

MSc thesis in Geomatics

Implementation of OGC SensorThings API standard for the integration of dynamic sensor data in a 3D urban digital twin

Eleni Theodoridou 2023



MSc thesis in Geomatics

**Implementation of OGC SensorThings API
standard for the integration of dynamic
sensor data in a 3D urban digital twin**

Eleni Theodoridou

October 2023

A thesis submitted to the Delft University of Technology in
partial fulfillment of the requirements for the degree of Master
of Science in Geomatics

Eleni Theodoridou: *Implementation of OGC SensorThings API standard for the integration of dynamic sensor data in a 3D urban digital twin* (2023)

© ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The data preparation and pipeline implementation code can be accessed at the following link: [Thesis GitHub Repository](#).

The work in this thesis was carried out in the:



Geo-Database Management Centre
Delft University of Technology

Supervisors: Prof.dr. Azarakhsh Rafiee
Prof.dr. Martijn Meijers
Co-reader: Dr.ir. G.A.K. Arroyo Ohori

Abstract

Urban areas are home to over half of the global population and consume significant natural resources, necessitating effective city management for urban resilience and sustainability. Resilient cities are better prepared to face challenges, from natural disasters to economic downturns, and adapt to climate change, resource scarcity, and population growth. Urban digital twins, which are virtual replicas of cities integrating data from various sources, play a pivotal role in enhancing and supporting urban resilience. These digital twins empower urban planners, designers, and the public to make informed decisions, identify vulnerabilities, and respond to acute shocks in real-time. Leveraging technologies like the Internet of Things (IoT) and 3D city models, urban digital twins offer a dynamic representation of the city's functions, allowing for better visualization, monitoring, and decision-making. However, challenges related to interoperability and data acquisition need to be addressed to maximize the potential of urban digital twins.

This thesis investigates how environmental sensor data can be collected, processed, and integrated into 3D city models to visualize the dynamic elements of a city comprehensively. It explores the use of international standards, such as the OGC SensorThings API standard to acquire, store, and manipulate real-time and historical crowd-source, sensor data in a consistent and interoperable manner. The concepts of sensor data pre-processing and interpolating are also explored through OGC standards, in the context of providing detailed spatio-temporal information for the urban environment, while the results are visualized in a 3D web application.

Keywords: urban digital twins, OGC, standards, SensorThings API

Acknowledgements

I want to extend my sincere thanks to my supervisors, Azarakhsh Rafiee and Martijn Meijers. Your support and guidance have been incredibly important to me throughout this thesis. Your expertise and patience have made a significant difference in my academic journey, and I am grateful for your mentorship.

Parents, without your support, understanding and endless encouragement, I could never make it in this new chapter in my life. Thank you for being my pillars of strength, unconditional love and acceptance.

Staurou and Marilou, thank you for believing in me, even in times I could not and for cheering me up when it was most needed. I am so blessed to have you in my life and thank you for proving that friendships are more important than distance.

Ioanna, Christos, Adele, Chrysanthi, Pelagia and Faidon, thank you for all the late (study) nights, the long conversations and for sharing all these new experiences. You have helped me to grow, while creating a home away from home.

