a list of some possible implementations [23]:

[2] `https://support.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf`

- *h5serv* - REST-based service to access HDF5 data written in Python (HDF5 Group).
- *HSDS* (Highly Scalable Data Service)[3] - Python implementation of the REST-based service to access HDF5 data stores. Data can be stored in the POSIX file system or object-based storage such as AWS S3. It can run in a Docker as a single machine or on a Kubernetes cluster.
- *hdf-rest-api*[4] - is HDF5 REST API that provides CRUD support (create, read, update, and delete) for all HDF5 objects.
- *h5grove* - Back-end service written in Python providing access to HDF5 file content.
- *http.server* - Python implementation of a simple HTTP server. However, this is better used only for testing when accessing local files.

3.2.2 WebSockets

WebSockets (or WebSocket API) enable two-way communication over *Transmission Control Protocol* (TCP). IEFT standardized it in RFC6455[5]. All modern browsers support the WebSocket protocol.

The communication starts with an HTTP-compatible handshake, so only one socket can communicate with the server. There are also other header types available for different uses. The server responds with an HTTP Upgrade, the connection is established, and bidirectional communication can begin. Communication closes when either side closes the connection and starts completing the handshake. The other side responds with a *Close frame* message, closing the connection.

The protocol is frame-based, as is HTTP protocol, but simultaneously, it tries to be frame-based as little as possible. Just enough to make sure that it can use the HTTP interface for communication; otherwise, it tries to be as minimalist. The authors of the WebSocket protocol wanted it to be low-level, and as close to TCP as possible [28]. WebSockets have an implementation in the Python programming language called `websockets`[6].

[3] `https://github.com/HDFGroup/hsds`
[4] https://github.com/HDFGroup/hdf-rest-api
[5] https://www.rfc-editor.org/rfc/rfc6455
[6] https://pypi.org/project/websockets/

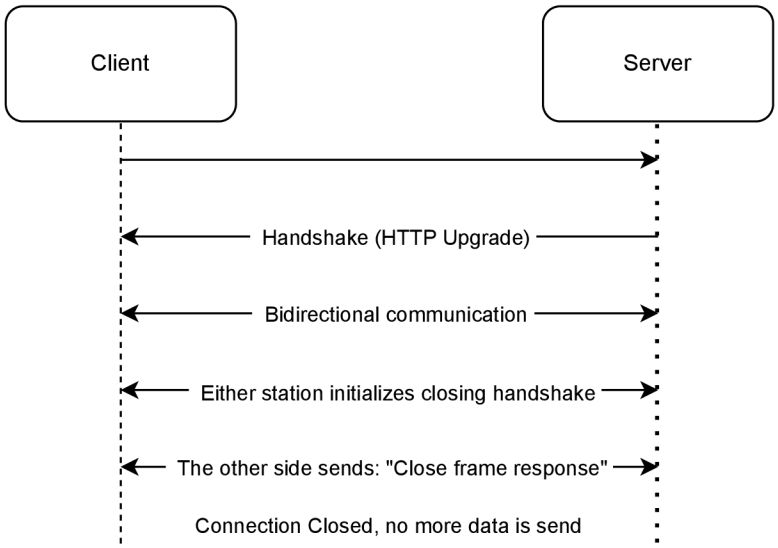

Fig. 3.4: WebSocket handshake, communication, and connection close diagram.

3.2.3 Python asyncio library

Our application must receive and send messages simultaneously when dealing with client-server communication. There are multiple ways of achieving this behavior. Either use multithreading[7] or multiprocessing[8] as all of these solutions solve the same problem - doing multiple things at the same time. There is also an alternative way to receive and send messages asynchronously. Asynchronous programming in Python uses an asynchronous library *asyncio*. All of them are a viable solution to this problem.

It might be a better solution to use multithreading in the future. Still, it is unnecessary, as it makes the code more complicated and creates an overhead when sharing data between threads. The asynchronous approach is simple, effective, and sufficient for our usage, so we have decided to use it.

There is not that much code involved, and it is relatively easy to program in, although asynchronous programming is quite tricky to understand. In our testing, this solution was working well; see Section 4.1 and Listings 4.1.

The *asyncio* library provides a set of high-level APIs to run Python coroutines, distribute tasks using queues, perform I/O operations, and synchronize tasks in concurrent mode. It has a *running loop*, to which the developer can register many different *coroutines*, functionalities, and tasks concurrently. It can manage these by using queues, tasks, and events. The Python implementation of WebSockets also uses the asyncio library. The *event loop* is usually run at the beginning of the

[7]https://docs.python.org/3/library/threading.html
[8]https://docs.python.org/3/library/multiprocessing.html

application. There can be more the one event loop.

The event loop can register a `Task` classes to run multiple tasks. Although it may seem like the tasks are running in parallel, the reality is that the asyncio has a *scheduler* - a logic that decides which task should be run at what time. This way, the tasks seem to run in parallel, but in reality, they run in a single process or thread, and the tasks are just switching from one to another. This enables one thread to wait for a message from a client asynchronously and, at the same time, send messages to the client side.

Asyncio also implements `Event` class, which can be used for awaiting certain messages or conditions. By calling an `await event.wait()` the coroutine can wait for another coroutine to execute `event.set()`, which allows the waiting coroutine to continue execution. To stop execution `await event.clear()` needs to be called, and on the next `await event.wait()` the execution will stop.

3.3 Frontend

This section provides an overview of visualization techniques and properties for visualizing data from the DAS interrogator described in the previous chapter. It introduces different technologies that can be used for this data visualization. Firstly it describes the JavaScript framework Svelte which is the heart of the application. We provide an assessment of real-time capabilities.

3.3.1 Svelte

Svelte is a component framework written in JavaScript, using a new approach to building web applications. Instead of looking for differences in virtual DOMs as React does, which is done in the browser and consumes quite a lot of resources, Svelte does everything at the compilation stage. The compilation output is a JavaScript file `bundle.js`, which contains all the necessary code to run the web application or, better said, it manipulates the DOM directly. The result is a fast and reactive web page, it also saves resources, and the code can be run on small devices like handheld devices. Web development is also very enjoyable because compilation does not take long, and the changes can be visible immediately. The structure of a Svelte component consists of three parts - JavaScript code tag `<script>` a style tag `<style>` and other HTML elements.

Svelte does not provide more advanced features like page routing. For this purpose, the Svelte team created SvelteKit, which is a framework for building web applications and allows page routing. Routing is folder-based - the developer creates a folder and file structure.

```
src/routes/about/+page.svelte <=> /about
src/routes/about.svelte <=> /about
```

Svelte has been changing and has become a Vite [9] plugin. Vite provides fast development experience by running a development server, in the case of Svelte - Hot Module Replacement (HMR)[10]. This way, every change made during development can be immediately seen in the browser without reloading the page, which makes development much faster and more enjoyable. It is necessary to say that Svelte is still in development, and although it is now at version 3, it may change in the future.

When fetching the data from the server, it is good practice to move this functionality to a Svelte Store. From a programming point of view, a store is an object with a `subscribe()`, `set()` function. An example of a WebSocket implementation in Svelte can be the Svelte component library *svelte-websocket-store*[11].

3.3.2 Real-time capabilities

The data bandwidth (the amount of data necessary to be sent from the server side to the client side) of the application is the biggest factor. The OptaSense Interrogator can produce quite a lot of data, but if it is saved in an HDF5 file, as it is compressed, it is quite small. As discussed in Section 4.2.1, a 10 s file produces around 52 MB of data. When this data is transformed into text form, it has only 420 MB, and when transformed into JSON, it has 946 MB as the data are read at 10 kSPS[12]. Data will be displayed on display with standard resolution and cannot display 10 000 SPS on a small part of the display. Data processing is necessary for this purpose.

3.3.3 Data processing

Sending data in the form of REST requests and responses is possible, but it is really useful only when sending a small HDF5 file as a whole, not as a stream of data. The h5grove[13] back-end provides the ability to read sections of the data. The raw data file is unfortunately not suitable for display as it needs to be processed before the visualization. As the files are quite large, it is far better to run some data processing algorithm, see Section 4.1.2. So the data need to be smaller, in a suitable format, and ideally subsampled as it is not viable to display thousands of samples per second on a small visualization screen.

[9] `https://vitejs.dev`
[10] `https://vitejs.dev/guide/features.html#hot-module-replacement`
[11] `https://github.com/arlac77/svelte-websocket-store`
[12] *Samples Per Second* (SPS)
[13] https://pypi.org/project/h5grove/

Data processing can be done on the server or in the browser. As the browser will be busy redrawing waterfall visualization, it is better to process data on the server side. Server-side preprocessing will also save bandwidth as the data will be significantly filtered and in easier to send type. There is no need to send *float64* values as a text using JSON and WebSockets or with REST API.

3.3.4 Software design frontend for DAS data visualization

As we discussed in Section 2.6.2, it is possible to create such software to display the data in real-time. The proprietary application from Optasense is a native Windows application. The requirement for this application was to be able to run on multiple platforms. The chosen platform is the Python back-end for opening files, processing the data, and sending the data to the client application using Svelte. The back-end will process the data as explained in Section 4.1.2. The waterfall graph will be an HTML Canvas element that displays the data in real-time, redrawing itself as the data arrive at the browser. For ease of displaying the data in Canvas, D3.js will be used. D3 will, for example, apply a color map to the correct scale according to the data. The user interface written in Svelte will also have inputs to change the properties of the visualization so that the user can select specific channels from the data, choose subsampling effect properties, cutting the frequency range. The ability to export the data to a WAV file will also be implemented the same way as done in Section 4.2.