Contents

1. Introduction	1
1.1. Motivation	2
1.2. Research Questions	3
1.3. Thesis outline	4
2. Theoretical Background and Related work	5
2.1. Urban Digital Twins	5
2.2. Geospatial data and sensor integration	6
2.2.1. Open Geospatial Consortium (OGC) standards	6
2.2.2. Smart Emission initiative	10
2.2.3. Integrating Sensor Web With Spatial Data Infrastructures	11
2.2.4. OGC 3D-IoT Platform for Smart Cities	13
2.3. Environmental Sensor Data	14
2.3.1. Crowd-sourced data for urban studies	14
3. Methodology	17
3.1. Methodology Overview	17
3.2. Sensor data	17
3.2.1. Real time sensor data	18
3.2.2. Historical sensor data	19
3.3. SensorThings API standard for data accessing	20
3.4. Historical sensor data pre-processing	20
3.5. Sensor data interpolation	24
3.6. Publishing the interpolation results using OGC standards	26
3.7. Preparation of the 3D city model for CesiumJS	27
3.7.1. 3D city models	27
3.7.2. OGC 3D Tiles	27
3.8. Visualization in CesiumJS	28
3.8.1. Loading 3D city models in CesiumJS through Cesium ion	29
4. Implementation	31
4.1. Tools and datasets	31
4.2. Study Area	33
4.3. Real-time sensor data pipeline	33
4.3.1. Acquisition of real-time sensor data	34
4.3.2. Integrate the SensorThings API standard using FROST-Server	34
4.3.3. Data interpolation	36
4.3.4. OGC API Processes	36
4.3.5. Publishing the interpolation results	37
4.3.6. Accessing the interpolated data through CesiumJS	40
4.4. Historical sensor data pipeline	40
4.4.1. Acquisition of historical sensor data	40

Contents

4.4.2. Storing in FROST-Server the historical sensor data	41
4.4.3. Historical data cleaning	41
4.4.4. Data Interpolation	42
4.4.5. Publishing the interpolation results	44
4.4.6. Accessing the interpolated data through CesiumJS	46
4.5. 3D city models to 3D Tiles	46
5. Results and Discussion	49
5.1. Netatmo data acquisition	49
5.2. Historical data cleaning	49
5.3. Data interpolation	51
5.4. Publishing the data	51
5.5. Visualizing the interpolation results in CesiumJS	52
5.6. 3D city model	53
6. Conclusions, limitations and future work	55
6.1. Future work	57
A. Reproducibility self-assessment	59
A.1. Marks for each of the criteria	59
A.2. Self-reflection	59

List of Figures

2.1. UML model of the sensing entities	9
2.2. Overall Architecture with ETL Steps. Source: [24]	11
2.3. SmartApp. Source: [24]	12
2.4. APIs of SDI that integrate to outside frameworks. Source: [2]	12
2.5. Top-down development structure for open access. Source: [2]	13
2.6. Overall architecture. Source: [16]	14
3.1. Methodology overview	18
3.2. FROST-Server API response at http://localhost:8080/FROST-Server/	21
3.3. <code>scipy.interpolate.griddata</code> different methods results	25
3.4. <code>netCDF</code> or <code>xarray</code> structure	26
3.5. Shortened title for the list of figures	27
3.6. Shortened title for the list of figures	28
4.1. Implementation overview	32
4.2. Shortened title for the list of figures	33
4.3. Netatmo's <code>"/getpublicdata"</code> endpoint response schema	35
4.4. Real-time, raw data to SensorThings API methodology, part 1	36
4.5. Real-time, raw data to SensorThings API methodology, part 2	37
4.6. Custom API Process landing page	38
4.7. Example of the JSON body of the process request	38
4.8. Real-time, raw data methodology, part 3	39
4.9. Real-time, raw data methodology, part 4	39
4.10. NetCDF interpolation result on QGIS	43
4.11. Xarray provider in <code>pygeoapi</code>	44
4.12. OGC API Coverages landing page in <code>pygeoapi</code> server	45
4.13. Cesium platform. Source: Cesium, https://cesium.com/platform/	47
5.1. Real-time sensor locations in Rotterdam (September 15, 2023)	50
5.2. Raw historical sensor locations in Rotterdam (May 2023)	50
5.3. Eliminated historical sensors (white dots) and the accepted sensors after the cleaning process (blue dots) for May 2023	51
5.4. Interpolation results using SciPy for raw real-time measurements.	52
5.5. Interpolation results using SciPy.	52
5.6. Cesium visualization of interpolated data at different time stamps	53
5.7. Shortened title for the list of figures	53
5.8. Cesium visualization	54
5.9. Cesium visualization	54
A.1. Reproducibility criteria to be assessed.	59

List of Tables

2.1. OGC Web Services and OGC API corresponding instances	8
3.1. Netatmo Weather Station modules[40]	19
3.2. Root Mean Square Error (RMSE) for Kriging, IDW, NN [25]	24
3.3. Cesium ion accepted data formats and asset types [13]	30
A.1. Evaluation of reproducibility criteria	59

1. Introduction

Urban areas around the world are home for over 55 % of the global population [67], while being responsible for utilizing huge amounts of natural resources [29]. Due to the aforementioned facts, an effective system for city management that focuses on urban resilience and sustainability needs to be established and implemented. Urban resilience is important because it enables cities to effectively respond to shocks and stresses, contributes to long-term sustainability, fosters social cohesion, generates economic benefits, and promotes environmental sustainability. By building resilience, cities can enhance their capacity to withstand and thrive in the face of challenges, ensuring the well-being and prosperity of their residents.

A widely accepted definition given by Rockefeller Foundation describes resilience as “the capacity of individuals, communities, institutions, businesses, and systems within a city to survive, adapt, and grow no matter what kinds of chronic stresses and acute shocks they experience” [61]. Urban resilience has a crucial role for a variety of reasons by allowing cities to recover from a range of shocks and stresses, including natural disasters, economic downturns, and public health crises [61]. By building resilience, cities become better prepared to face these challenges head-on, therefore safeguarding their communities and infrastructures. Resilient cities are well-equipped to confront the multidimensional challenges posed by climate change, resource scarcity, and population growth [41]. They possess the capacity to adapt their infrastructure, policies, and systems to not only mitigate but also effectively manage these challenges. In doing so, they reduce their vulnerability and enhance their ability to thrive amidst uncertainty [31]. It is also crucial to recognize the intrinsic connection between urban resilience and environmental sustainability. Resilient cities prioritize sustainable practices, encompassing energy efficiency, green infrastructure, and waste management [41]. By embedding environmental considerations into their planning and development processes, cities reduce their ecological footprint and enhance their ability to adapt to environmental changes [31]. This symbiotic relationship ensures that urban resilience not only safeguards cities from immediate threats but also contributes to their long-term viability within the broader context of a changing planet.

In the context of an urban digital twin, the concept of urban resilience can be enhanced and supported. An urban digital twin is a virtual replica of a city that integrates data from various sources to provide a comprehensive understanding of the city’s functioning [17]. It serves as a collaboration and communication tool for urban planners, designers, and the general public, enabling more intentional decision-making for creating smart and sustainable cities [17]. By utilizing an urban digital twin, cities can enhance their resilience by gaining insights into the functioning of various systems and identifying vulnerabilities and potential risks [37], while it can also facilitate real-time monitoring and response to acute shocks, such as natural disasters or pandemics, by providing up-to-date information and enabling coordinated actions [29].

Digital technology and its recent advancements, such as Internet of Things (IoT) technologies and 3D city models can be applied in order to have a better visualization and monitoring

1. Introduction

of the city and the processes that take place locally. Cities and their physical components, such as buildings, road and railway networks, streets and highways or green areas can be represented digitally with high accuracy. This graphic representation of the urban environment is a 3D city model [58] and is mainly used for visualization. Additionally, it can be used for solar radiation estimation, energy demand applications, classification of building types, visibility and shadow cast analysis, kadaster and urban planning, to name a few [4]. On the other hand, the dynamic and up-to-date representation of the city's functions is implemented through the Internet of Things (IoT). It holds immense significance in collecting and transmitting data in urban digital twins through devices, such as sensors, that are deployed throughout the city to gather real-time data on various parameters, including air quality, traffic flow, and energy consumption [38].

The Urban Digital Twin industry is rapidly growing and has the potential to significantly improve how cities are managed and decisions are made on a large scale. However, there are some important challenges that need to be addressed for it to reach its full potential. Firstly, there is the need to ensure that the digital representation of a city can seamlessly connect with the real physical city [58]. This is important for effective city management and decision-making. Secondly, the lack of interoperability among different urban digital twin platforms and systems needs to be addressed in order to reach their full potential and effectiveness [57]. Interoperability refers to the ability of different systems, platforms, or technologies to exchange and use information seamlessly and is extremely important for integrating data from various sources while facilitating the exchange of information and knowledge [57].