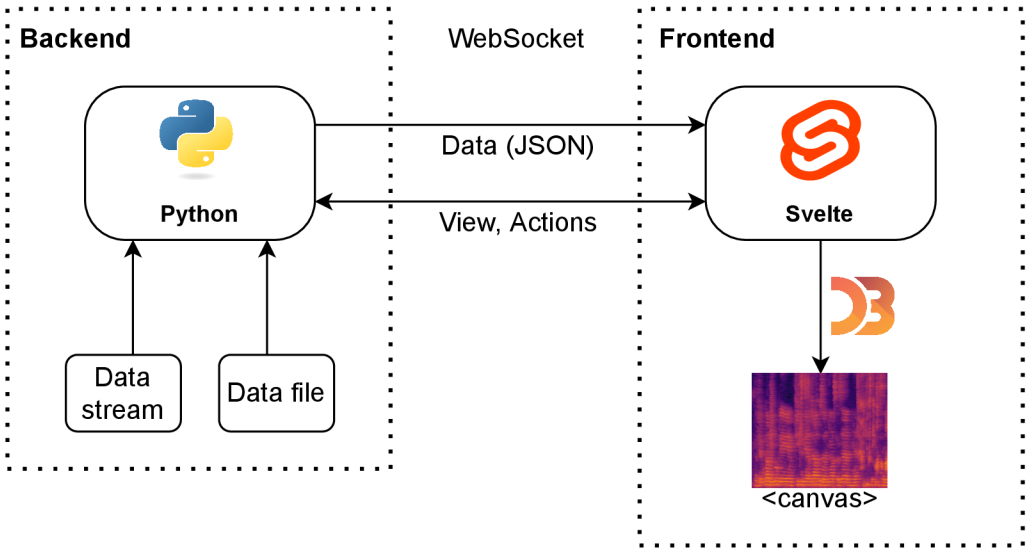

Fig. 3.5: Application overview.

3.4 HTML chart rendering

For in-browser rendering, two main options exist using HTML canvas rendering or using *Scalable Vector Graphics* (SVG) elements. In this section, we will discuss both their advantages over each other and their drawbacks. In general, Canvas rendering performs better than SVG rendering. This is because SVG is based on a *Document Object Model* (DOM) structure[14]. It is more suitable for large datasets and graphics-heavy games and animations[15].

3.4.1 HTML SVG graphics

SVG is an XML-based markup language developed by W3C[16]. It describes 2D vector graphics in XML text files. SVG supports three types of graphic objects: vector shapes, bitmap images, and text. The main advantage compared to bitmap graphics is that all the elements can be rendered at any size without losing quality.

The description of graphics elements is the same way as the web page description written in HTML to the final web page. There are tags for different geometrical objects, for example, rectangles, ellipses, lines, and animations. Everything is defined in an XML text file which can be edited, searched, or compressed[17].

Data visualization using SVG graphics results in smooth and sharp visualizations. It also enables interactivity with each displayed object - selecting, hovering, or zooming. Libraries for SVG manipulation manipulate text based on the data given. *Data-Driven Documents, a JavaScript framework* (D3.js) is a JavaScript library using HTML, SVG, and CSS to visualize data. It can bind data to the HTML DOM and then apply data-driven transformations. D3.js allows developers to create custom visualizations as it is not a charting library but a set of data visualization tools. It is also the base for other charting libraries that build on the D3.js'f framework. They include Plotly's JavaScript implementation, C3.js[18] or Britecharts[19].

3.4.2 Canvas graphics

JavaScript makes drawing to the HTML `<canvas>` element possible with the use of *Canvas API* or *WebGL API*. It can render data visualizations, game graphics, real-time video processing, and animation. WebGL can draw 2D and 3D graphics

[14]text-based structure defining objects displayed on the web page. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[15]https://www.chartjs.org/docs/latest/#canvas-rendering

[16]World Wide Web Consortium www.w3.org

[17]https://developer.mozilla.org/en-US/docs/Web/SVG

[18]https://c3js.org/

[19]https://britecharts.github.io/britecharts/

to the canvas element, but we will not be covering WebGL further. Canvas API focuses on 2D graphics. Libraries using Canvas API that make rendering to canvas easier differ on the use case - EaselJS (web game development), Konva.js (desktop and mobile applications), Chart.js, and many more. Canvas rendering depends on the resolution of the screen.

The `<canvas>` tag is used for drawing graphics. You can imagine the canvas as a rectangular area with the starting point at the top left corner with coordinates (0,0) and defined width and height. By default, this area is transparent. When we want to draw to it, we need to call functions from the Canvas API. They define basic shapes and primitives that can be drawn on the canvas. Rectangles and paths are the only primitive shapes that can be drawn to `<canvas>` element, and more complex shapes can be drawn by combining these primitives[20]. Basic example of drawing to canvas can be seen in the Listing 3.1. Usually, the functions define certain shapes filled with color and added to the canvas. Then "clearing" functions remove what is displayed or create transparent areas. Lastly, there are "stroke" functions that create lines.

Listing 3.1: D3js x-axis implementation.

```
function draw() {
    //First, we need a~reference to canvas element
    const canvas = document.getElementById("canvas");

    //Next, we need context to draw to
    const ctx = canvas.getContext("2d");

    /**** Drawing  ****/

    // drawing a~rectangle to canvas
    ctx.fillRect(10, 10, 100, 100);
}
```

The biggest drawback of displaying data with `<canvas>` is its lack of interactivity with displayed elements and poor text rendering capabilities[21]. The interactions are achievable but usually require "hacky" solutions like hidden canvas or invisible layers. Having more than 1000 objects rendered on screen that can be selected can also cause performance issues.

Drawing to canvas and animation in canvas

Creating animations on canvas elements is done in two ways. Calling function `setInterval()` in which a callback function is given with a time delay value in

[20] https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes
[21] https://www.w3schools.com/html/html5_svg.asp

milliseconds. This was a preferred way to create animations until the implementation of the `requestAnimationFrame()` method. It also takes a callback but without a time value. This is because it just tells the browser to perform an animation, so the developer doe not need to worry about managing the number of frames per second. It can perform a redraw up to the frame rate of the screen. So if the user has a 60 Hz screen, it redraws every 16.6 ms and respectively, if 120 Hz screen, it redraws every 8.3 ms. For updating at a lower FPS than the screen refresh rate, the callback is passed with a time stamp value of the request. This value can be compared with the time of the previous update, and if the delta is smaller than desired, it will not update the canvas. If the delta is bigger, it will allow the redraw.

3.4.3 Heatmap visualization

A *heatmap* is a graph depicting data values in cells in a two dimension grid. Usually, the cell shape is made of simple rectangles or squares, but any shape is possible. For example, population data can be visualized on a map or a globe. The color of each cell is a representation of the value. Choosing the right color palette; otherwise, the data can be hard to understand. We discuss color palettes in Section 3.4.4. The colors used in the heatmap indicate the relative values of the data. Depending on the color palette chosen, the result can be a visualization with darker colors representing higher values and lighter colors representing lower values. Showing a color scale legend near the graph is good practice for more straightforward data interpretation.

Heatmaps are used in different scenarios as they can make data more interesting or easily understandable. They can show the relationship between two variables on two axes or display data the same way as in tables, but thanks to colors making the table more understandable. They can help to find patterns in the data or locate interest points on maps[22].

Heatmaps are implemented in different programming languages and frameworks. Some interesting heatmap implementations are D3js [23], Matlab[24], Plotly[25].

A heatmap visualization using the Svelte framework and canvas element can be done in a few different ways. As the Svelte store updates, the framework also updates everything that this store is used in or connected to. The easiest way is to create a reactive statement `$: command`; see an example in Listing 3.2. When the data store updates its value every time the client sends a new message with the data, the reactive command executes. This way, we do not need any algorithms to

[22]https://chartio.com/learn/charts/heatmap-complete-guide/
[23]https://d3-graph-gallery.com/heatmap
[24]https://www.mathworks.com/help/matlab/ref/heatmap.html
[25]https://plotly.com/python/heatmaps/

update the data. It is as simple as data arrives and redraws the canvas, as shown in Listing 3.2. Although this solution is simple, it lacks the possibility to control the speed in any way.

Listing 3.2: Svelte reactive statement for redrawing canvas element.

```
$: {
    $data_store;
    draw()
}
```

The second way of drawing to a canvas is to call `requestAnimationFrame()` every time there is new data and then close the animation. This is, however, too much overhead, so we abandoned this idea. A better approach would be to have the animation loop started by calling `requestAnimationFrame()`, and each time new data arrives, the chart redraws. The redraw function needs logic placed at the beginning of it to set conditions for when to redraw the canvas according to speed.

Another solution is to redraw the canvas at a set interval a few times per second. We use a `setInterval()` function to call an `increment()` function that would be triggered by a set amount of time determined by a speed value; see Listing 4.4. It needs to be noted that the use of `setInterval()` function is discouraged since the arrival of `requestAnimationFrame()` and should not be preferred.

Listing 3.3: Svelte reactive statement for redrawing canvas element using `setInterval()`. The reactive statement updates on every `$last_row_number` store update.

```
$: {
    $last_row_number;
    draw()
}
```

For ease of programming and control over the speed of the animation, we have chosen the solution using the `setInterval()` function.

3.4.4 Colormaps for a heatmap chart

Choosing a suitable colormap is crucial when displaying data to users. The purpose of a colormap is to understandably represent the data so that it is easy for users to understand what is shown in front of them. A colormap with equal steps between colors and steps in data is best perceived - called *perceptually uniform colormap*. It was found that perception of color change is best understood by the human brain rather than changes in hue.

There are four basic colormap classes based on their function:

- Sequential - is used for ordering. Incremental changes in lightness and saturation of the single color, often with a single hue, for example, *vidris*, *plasma*, *inferno*, *binary*.
- Diverging - is used when values deviate around a median value. It uses changes in lightness and saturation of two different colors that meet in the middle, for example, *spectral*, *PiYG*, *BrBG*, *seismic*.
- Cyclic - should be used for values that wrap around at endpoints. It uses two different colors that meet in the middle, beginning, and end, for example, *twilight*, *hsv*.
- Qualitative - used for displaying information without order or relationships. Miscellanneous colours, for example *accent*, *paired*, *pastel1*.

This section was based on [29]. The data from the DAS system is sequential, so the best type for this application is one of the sequential colormaps. D3.js has a plugin library for generating colormaps *d3-scale-chromatic*[26].

Fig. 3.6: Sequential colormap [29].

Fig. 3.7: Examples of cyclic colormaps [29].

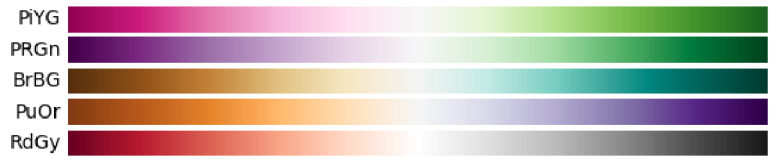

Fig. 3.8: Examples of diverging colormaps [29].

[26]https://github.com/d3/d3-scale-chromatic

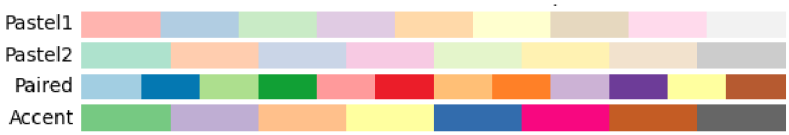

Fig. 3.9: Examples of qualitative colormaps [29].

3.5 Prototype

To pitch ideas and show the design, a prototype was made. This prototype was not a final working application. The prototype was built with the Svelte framework for its high performance and fast development; see Section 3.3.1. The prototype used Flowbite components[27] as they make it easy to create stylized web pages. The web page is divided into Svelte components. The main component is the waterfall graph on the left side of the screen and the control panel on the right side. Layout was done with svelte-layouts[28] package. Moving forward, the *svelte-layouts* package will be dropped as it is clunky in this version. Although it provides basic usage, customization is challenging because of the lack of documentation.

Uploading files into the browser using REST API was also tested. Python's `http.server` was used as the back-end. When fetching files into a browser from local storage using HTTP, it is necessary to allow Cross-Origin Resource Sharing (CORS) because browsers restrict this feature for security reasons. Some resources like CSS, Web Fonts, and WebGL textures have enabled CORS. For sending HDF5 files to the browser, a special HTTP header has to be added on the server side. Without this feature, the browser would throw an error into the JavaScript console. CORS is not used for the later stages and implementation because we are not uploading whole HDF5 files. It is much more efficient to process the files in the back-end (server) side of the application.

[27] `https://flowbite-svelte.com/`
[28] `https://www.npmjs.com/package/svelte-layouts`

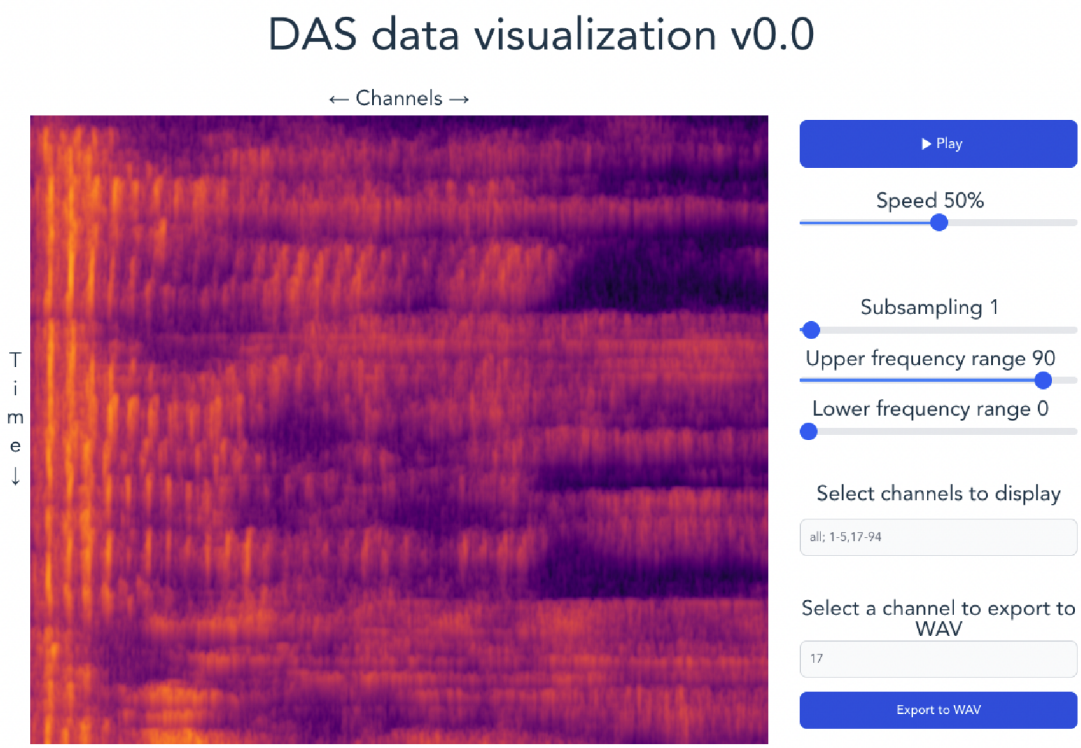

Fig. 3.10: Prototype of DAS visualization application.

4 Implementation of DAS visualization application

This chapter provides an implementation overview of the DAS visualization application. Based on the previous chapters, we put everything together and created an application for data processing and visualization. The whole application consists of two main parts a Python back-end and a JavaScript front-end.

4.1 Python back-end application implementation

The back-end is written in the programming language Python3.10[1] was chosen. Python is a great language for scientific use, data visualization, and graph plotting, which is the goal of this work. The most significant advantage comes from the availability of scientific libraries.

To run the application, creating a virtual environment and installing all the dependencies in that environment is advisable. Installation steps can be found in Attachments A. The application consists of multiple files, as seen in the Directory tree 4.1.

Python back-end file structure:

```
optasense_visualizer
├── app.py ..... Python script to run the application
└── requirements.txt ..... Dependencies .2 src/ ..... Folder with source files
    ├── file_reader.py ..... Opens and reads HDF5 files
    ├── message_classes.py ..... Data classes for incoming messages
    ├── optasense_server.py ..... Back-end part of application
    ├── range_parser.py ..... Functions for parsing channels
    ├── spectral_analysis.py ..... Functions for processing data
    └── streaming.py ..... Managing data streaming
```

4.1.1 Client-server communication from the server side

The application is started by calling `python3 app.py --port 8001`. Everything runs inside with the use of *asyncio* library. The `app.py` file parses arguments with the use of *argparse* library. The only argument is the optional *port* argument. The default value is 8001, which is later used by *websocket*.

The application has a `Server` class object created that encapsulates the behavior of the backend side of the application. It has a WebSocket instance for sending

[1] `https://www.python.org/`

messages when needed. It also has an instance of `Stream` class created to send loaded data to the client. It also saves the ”state“, which is the latest message received from the client. It also saves the name of the opened file and the dataset name handed to `DASHDF5FileReader` class.

Websocket uses *async for* waiting for messages from the client side. Incoming messages are in JSON format. Every message has a *type* which determines the incoming message. Messages are parsed using `MessageParser` class, which implements a *factory object. Factory* object is one of the most used object-oriented design patterns. It creates objects without revealing the logic to the client. The only implemented method is `parse()`, which reads the type and then creates one of the message classes and returns it. Each message corresponds to its message class. For ease of programming, *dataclasses* module was used[2]. It provides functions and a decorator for the automatic generation of special methods (`__init__()` or `__repr__()`). There are five message dataclasses `FindFiles`, `OpenFile`, `Streaming`, `Properties` and `ChannelSelection`, see Figure 4.2. When the message parsing fails, it raises an `UnknownMessageException`. The exception is thrown away as we do not want to terminate execution whenever an incorrect message is received.

Message classes, the factory object `MessageFactory` and an *Exception* class `UnknownMessageException` are located in the *message_classes.py* file.

An example of an incoming message from the client received by the server:

```
{
    'type': 'path',
    'path': '/Users/user/Documents/',
    'suffix': '.h5'
}
```

Next, the MessageFactory checks the message `type` - *"path"* corresponds to `FindFiles` class and returns it. The message class is then saved as a server state. Each message class has its own *if* statement with its behavior. Not to be confused, the behavior is defined outside of the class. It is not implemented in the data class itself.

`FindFiles` is the first message sent from the client at the start of the application, right after the WebSocket client opens the connection. The `FileReader` returns a list of files found locally on the machine and sends the list to the client. Found files are with the given suffix *“.h5”*. Setting user-defined suffix features can be added to the application if needed. The user then chooses the file they want to open, and the *openfile* message is sent but with *datasetname* field empty. The server invokes `Server.open_file()` method which scans the `.h5` file for datasets using

[2]https://docs.python.org/3/library/dataclasses.html

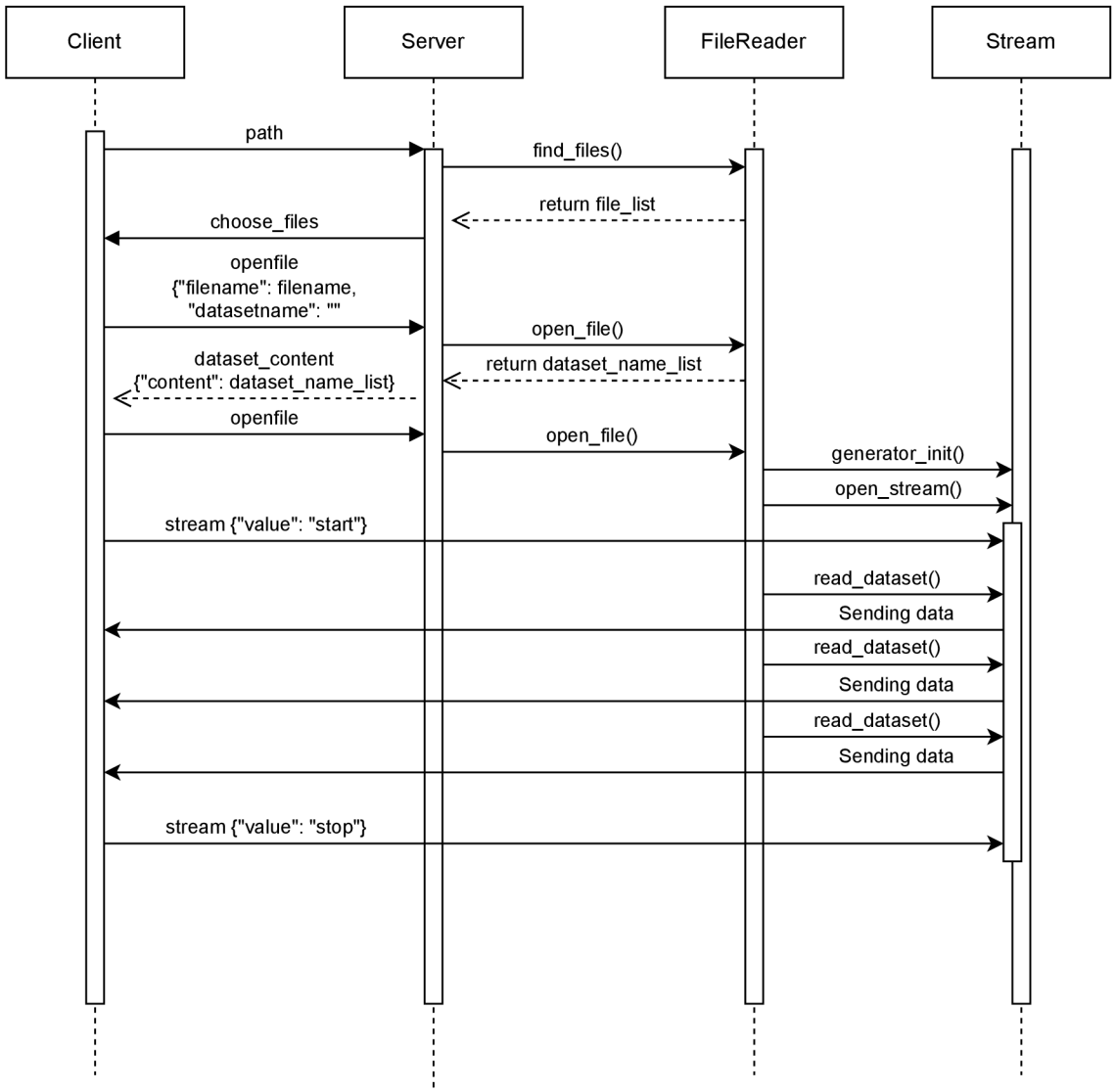

Fig. 4.1: UML diagram of the back-end.