1.1. Motivation

New technologies bring both opportunities and challenges. Urban digital twins are a new technology with the potential to improve the environmental performance of sustainable smart cities. However, there are significant challenges and open issues associated with urban digital twins that need to be addressed before they can be widely implemented [72].

Due to the fact that urban digital twins are composed of different but relevant components, interoperability or the ability of different systems to exchange data and work together. This is a major challenge, as current urban infrastructure and systems are often not interoperable [72]. This is particularly important for systems that use the Internet of Things (IoT) to improve environmental sustainability and climate change adaptation [3]. Moreover, a recent study [72] found that interoperability challenges are mentioned in 19% of the literature on urban digital twins, and while it is a well-known obstacle [6; 73], researchers [20] have identified the lack of interoperability as the most pressing unresolved issue regarding urban digital twins.

A crucial component of an IoT urban digital twin is the data acquisition, as it is the foundation of the urban digital twin technology and everything else depends on it [74]. This component, though, is often associated with challenges connected to sensor's network distribution and technical capabilities [72]. In many cases the sensor network is too sparse and heterogeneous and as a result the acquired sensor data show high temporal and spatial variability, especially in historical data [72]. In the same context of data limitations, data quality needs to be addressed, as the measurements need to be reliable enough in order to be used in the decision making process [28].

As technology advances, though, new forms of data acquisition techniques can provide alternatives by utilizing low-cost sensing equipment and citizen sensor networks [24]. Among the various crowd-sourcing instruments, citizen weather stations (CWS), also known as personal weather stations, have been gaining popularity. Citizen weather stations were initially assessed against official automatic weather station measurements and were subsequently employed to monitor urban temperatures in large cities [14; 36]. In fact, citizen weather stations enhance the potential for expanding geographical coverage of urban observations, reducing the reliance on limited official meteorological stations within cities, which often inadequately represent urban climate characteristics [9]. The growth of citizen weather stations networks not only helps us understand urban climate better but also has the potential to guide smart decisions about city planning, like parks, shade, ventilation, materials, buildings, and strategies to cope with heat [5]. For example, real-time monitoring of areas at a heightened risk of heat stress through citizen weather stations data [58] demonstrates the diverse applications offered by citizen weather stations. As a step further than the data acquisition, the visualization of the information provided is important in urban digital twins as it allows stakeholders to understand and analyze complex sensor data in a more intuitive and accessible manner [58].

The rapid development of urban areas, the increasing availability of environmental sensor data, and the growing need for resilient urban planning has created a demand for advanced tools that can integrate and visualize dynamic data. While 3D city models have the potential to fulfill this need, they currently lack the ability to represent and visualize the dynamic nature of the urban environment. This thesis aims to address this gap by proposing a platform that not only can visualize sensor data in a 3D urban model, providing an intuitive and informative representation of urban dynamics, but it is also developed according to internationally accepted standards that allow seamless interoperability of diverse geospatial components.

1.2. Research Questions

The research conducted in the current thesis is established on how environmental sensor data can be collected, processed and manipulated in order to produce information about the living environment. The main research question is formed as follows:

How can sensor data integration in a 3D city model facilitate comprehensive visualization of the city's dynamic elements, and what are the technical approaches to achieve this in a web-based environment?

The sub-questions are formed in a way that will answer the main research question and will provide a complete result. These questions are the following:

- How can international standards be employed to acquire, store, and manipulate real-time and historical sensor data, ensuring data consistency and interoperability?
- How can environmental sensor data be effectively processed to provide detailed spatio-temporal information for further analysis?
- How can environmental sensor data be made available and accessed in a standardized manner to enable effective utilization in geospatial applications?