`DASHDF5FileReader` class. The list of dataset names is then sent to the client and displayed for the user to choose. After the user chooses the dataset that interests them, a new message is sent to the server with the same properties, except this time, the *datasetname* value is filled with the chosen dataset name. Upon receiving the message the server invokes the `Server.open_file()`, which this time invokes the `Stream.generator_init()` and `Stream.open_stream()`.

The `Stream` class is implemented in *streaming.py* file. It implements reading the data using `DASHDF5FileReader` class; it sends the data to the client using WebSocket communication. The data streaming is running in a separate task, as discussed later.

The `Stream.generator_init()` saves the name of the file, the dataset to the `Stream` class and creates a local `DASHDF5FileReader` and invokes the `preprocess()` method, which looks for a preprocessed file, and if it does not find it runs preprocessing on the HDF5 file. Preprocessing is further explained in the Subsection 4.1.2. When the preparation is done, the `Server.open_stream()` method creates a stream task.

Streaming the data from server to client is done asynchronously using *asyncio* library[3]. A "private" method `Stream._create_stream_task()` initializes a property `streaming_task` which is an `asyncio.Task()` object with a callback function `Stream.stream_data()`. This registers a task that runs in a concurrent mode. This way, we can await new messages from the client and send the data we read from the file. This is done in a single thread by just using task switching.

When the `Stream.stream_data()` starts; it logs the fact that the stream is opened and starts to listen to the chunks of the dataset. The `read_dataset()` method is a Python generator object. This means it uses a word *yield* instead of *return* and passes values as needed. It also prepackages the data and encapsulates it in JSON format so the sending task is as short as possible so it does not block other tasks. There is little data processing still in place, but it takes place before sending the data, and as the data file is of acceptable size, it does not pose any risk of overflowing the memory. The data values are interpolated to the range between zero and one thousand, and the last step is to apply integer type to the data. These last stages of processing could be moved to preprocessing, but at the moment, they do not cause any performance penalty.

The async for loop gets the preprocessed data from the file reader class. To ensure that the *asyncio* scheduler has enough time to check for any received messages before sending any messages to the client, it puts itself to sleep by calling `await sleep(0)`. This way, incoming messages are not blocked.

The function then checks whether *Play* button was pressed on the client side of

[3]https://docs.python.org/3/library/asyncio.html

the application `await Stream.streaming_wait_event.wait()`. An instance of an `Event` class defined in *asyncio* is he `Stream.streaming_wait_event`, which is used to pause and play the data stream. `Event.set()` and `Event.clear()` are used for setting and clearing a waiting event. The `Event` class can be awaited as is shown on line 14 of the code snippet 4.1.

Listing 4.1: Data streaming function implementation.

```
1  async def stream_data():
2      print("Stream opened")
3      async for msg in DASHDF5FileReader.read_dataset():
4          # ensures that the scheduler has time to check
5          # received messages so that sending data
6          # does not block receiving messages
7          await sleep(0)
8
9          if not streaming_wait_event.is_set():
10             await websocket.send(
11                 dumps({
12                     "type": "ready"
13             }))
14             print("Stream waiting for an event...")
15         await streaming_wait_event.wait()
16         print(".", end="")
17         await websocket.send(msg)
```

4.1.2 Processing raw data from the DAS interrogator

The data generated by the OptaSense ODH-F Interrogator are saved in a HDF5 file format. Depending on the machine's setup - sampling rate, gauge length, and others. As shown further in 4.2.1, the output of a 10-second measurement with just 100 channels created a 52 MB file. It does not seem that much, but for longer measurements, this can be an issue. For the testing of data recorded, a person was running on the pavement near the school. The data is just under a minute long, and the HDF5 file is 2.7 GB big. This is not an amount that can be sent to the browser and displayed because it would drain the computer's RAM.

For data processing, I have taken a function from an existing project[4].

also tried to process the data during sending phase, and when the editing is simple enough, it is possible. Still, when more advanced processing is necessary, the whole process slows down the frequency the server can send the information making the whole application unusable.

[4]https://gitlab.com/optolab/das/data-viewer/-/blob/main/scripts/data-viewer.py

The function used for preprocessing the file is located in *spectral_analysis.py*. The function has been taken from an existing project for visualizing the DAS data using pure Python and the *matplotlib* library[5]. I have taken the function `spectral_analysis()` and used it to preprocess the data so there is less data to be displayed per second, and the data is more easily understandable.

Before preprocessing the data, it is good practice to retype it to a data type easier for Python to handle, like *numpy.float64*. In our tests, without retyping, the *spectral_analysis* algorithm takes ten times the time than with the retyping added. The function takes the data from the file as a `numpy.array()`; the other argument is the pulse rate that we get from an attribute in the HDF5 file, see Attachment B.3.

Firstly the function does a setup. It creates a *Hanning window* of the size of two to the power of *n*. *Hanning window* makes the data on the edges smaller. Next, it creates a list of frequency bin centers in cycles per unit of the sample spacing, starting from zero. From this list, it finds the minimum and maximum frequency. The function gets the dimensions of the dataset and calculates the number of blocks to go through from it `blocks=(rows-fsize)//shift`. The `shift` property is a step in the dataset for the for loop, and the `fsize` property is the size of the Hanning window.

After the setup, it loops through all the blocks in the dataset moved by the index of `shift`. The data is transposed, and *Hanning window* is applied to it and then transposed again because of the nature of matrix multiplication. Then the function calculates the spectrum coefficients for the data by computing a one-dimensional discrete Fourier transformation. From these coefficients, the algorithm discards half of it and leaves just one side of the spectrum and the zeroth coefficient. The power spectral density is then calculated, and as half of the spectrum is thrown away, it is applied back. Both sides are the same, so it was not needed before. Now as the power needs to be calculated, the half is restored. The power is a sum of all powers between the minimum and maximum frequency indexes calculated before. The last step of the analysis is to take the values and apply the logarithm scale to them.

$$psd = \frac{1}{pulse_rate * fsize} * |fft\_coefficients|^2 \tag{4.1}$$

The preprocessing ends by saving the data to a file by calling `numpy.save()` function. The file saves the variable contents returned by the spectral analysis function in binary format.

As an example, we took a one-minute-long measurement of a person running near the wire. The output file from the DAS interrogator was 2.7 GB in size. The

[5]https://gitlab.com/optolab/das/data-viewer/-/blob/main/scripts/data-viewer.py

function processed the file to just 7.5 MB file. Properties applied were: `fsize` was set to 4096, and `shift` was set to 4096.

4.2 Program implementation of HDF5 to WAV

This part of the project aims to create an application for reading data from the DAS system and converting the data to the `.wav` audio file format.

The opening of `.h5` files is done with `h5py` library[6]. For working with dataset data types, `numpy`[7] library is used. The function for interpolating arrays to a certain range is also used `interp`. Lastly, to convert the signal data to `.wav` audio format `scipy` library is used specifically `io` module function `wavfile`. To read all options and input arguments, the `argparse` library is used[8].

4.2.1 Reading HDF5 files

The sample file recorded in the OptaSense ODH-F is in HDF5 file format. As HDF5 files have a user-defined structure on the application layer and in the binary form, it is hard to say what is actually in the file. To better understand the contents of the `.h5` file, `h5dump` was performed and a conversion to JSON[9] was also performed by the `h5tojson` program[10]. The JSON file is quite large - the original HDF5 file is only 52.7 MB, and the JSON file is 946.6 MB. The dump text file is half the size and provides the same information, but the datasets are harder to understand, but the whole file is only half the size of the JSON file at "only" 420 MB. The JSON format is much easier to read. The structure of the file is divided into three parts:

- *apiVersion* - 1.1.1 version of API
- *datasets* - Contain all the datasets organized by their UUID[11] that are defined in the groups section. There is also an *alias* that is in the format of a Unix-based system path, `"/Acquisition/Raw[0]/Custom/SampleCount"` is an example. Other properties define the shape and type of stored data. In this case, the properties are shown in the table B.1.
- *groups* - Groups are named by a UUID. The group object has:
  - *alias* - Unix-like name; the first is root "/" group
  - *attributes* - define the type, name, and shape of the value of the attribute, which is a string `979bb2ac-99bf-4cb5-b410-5c16cd7872dc`

[6] `https://www.h5py.org/`
[7] `https://numpy.org/`
[8] `https://docs.python.org/3/library/argparse.html`
[9] `https://www.json.org/json-en.html`
[10] `https://hdf5-json.readthedocs.io/en/latest/tools/h5json.html`
[11] Universally unique identifier

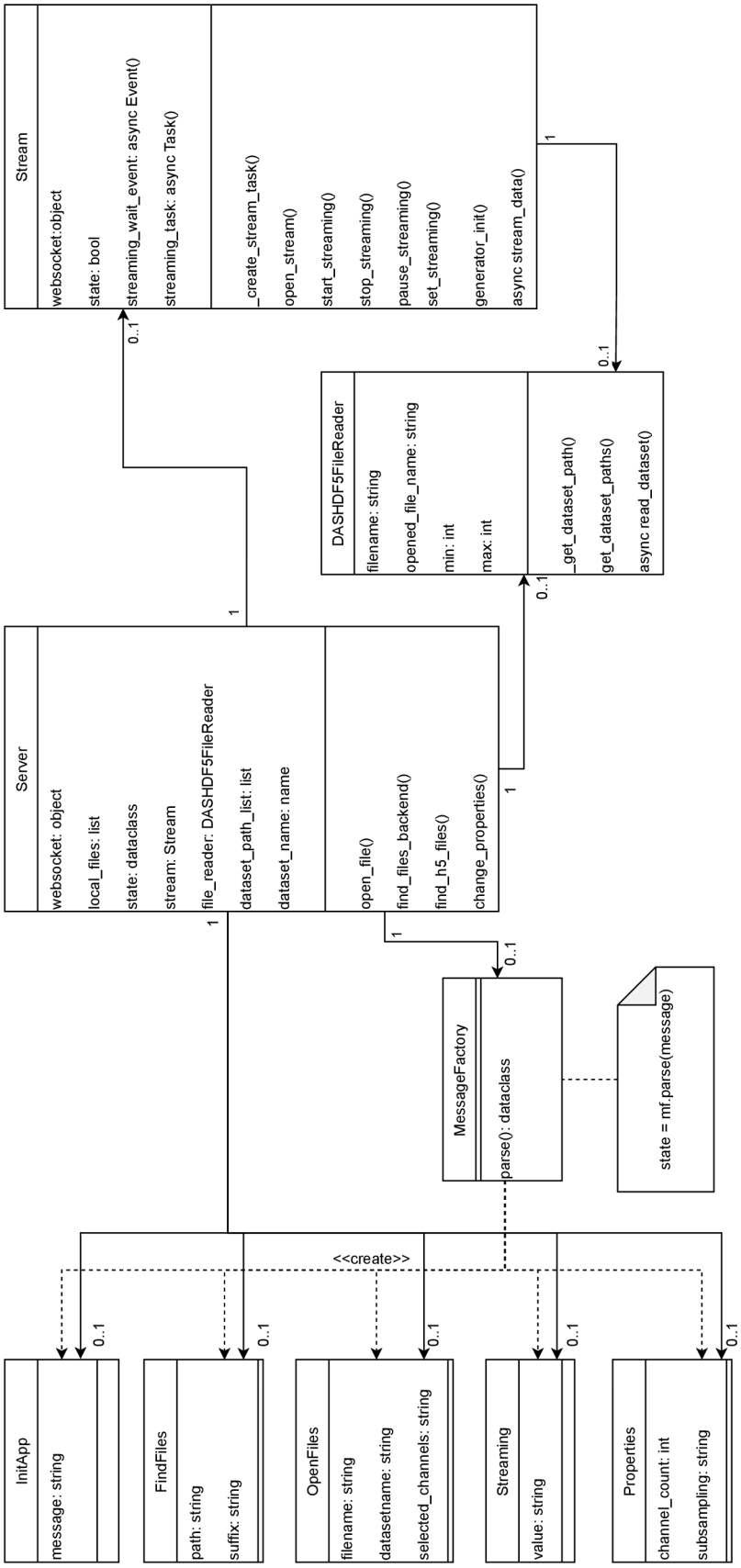

Fig. 4.2: UML diagram of the back-end.

– *links* - links to other groups that create a treelike structure. The link object contains the class of a link (e.g., H5L_TYPE_HARD for hard link), the *collection* property telling that it is a group, and the name of the group. The group object also contains other important metadata, such as measurement settings. All important details are given in the table B.1

There is a library for reading HDF5 files written in Python called `h5py`. To read the HDF5 file's contents, a function was created called `get_dataset_path()`, which recursively looks for all groups according to their name provided by the `.keys()` method, in the dataset. The result of this function is propagated through recursion and saved in Python `set()` built-in type. The user can then choose which one is the suitable dataset to use because there is probably more than one dataset. When calling the program, the user can save the string and use it as an argument. This saves time in searching for the contents of the file.

This is the data structure of the DAS file from the OptaSense Interrogator:

DAS output file structure in HDF5 format

```
/ ..... root
└── Acquisition ..... Recorded data
    ├── Custom ..... Empty
    └── Raw[0] ..... HDF5 group (3 members)
        ├── Custom ..... HDF5 group (1 members)
        │   └── SampleCount ..... HDF5 dataset, shape (332032,), type "<i8">
        ├── RawData[0] ..... HDF5 dataset, shape (100, 332032), type "<i2">
        └── RawDataTime ..... HDF5 dataset, shape (332032,), type "<i8">
```

The type of explanation in 4.2.1 is `i8`, which is `numberpy.int64`. `SampleCount` contains numbering of all samples, `RawDataTime` contains time, and `RawData[0]` contains sensor data we need to read in the following steps; see 4.1.2. More details are provided in Table B.1.

All important HDF5 attributes are shown in the table B.1. It contains metadata about the datasets, data dimensions, number of channels, kinds of filters used, time information, length of pulses, laser wavelength, and more. Some properties can be derived from those in the table. The capture duration can be calculated from the start and end of the capture, which is 10.376 s.

4.2.2 Data processing for converting raw HDF5 data to WAV

The data have 100 channels with N samples, in this example 332 032 samples saved in `/Acquisition/Raw[0]/RawData[0]`, see the file structure in 4.2.1. The function `scipy.io.wavfile.write()` saves samples to a file, and the data need more processing before the function can be called. After the data are read from the file, it is

Data type	Minimum value	Maximum value	WAV format
float32	-1.0	+1.0	32-bit floating-point
int32	-2147483648	+2147483647	32-bit PCM
int16	-32768	+32767	16-bit PCM
uint8	0	255	8-bit PCM

Tab. 4.1: WAV compatible types.

saved into `samples` variable of type `numpy.array` it is then processed in 4 steps as preparation for saving into `.wav` file. The steps are:

1. **Channel selection** - only channel one can be selected.
2. **Data interpolation** - The original data have a terrible value range from -24838 to -30758, triggering an exception when writing the data into a `.wav` file. The `numpy.interp()`[12] function interpolates the data into the range of the maximum and minimum values specified in this case by the 16 bit PCM[13], which can be written to WAV file by `wavfile` module.
3. **Resampling** - as the data are recorded at a certain sampling frequency, in this case, 10 kHz, resampling by the function `/scipy.signal.resample()`[14] is necessary. The right number of samples is calculated by the formula 4.2.
4. **Retyping** - the resampled data need to be in the correct format, and since the interpolation was done in the range of `int16`, the output type of choice is the same `samples.astype(np.int16)`.

$$numSamples = \frac{44100}{fs.len(samples)} \tag{4.2}$$

4.3 Frontend client application

As discussed in Section 3.3.1, the frontend is written in the Svelte framework. The application consists of three main parts - *waterfall component*, *properties column*, and a *footer*. *Waterfall* component is the center of the application. It renders the data and does the visualization. *Properties column* lets the user browse files, set properties, adjust values, and start and stop the stream, as will be discussed in the following sections 4.4 and 4.4.1.

[12] `https://numpy.org/doc/stable/reference/generated/numpy.interp.html`

[13] Pulse Code Modulation

[14] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample.html`

Python back-end file structure:

```
optasense_visualization_app
├── jsconfig.json ..... Application configuration
├── package.json ..... Dependencies with version
├── server.py ..... HTTP server
├── index.html ..... The root file
├── dist/ ..... Build application output folder
├── public/ ..... Images and media
└── src/ ..... Source files and components
    ├── App.svelte ..... Main component
    ├── main.js ..... Dependencies
    ├── store.js ..... Application stores
    ├── xAxis.js ..... D3.js x-axis definition.
    ├── app.css ..... Stylesheet document
    ├── assets/ ..... All media and images
    └── lib/ ..... Svelte components
        ├── canvas_properties.js ..... Stores and functions for canvas
        └── properties/ ..... Svelte components from the right panel
```

4.4 Svelte components

All Svelte components and JavaScript source files are saved in *src/* folder. The application component *App.svelte* defines the basic layout of the application. The layout itself is implemented using the *Bootstrap* library[15] in comparison to the *svelte-layouts* package components used in the Prototype, see Section 3.5. The layout was inspired by the *h5web* project; see Figure 2.4 and Section 2.6.3. The layout can be seen in Figure 4.3. There are three main parts of the application - the *Waterfall element* (defined in the file Waterfall.svelte), *Properties column*, and a *Footer element*.

4.4.1 Svelte stores

Svelte stores are JavaScript objects for storing values that must be shared between all the components. They also save the application state in a way. To use a store in a component and access its value developer needs a dollar notation, for example, `$store_name`. This way, the value saved in the store is accessed. Svelte stores are explained further in Section 3.3.1. There are two files defined with stores and functions. The files are *store.js* and *canvas-properties.js*.