1.3. Thesis outline

The thesis consists of 5 chapters. Chapter 1 introduces the study, highlighting the motivation for the research and presenting the key research questions that drive the investigation. In chapter 2, the theoretical background and related work delves into fundamental concepts, such as Urban Digital Twins and the integration of geospatial data and sensors, notably through Open Geospatial Consortium (OGC) standards. In chapter 3, the methodology provides insight into data collection and processing, with a particular emphasis on sensor data, its accessibility using standards like the SensorThings API, and the techniques for data interpolation. The subsequent chapter 4 details the implementation, including the tools, datasets, and the specific study area, shedding light on the real-time and historical sensor data pipelines and the transformation of 3D city models for visualization. Chapter 5 presents the results and discussion, future work and conclusions of the thesis.

2. Theoretical Background and Related work

2.1. Urban Digital Twins

According to [58], there are five digital twins components : a) data management, b) situational awareness, c) integration and collaboration, d) planning and prediction, e) visualization. This research will mainly focus on the data management aspect and on digital twins that integrate IoT technologies and sensors into their architecture.

Following, there are some indicative examples of urban digital twins that integrate the use of IoT technologies and open-source data, information models and global standards. These examples include the urban digital twins of the cities of Helsinki, Zurich and Dublin. The main scope of these is the development of future scenarios with more attention paid to disaster related scenarios and climate scenarios. Additionally, the scenarios could include solar radiation and ventilation or noise and atmospheric pollution [7]. These urban digital twins use a variety of technologies that provide the data for the 3D city model, that include Geographic Information Systems (GIS) models and more specific Building Information Modelling (BIM) models, laser scanners and Unmanned Aerial Vehicles (UAVs). At the same time, IoT technologies are also included, but this data is rarely used as input for the simulations [7].

Another well developed project is Virtual Singapore. This refers to a 3D urban digital twin of the city of Singapore, that provides a detailed visualization of the city's objects, including height of the buildings, surfaces' materials, but also real-time data streams. The platform is used for what-if scenarios, decision making, urban planning and potential energy production [43]. It uses sensor data to monitor and manage the city's water supply, traffic flow, and energy consumption, and to provide citizens with real-time information about the city's services and amenities. This is a case of a non open-source application.

"Digital Twin Victoria", is a project developed in Melbourne that has utilized all the available, diverse partnerships with various stakeholders to establish a complete database for the digital twin. The platform is active in matters such as disaster and asset management, scenario analysis and planning. A main aspect of the project is the use of open, real - time data that are visualized both in 2D and 3D web environments. With the development of the "Ararat Spotlight project", a significant effort is made to make full use of local data to be used in optimal regional management [18]. This digital twin is only a part of the large scale project Australia is leading since 2014, which includes the development of digital twins for three states, with collaboration of Cesium. A large advantage of these platforms is not only that they manage to gather each city's important information in one platform, but also the integration of real - time data regarding wind farm production, air quality and mobility, while enabling the use of simulations for historic or upcoming scenarios [12].

2. Theoretical Background and Related work

“Smart City Takamatsu” is an urban digital twin that provides solutions regarding tourism, well-being, traffic accidents and most importantly disaster management. In order to achieve this, there is extensive use real - time data of sensors regarding water and tide levels. By having the data collected and properly processed the city is not only saving resources on human power when disaster occurs, but also prediction and future scenarios can be tested-out for prevention [64].

2.2. Geospatial data and sensor integration

On account of the many components of the urban digital twins and the complexity of the systems, this section focuses on standardized ways of handling and integrating sensor data with geospatial data and web applications. First, an introductory subsection on the widely used international standards and their importance will be made and then a subsection where selected projects are presented, will follow.

2.2.1. Open Geospatial Consortium (OGC) standards

The importance of standards and interoperability in digital markets is very high [59]. According to McKinsey’s estimation, 40% of potential economic benefits of the Internet of Things (IoT) could be unrealized without interoperability [59]. The European Commission also emphasizes the role of standards in EU industrial policy, as they ensure smooth and reliable technology integration, provide economies of scale, foster research and innovation, and maintain open markets [59]. The Open Geospatial Consortium (OGC), a leading organization in the geospatial IT domain, has been developing data technology standards for over two decades [59]. They have produced over 60 standards and documents that have transformed the processing, discovery, documentation, and visualization of geospatial data [59]. The different OGC programs collaborate, exchanging requests, results, and engineering requirements with the geospatial IT community [59].