[15] https://getbootstrap.com/

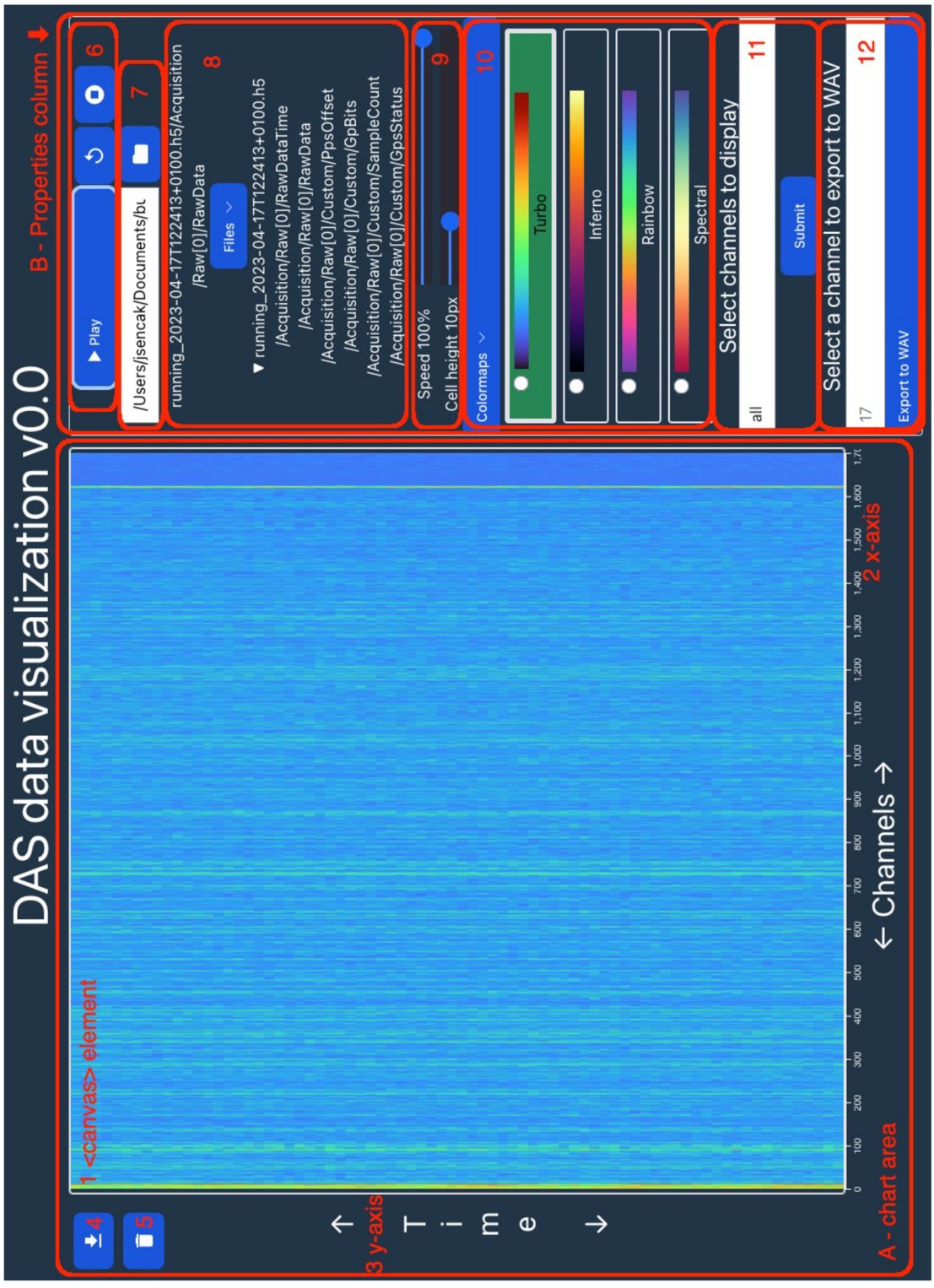

Fig. 4.3: File structure of the client side of the project.

The *store.js* file defines stores, functions, and variables for all components to use. Most notable are:

- `socket` - a WebSocket object for communication with the server side. There are also definitions for event listener for WebSocket communication *"open"* and receiving messages.
- `stream_state` - Boolean value signals whether the heatmap is redrawing.
- `data` - store for saving incoming data for display.
- names of files and datasets.
- properties for sliders, text input, and buttons.

The second file *canvas-properties.js* defines stores and variables for the `<canvas>` element. Most notable:

- `width` and `height` canvas dimensions.
- `channel_number` - number of channels, or number of rectangles on the x-axis.
- `cellwidth` - a derived store defined as $*width*/$*channel_number*.
- `colormap` - definition of active D3js colormap, default is Turbo, see Section 3.4.4 regarding colormaps.

4.4.2 Stylesheet with dark and light mode

Modern browsers provide users with dark or light modes, so they can change the mode according to surrounding light sources and brightness. To change the application's style, variables were defined in the *app.css* file. The stylesheet checks for the set color scheme and defines variables accordingly. It can be seen in the 4.4.2.

```
@media (prefers-color-scheme: light) {
  :root {
    --background-color: #ffffff;
    --color: #213547;
    --select-text-color: black;
    --footer-bg-color: white;
  }
}
@media (prefers-color-scheme: dark) {
  :root {
    --background-color: #213547;
    --color: white;
    --select-text-color: black;
    --footer-bg-color: rgb(34, 38, 53);
  }
}
```

4.4.3 Heatmap chart rendering to a canvas element

The heatmap chart rendering part of the application can be seen in Figure 4.3, letter A - chart area, elements 1 to 5. The Svelte component is defined in the *Waterfall.svelte* file. The *Waterfall* component consists of four main parts:

- Y-axis element - There is a "time" label and two buttons. One button is for downloading the contents of the `<canvas>` element. To save canvas content, a link is created using `canvas.toDataURL("image/png")`. The second button for clearing the canvas from the visualization.
- X-axis element - Consists of an x-axis element defined by D3js `axisBottom()`, and it displays the number of channels the DAS system is watching over. The definition is in the `xAxis.js` file.
- Canvas.svelte - See canvas rendering 3.4.3

The chart is a `<canvas>` element. Drawing to canvas uses the `draw()` function defined in the *Canvas.svelte* file. The snippet from the implementation can be seen in the Listing 4.2. First, `$chart_row_num` is set to the number of channels fitting into the frame. It is derived from the formula $(\$height/\$cellHeight) + 1$. The plus one is there to round the number up as the result of division is in float and would be rounded down. This way, we round up without calling any function. Either the number of rows is the length of the dataset when the number of rows is smaller than the number of cells that fit the graph, or it is the maximum value of rows possible to display. See Listings 4.2.

The rendering of a heatmap chart is done by drawing rectangles in a two-dimensional plane on a canvas element. Rectangles are drawn by a `fillRect()` function. It takes four arguments - coordinates of the top left corner and the bottom right corner. To apply a style, in this case, a color from a colormap saved in the `$colormap` store, we call the function `getFill()`, with the value we want to display. The returned value is the color of the rectangle.

Listing 4.2: Implementation of drawing to canvas.

```
function getFill(value) {
    // returns colormap from the $colormap store
    return $colormap(colorScale(value));
}

function draw() {
    $chart_row_num = ($height/$cellHeight)+1;
    let row_number;
    if ($data_store.length < $chart_row_num) {
        row_number = $data_store.length;
    } else {
        row_number = $chart_row_num-1;
    }
    for (let y = 0; y < row_number; y++){
        // get latest data available
        dataRow = $data_store[y];

        // position of the rectangle on the y-axis
        posY = y * $cellHeight;

        // loop through all values in the row
        for (let x = 0; x < dataRow.length; x++){
            // position of the rectangle on the x-axis
            posX = x * $cellWidth;

            // get color for a~rectangle
            ctx.fillStyle = getFill(dataRow[x]);

            // draw a~rectangle
            ctx.fillRect(
                posX,
                posY,
                posX + $cellWidth,
                $cellHeight
            );
        }
    }
}
```

Under the data visualization is an x-axis. It is made using the D3js framework for visualizing SVG graphics. When creating an object using D3.js, the first is to call `d3.select` so the framework knows where to append the SVG object. We set properties like width and height. The axis is created by calling `d3.axisBottom()`. Then the scale is applied to set values from lowest to highest, and lastly, it is important to set the number of `ticks`, which tells the graph how many points

should be displayed on the axis. To perfectly align the axis with the graph, the axis is translated by five pixels to the right. The implementation is shown in Listing 4.3.

Listing 4.3: D3js x-axis implementation.

```
<script>
    let chart = d3.select("#xAxis")
                .append("svg")
                .attr("width", $width + "px")
                .style("font-size","50px")
                .attr("height", 20);

    scale = d3.scaleLinear()
              .domain([0, $channel_number])
              .range([0, $width]);
    x_axis = d3.axisBottom()
                .scale(scale)
                .ticks($channel_number/5)

    chart.append("g")
        .call(x_axis)
        .attr("transform", "translate(" + 5 + ",0)");
</script>
<div id="xAxis"></div>
```

4.4.4 Setting the speed property

To set the speed property, a component *Speed_slider.svelte* was created. It is a slider with a command trigger `on:mouseup` which triggers a function `handleMouseup`. This function sets a store variable `$speed`. It also (re)sets the update interval by clearing the last interval and setting a new value; see 4.4. The `setInterval()` function takes two arguments, the first is a callback, in our case, it is an increment on a value, and the second is a time value in milliseconds. The function `countDelay()` was implemented in *canvas_properties.js* that takes the speed slider value, which has values between zero and hundred, and applies a scale with new values from one to one thousand milliseconds.

Listing 4.4: Svelte reactive statement for redrawing canvas element.

```
// Canvas.svelte component
$: {
     $last_row_number;
     draw()
}



// canvas_properties.js
let idx = 0;
export function increment(){
    idx++;
    last_row_num.set(idx);
}

//Speed_slider.svelte
const handleMouseup = () => {
    $speed = slider_val;
    if ($stream_state) {
        clearInterval($intervalID);
        $intervalID = setInterval(increment, countDelay());
    } else {
        clearInterval($intervalID);
    }
};
```

4.4.5 Properties column items explained

The second part of the application is *Properties column* where the user can start and stop the visualization animation, choose what files will be opened, set visualization properties like speed and cell height, choose a colormap for better understanding the data, select what channels will be displayed and which channel will be exported to WAV audio format. Each of these properties has its own Svelte component created in *src/properties* folder, except for *FileBrowser.svelte*.

Starting with steam control buttons, see Figure 4.3. There are three buttons to control the heatmap animation. Each button has its component file with its implementation - *Play.svelte*, *Stop.svelte* and *Restart.svelte*. There is a handler function associated with each button that handles input and sets the `$stream_state` store, which starts and stops the animation by stopping the data from being sent

from the server. The *Play* button is disabled by default and enabled once a dataset is ready to be sent from the server.

The *OpenFiles.svelte* defines a text input, which sets the `$directory_store` that saves the path to the HDF5 file on the device the server is running on. By clicking a button, the client sends a message requesting a list of all files with an *.h5* suffix. See Figure 4.3 item number 7.

The *FileBrowser.svelte* component creates a tree-like structure for opening and closing the files and their datasets. There is a button on the top that can make the component hide or appear. By setting a path in the text input, item number 7 in Figure 4.3, all the files listed below the "Files" button. By clicking a filename, the client sends a WebSocket message to the server, which looks for the chosen file's datasets. The server returns the list of datasets. The user then clicks on the desired dataset, and a new message is sent to the server now opening the file and the dataset and running the preprocessing stage, as explained in Section 4.2.2. During the preprocessing, a loading wheel element is shown to tell the user to wait. After loading the file, the *Play* button is automatically enabled.

Then there are two sliders - one sets the speed and one the height of each of the rectangles in the heatmap; see Figure 4.3 item number 9.

The user can select a colormap from the element; see Figure 4.3 item number 10. The definition is in *Colormap.svelte*. There are four to choose from *Turbo*, *Inferno*, *Rainbow*, and *Spectral*.

Item number 11 enables the user to choose which channels should be displayed. The values can be "all", or numbers divided by a semicolon (";"). Setting value ranges is also possible by putting a number followed by a dash ("-") and a second number. Setting incorrect values results in applying either by selecting all channels.

Lastly, there is the element number 12, which lets the user select a channel, which will be converted to audio.

4.5 Testing the application

We tested the application on the data we acquired from the OptaSense ODH-F Interrogator. The data is saved in HDF5 file format. The machine setup can be seen in the Attachment B.3. We got the properties by running a Python script, which is part of the backend *attibutereader.py*. As the HDF5 file structure depends on the interrogator setup, we provide only a hardcoded version. It should serve an example for acquiring attributes from HDF5 files using *h5py* Python library.

The data recorded had a person running on a pavement near our school. Optical fiber is buried around half a meter under the ground and about a meter from the pavement. Our data visualization software can display the data and play them to

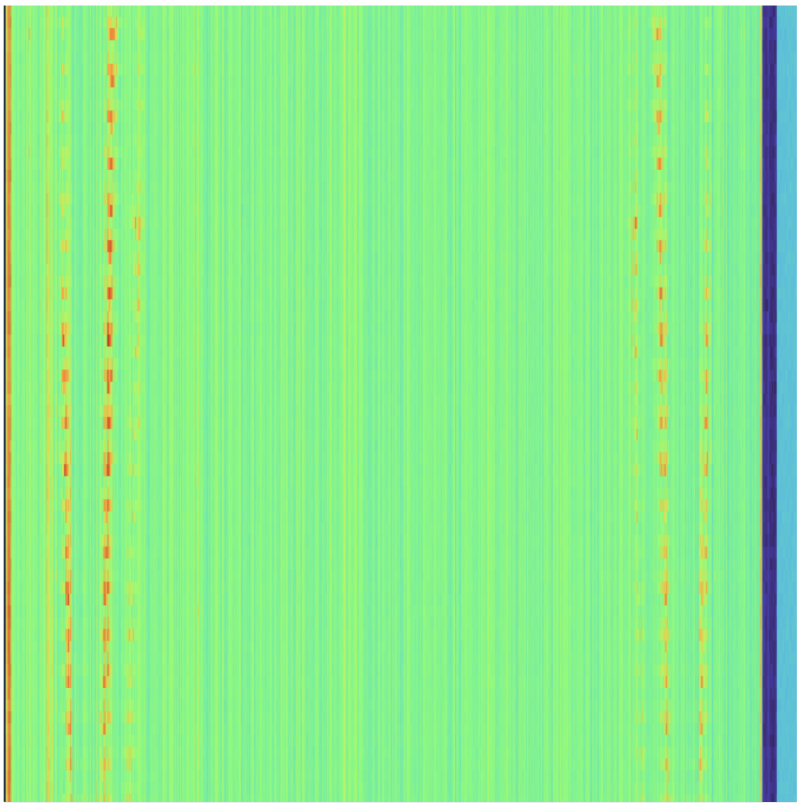

Fig. 4.4: The heatmap data visualization. Someone is running along the buried fiber optic cable.

the user. Thanks to the spectral analysis function, the running is visible in the visualization. The visualization is shown in Figure 4.4.

Conclusion

The main goal of this work was to study the DAS system and its output files in the HDF5 file format. As for the implementation part, the goal was to create a multi-platform application capable of displaying the data from the DAS interrogator, processing it, and displaying it in a suitable form. The visualization should be able to play and pause the animation of the flowing data, also with the ability to convert the data into an audio file.

In the theoretical part, the principles of optical reflectometry were explained, as well as different types of light scattering - Mie scattering, Rayleigh scattering, Raman scattering, and Brillouin scattering. the inner workings of the DAS system and methods like OTDR and Φ-OTDR were also studied. The output file format HDF5 was carefully examined with objects such as Datasets, Groups, Attributes, and Links. The output file structure of the DAS system OptaSense ODH-F was shown in a readable form. The existing technology was studied with available software for the front-end and server side. Real-time capabilities were also discussed with the data bandwidth requirements. A prototype was created and written in Svelte to showcase the application's design with a waterfall graph and HTML elements to set properties for display.

In the design section, we discuss software requirements and the basic back-end technologies used later in the implementation section, such as WebSockets and reading HDF5 files. The front-end technologies for data visualization based on Canvas and SVG graphics were discussed. The basics of the Svelte framework were introduced.

A web application was created with a server written in Python and a client using the Svelte framework. The back end can read and process the data in HDF5 file format. The back-end communicates with the client side using WebSockets. A simple message system was created for this purpose. The client allows the user to choose the file and the dataset, as well as properties like speed choosing a colormap and channels to display. The visualization can be started, stopped, and replayed. There is also a feature to download the image currently on display.

The application in this state is a one-page website; the next step would be to incorporate it into the SvelteKit framework, allowing page routing. This should be pretty straightforward. Future work can include further improving the visualization, including zooming and data selection features, either by creating custom `<canvas>` rendering, SVG rendering with D3js, or using existing libraries such as Plotly. It might be useful to have multiple algorithms to choose from when processing the raw DAS data. Although we tried a few methods for data processing, we have chosen one, and there is no selection possible at this stage.

Bibliography

[1] SHAN, Yuanyuan, Jiayun DONG, Jie ZENG, Siyi FU, Yinsen CAI, Yixin ZHANG, and Xuping ZHANG. A Broadband Distributed Vibration Sensing System Assisted by a Distributed Feedback Interferometer. IEEE Photonics Journal [online]. 2018, 10(1), 1-10 [cit. 2023-04-17]. ISSN 1943-0655. Accessible at: doi:10.1109/JPHOT.2017.2776919

[2] ZYCZKOWSKI, Marek, Edward M. CARAPEZZA, Mieczyslaw SZUSTAKOWSKI, Norbert PALKA a Marcin KONDRAT. Fiber optic perimeter protection sensor with intruder localization [online]. In: . 2004-11-30, 71- [cit. 2023-04-18]. Accessible at: doi:10.1117/12.578186

[3] FILKA, Miloslav. Optické přenosy informací pro integrovanou výuku VUT a VŠB-TUO. Brno: electronic, 20114. ISBN 978-80-214-5064-6.

[4] RAO, Yunjiang, Zinan WANG, Huijuan WU, Zengling RAN and Bing HAN. Recent Advances in Phase-Sensitive Optical Time Domain Reflectometry (Φ-OTDR). Photonic sensors (Berlin) [online]. Singapore: Springer Singapore, 2021, 11(1), 1-30 [cit. 2023-05-05]. ISSN 1674-9251. Accessible at: doi:10.1007/s13320-021-0619-4

[5] WOODWARD, Bill, and Andrew OLIVIERO. Cabling The Complete Guide to Copper and Fiber-Optic Networking. 5th edition. Indianapolis, Indiana: John Wiley, 2014. ISBN 978-1-118-80732-3.

[6] WEIK, Martin H. Scattering center. In: WEIK, Martin H. Computer Science and Communications Dictionary [online]. Boston, MA: Springer US, 2001, 2000-11-30, s. 1522-1522 [cit. 2023-04-19]. ISBN 978-0-7923-8425-0. Accessible at: doi:10.1007/1-4020-0613-6_16662

[7] Svelte documentation. Documentation [online]. 2023 [cit. 2023-04-20]. Accessible at: <`https://svelte.dev/docs`>

[8] POSPÍŠILOVÁ, Marie, Gabriela KUNCOVÁ, and Josef TRÖGL. Fiber-Optic Chemical Sensors and Fiber-Optic Bio-Sensors. Sensors [online]. 2015, 15(10), 25208-25259 [cit. 2023-04-18]. ISSN 1424-8220. Accessible at: doi:10.3390/s151025208

[9] WANG, Yu, Baoquan JIN, Yuncai WANG, Dong WANG, Xin LIU, Qing BAI: *Real-Time Distributed Vibration Monitoring System Using Φ-OTDR.* IEEE sensors journal [online]. PISCATAWAY: IEEE, 2017, 17(5), 1333-1341 [cit. 2022-11-22]. ISSN 1530-437X. Accessible at: doi:10.1109/JSEN.2016.2642221

[10] KISLOV, K. V., and V. V. GRAVIROV: *Distributed Acoustic Sensing: A New Tool or a New Paradigm. Seismic instruments* [online]. Moscow: Pleiades Publishing, 2022, 58(5), 485-508 [cit. 2022-11-22]. ISSN 0747-9239. Accessible at: doi:10.3103/S0747923922050085

[11] MATSUMOTO, Hiroyuki, Eiichiro ARAKI, Toshinori KIMURA, et al. Detection of hydroacoustic signals on a fiber-optic submarine cable. Scientific reports [online]. England: Nature Publishing Group, 2021, 11(1), 2797-2797 [cit. 2023-04-22]. ISSN 2045-2322. Accessible at: doi:10.1038/s41598-021-82093-8

[12] PARKER, Tom, SHATALIN, Sergey, FARHADIROUSAN Mahmoud: *Distributed Acoustic Sensing – a new tool for seismic applications.* Earthdoc [online]. [cit. 2022-11-22]. Accessible at : https://doi.org/10.3997/1365-2397.2013034

[13] MERKLEIN, Moritz, Irina V. KABAKOVA, Atiyeh ZARIFI, and Benjamin J. EGGLETON. 100 years of Brillouin scattering: Historical and future perspectives. Applied Physics Reviews [online]. 2022, 9(4), 41306 [cit. 2023-04-21]. ISSN 1931-9401. Accessible at: doi:10.1063/5.0095488

[14] PALMIERI, Luca and Luca SCHENATO. Distributed Optical Fiber Sensing Based on Rayleigh Scattering. The Open Optics Journal. 2013, 7(1), 104-127. Accessible at: doi:10.2174/1874328501307010104

[15] SHAN, Yuanyuan, Jiayun DONG, Jie ZENG, Siyi FU, Yinsen CAI, Yixin ZHANG and Xuping ZHANG. A Broadband Distributed Vibration Sensing System Assisted by a Distributed Feedback Interferometer. IEEE Photonics Journal [online]. 2018, 10(1), 1-10 [cit. 2023-05-04]. ISSN 1943-0655. Accessible at: doi:10.1109/JPHOT.2017.2776919

[16] WAIT, P.C, K DE SOUZA and T.P NEWSON. A theoretical comparison of spontaneous Raman and Brillouin based fibre optic distributed temperature sensors. Optics communications [online]. AMSTERDAM: Elsevier B.V, 1997, 144(1), 17-23 [cit. 2023-05-12]. ISSN 0030-4018. Accessible at: doi:10.1016/S0030-4018(97)00482-3

[17] BOYD, Robert W and Debbie PRATO. Nonlinear Optics (3rd Edition). San Diego: Elsevier, 2008. ISBN 0123694701. Accessible at: doi:10.1016/B978-0-12-121682-5.X5000-7

[18] AYANA, Lamessa Abebe, Qi CHU, Yulin PEI, Liqiang QIU, Dexin BA and Yongkang DONG. Distributed optical fiber sensing based on the combination of Brillouin and Rayleigh scattering [online]. In: . SPIE, 2021, 118500D-118500D-5 [cit. 2023-05-13]. ISBN 9781510645448. ISSN 0277-786X. Accessible at: doi:10.1117/12.2599257

[19] GABAI, Haniel, and AVISHAY Eyal: *On the sensitivity of distributed acoustic sensing.* Optics letters vol. 41,24 (2016): 5648-5651. [online]. [cit. 2022-11-22] doi:10.1364/OL.41.005648

[20] WANG Z, LU B, YE Q, CAI H.: *Recent Progress in Distributed Fiber Acoustic Sensing with Φ-OTDR.* Sensors (Basel). 2020 Nov 18;20(22):6594. doi: 10.3390/s20226594. PMID: 33218051; PMCID: PMC7698859.

[21] KISLOV, K. V., and V. V. GRAVIROV.: *Distributed Acoustic Sensing: A New Tool or a New Paradigm. Seismic instruments* [online]. Moscow: Pleiades Publishing, 2022, 58(5), 485-508 [cit. 2022-11-25]. ISSN 0747-9239. Accessible at: doi:10.3103/S0747923922050085

[22] *The HDF5 Data Model and File Structure* HDF Group [cit. 2022-12-08]. Accessible at <`https://docs.hdfgroup.org/hdf5/develop/_h5_d_m__u_g.html`>.

[23] HEBER G.: *RESTful HDF5* The HDF Group [cit. 2022-12-08]. <`https://support.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf`>.

[24] HORNMAN, J.C.: *Field trial of seismic recording using distributed acoustic sensing with broadside sensitive fibre-optic cables.* Geophysical Prospecting [online]. 2017, 65(1), 35-46 [cit. 2022-12-06]. ISSN 0016-8025. Accessible at: doi:10.1111/1365-2478.12358

[25] BAO, Xiaoyi, and Liang CHEN.: *Recent Progress in Distributed Fiber Optic Sensors.* Sensors (Basel, Switzerland) [online]. BASEL: Mdpi, 2012, 12(7), 8601-8639 [cit. 2022-12-08]. ISSN 1424-8220. Accessible at: doi:10.3390/s120708601

[26] PALMIERI, Luca; SCHENATO, Luca. Distributed optical fiber sensing based on Rayleigh scattering. The Open Optics Journal, 2013, 7.1.

[27] OptaSense OS6 Visualization Software Demo. YouTube, uploaded by OptaSense, 22 June 2021, <`https://www.youtube.com/watch?v=6-hj1ySERIA`>.

[28] Melnikov, A., Fette, I.: *The WebSocket Protocol.* RFC Editor. [online] [cit. 2022-08-12]. Accessible at: <`https://doi.org/10.17487/RFC6455`>.

[29] Choosing Colormaps in Matplotlib. Choosing Colormaps in Matplotlib [online]. The Matplotlib development team, c2002–2012 [cit. 2023-04-29]. Accessible at: <https://matplotlib.org/stable/tutorials/colors/colormaps.html>

[30] Os7500 Optical Accelerometer [online]. [cit. 2023-05-05]. Accessible at: <https://lunainc.com/product/os7500>

[31] The fiber-optic gyroscope: Herve Lefevre Artec House, 1993, ISBN 0-89006-537-3, pp. 300, £65. Optics and Laser Technology [online]. Elsevier, 1993, 25(6), 406-406 [cit. 2023-04-22]. ISSN 0030-3992. Accessible at: doi:10.1016/0030-3992(93)90010-D

[32] What is Use Case Diagram? [online]. 2022 [cit. 2023-05-05]. Accessible at: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>

[33] What is a REST API? [online]. IBM, 2023 [cit. 2023-05-06]. Accessible at: <https://www.ibm.com/topics/rest-apis>

[34] HUSAR, Alex. How to use REST APIs [online]. Oakland, CA 94607: Free Code Camp, 2023 [cit. 2023-05-06]. Accessible at: <https://www.freecodecamp.org/news/how-to-use-rest-api/>

[35] *Cross-Origin Resource Sharing (CORS)* mozilla.org [cit. 2022-12-08]. Accessible at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.

[36] Illustrative spectra of light scattering in optic fibers. Springer Link [online]: Springer [cit. 2023-04-25]. Accessible at: <https://link.springer.com/article/10.3103/S0747923922050085/figures/6>

Symbols and abbreviations

ADM	Abstract Data Model
API	Application Programming Interface
BOTDR	Brillouin Optical Time Domain Reflectometry
BOTDA	Brillouin Optical Time Domain Analysis
D3.js	Data-Driven Documents, a JavaScript framework
C-OTDR	Coherent Φ-OTDR
DAS	Distributed Acoustic Sensing
DOM	Document Object Model
DSS	Distributed Strain Sensing
DTS	Distributed Temperature Sensing
DVS	Distributed Vibration Sensing
DWDM	Dense Wavelength Division Multiplexer
EDFA	Erbium-doped Fiber Amplifier
FFT	Fast Fourier Transformation
HDF5	Hierarchical Data Format v5
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MEMS	Microelectromechanical Systems
OFDR	Optical Frequency Domain Reflectometry
Φ-OTDR	Phase-Sensitive Optical Time Domain Reflectometry
OTDR	Optical Time Domain Reflectometry
PCB	Printed Circuit Board
PZT	Lead Zirconate Titanate

REST	Representational State Transfer
SBS	Stimulated Brillouin Scattering
SPS	Samples Per Second
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
UML	Unified Modelling Language

List of appendices

A Installing dependencies

Using a virtual environment is suggested and installing all dependencies in the environment.

```
python3 -m venv env
source env/bin/activate
pip3 install -r requirements
```

A.1 Optasense visualizer application usage

To run the application it is necessary to have an environment set up and all dependencies installed.
Run the server part of the application:

```
python3 app.py
```

The application usage:

```
usage: app.py [-h] [--port PORT]

DAS file visualization software. Server application.

options:
  -h, --help   show this help message and exit
  --port PORT  Port number for websocket.
```

A.2 Svelte

To create a Svelte application from template:

```
npm create svelte@latest myapp
```

Run the client on the localhost:

```
npm run dev
```

To run a build call:

```
npm run build
```

The output will be saved in */dist* folder. To run the application from there, run Python script in the project's client folder:

```
python3 server.py
```

Server created this way is a simple Flask server.

B OptaSense DAS system measurement properties

Property	Value
/Acquisition	
MaximumFrequency	16000
MinimumFrequency	0
NumberOfLoci	100
PulseRate	32
PulseWidth	20 ns
SpatialSamplingInterval	1.020 95 m
StartLocusIndex	600
/Acquisition/Custom	
Data Width (Bits)	16 bits
FPGA infromation	Type and settings
FPGA Drawing Number	4802701
FPGA Version	1.2
Fibre Refractive Index	1.468199968338
GPS Enabled	True
Laser Wavelength (nm)	1550
Num Output Channels	100
Ping Period (CSU)	3125
Pulse Width (CSU)	2
Sequencer Clock (MHz)	100 MHz
Sequencer Mode	1
/Acquisition/Raw[0]	
OutputDataRate	32000
NumberOfLoci	100
RawDescription	Single Pulse, SR: 1.5, OCP: 1
StartLocusIndex	600

Tab. B.1: HDF5 groups and their attributes from the data file.

Property	Value
/Acquisition/Raw[0]/Custom/SampleCount	
Filters	shuffle and deflate
maximum dimensions	6451612
shape (number of samples)	332032
type	64-bit integer
/Acquisition/Raw[0]/RawData	
Count number of all values	33203200
Start time	2021-08-31T16:22:39.739250Z
End time	2021-08-31T16:22:50.115218Z
Filters	shuffle and deflate
type	64-bit integer
Shape	100 channels 332032 samples
/Acquisition/Raw[0]/RawDataTime	
Start time	2021-08-31T16:22:39.739250Z
End time	2021-08-31T16:22:50.115218Z
Filters	shuffle and deflate
start index	0
shape	332032
type	64-bit integer

Tab. B.2: HDF5 groups and their attributes from the data file.

Property	Value
/Acquisition	
MaximumFrequency	10000.0
MinimumFrequency	0.0
NumberOfLoci	1700
PulseRate	20000.0
PulseWidth	20.0
PulseWidthUnit	ns
SpatialSamplingInterval	1.0209523
SpatialSamplingIntervalUnit	m
StartLocusIndex	0
TriggeredMeasurement	false
VendorCode	OptaSense IU Setup 1.8.6 c9c23ad2ab20eb82641984e0e1205500228e9fba
schemaVersion	2.0
uuid	bb17e24c-0c96-44d0-b4b0-a25a72db10e2
Filters	shuffle and deflate
maximum dimensions	6451612
shape (number of samples)	332032
type	64-bit integer
/Acquisition/Raw[0]	
Count number of all values	33203200
NumberOfLoci	1700
OutputDataRate	20000.0
RawDescription	b'Single Pulse, SR: 1.5, OCP: 1'
StartLocusIndex	0
uuid	b'34baa19e-cdde-48d7-8430-fbc9eb2187e6'
/Acquisition/Raw[0]/RawData	
Count	1938000000
Dimensions	array([time, locus], dtype='|S6')
PartEndTime	2023-04-17T11:25:10.456750Z
PartStartTime	2023-04-17T11:24:13.456800Z
StartIndex	0

Tab. B.3: HDF5 groups and their attributes from the running data file.

C Printing the HDF5 data structure and metadata

```
h5dump -n running_2023-04-17T122413+0100.h5

HDF5 "running_2023-04-17T122413+0100.h5" {
    FILE_CONTENTS {
     group      /
     group      /Acquisition
     group      /Acquisition/Custom
     group      /Acquisition/Raw[0]
     group      /Acquisition/Raw[0]/Custom
     dataset    /Acquisition/Raw[0]/Custom/GpBits
     dataset    /Acquisition/Raw[0]/Custom/GpsStatus
     dataset    /Acquisition/Raw[0]/Custom/PpsOffset
     dataset    /Acquisition/Raw[0]/Custom/SampleCount
     dataset    /Acquisition/Raw[0]/RawData
     dataset    /Acquisition/Raw[0]/RawDataTime
  }
}
```