The first generation of geospatial standards, often referred to as “OGC Web Services”, were developed by the Open Geospatial Consortium (OGC) and played a crucial role in defining how geospatial data and services were shared and accessed over the internet [Standards - OGC]. These standards were created during the late 1990s and early 2000s [Standards - OGC]. Following, a brief overview of the most commonly used standards is given:

- **Web Map Service (WMS):** WMS is one of the earliest OGC standards, first published in 1999. It allows clients to request and receive static map images from a WMS server. These images can be in formats like PNG, JPEG, or GIF. WMS is widely used for visualizing geospatial data in web-based mapping applications [Web Map Service - OGC].
- **Web Feature Service (WFS):** WFS was introduced in 2002. It provides a standard way to request and retrieve geographic feature data in vector format (e.g., GML) from a WFS server. WFS enables the exchange of complex geographic features, such as points, lines, and polygons, between systems [Web Feature Service - OGC].

- **Web Coverage Service (WCS):** WCS, first published in 2004, allows clients to retrieve geospatial coverage data, such as satellite imagery or scientific data, in a standard format. WCS is used in applications that require access to multi-dimensional, gridded data [[Web Coverage Service - OGC](#)].
- **Web Processing Service (WPS):** WPS is an OGC standard that was first introduced in 2007. It provides a standard interface for defining and executing geospatial processes or algorithms over the web. These processes can include tasks like geoprocessing, spatial analysis, and data transformation [[Web Processing Service - OGC](#)].
- **Catalog Services for the Web (CSW):** CSW was developed to enable the discovery and retrieval of geospatial metadata. It allows users to search for and access information about available geospatial datasets, services, and resources. The CSW standard emerged in the early 2000s [[Catalogue Service - OGC](#)].
- **Styled Layer Descriptor (SLD):** SLD is a specification for defining how to style and symbolize geospatial data for visualization. It was introduced in the early 2000s and is used in conjunction with WMS to control the appearance of map layers [[Styled Layer Descriptor - OGC](#)].

In recent years, there has been a widespread adoption of Web Application Programming Interface (API) technologies in various organizations that share data on the internet. As of November 22, 2022, Swaggerhub.com, a well-known API registry, listed over 500,000 registered APIs [26]. Recognizing the growing popularity of Web APIs, the Open Geospatial Consortium (OGC) initiated a focused effort in 2018 to create a set of standards that make use of modern web approaches. These standards, collectively referred to as the OGC API Standards, allow developers to consistently implement geospatial capabilities in web APIs that deal with location data and maps [26]. The new standards are developed according to the OpenAPI specifications (OAS), that has a main goal to establish a standardized, programming language-neutral interface description for HTTP APIs [30]. This enables both users and computers to explore and comprehend a service's capabilities without the need for access to source code, extra documentation, or the examination of network data. When correctly defined using OpenAPI, a user can engage with the remote service with minimal implementation complexity. Much like interface descriptions have simplified interactions in lower-level programming, the OpenAPI Specification eliminates uncertainty when making calls to a service. [30]

Following, there is a summary of some key second-generation OGC API standards:

- **OGC API - Common:** This standard provides a common framework and conventions used by other OGC APIs. It ensures consistency and interoperability across various OGC API implementations [[OGC API Common](#)].
- **OGC API - Features:** This standard is designed for querying and retrieving geographic feature data. It replaces the older Web Feature Service (WFS) standard. OGC API Features, exposes geospatial features as resources accessible via simple URLs. It uses modern protocols like HTTP and JSON for data exchange, making it more user-friendly and efficient [[OGC API Features](#)].
- **OGC API - Coverages :** This is a standard designed for accessing and manipulating coverage data. Coverage data can represent information such as satellite imagery, gridded data, or sensor observations. OGC API - Coverages provides a consistent

2. Theoretical Background and Related work

OGC Web Services	OGC API	Functionality
Web Map Service (WMS)	OGC API Maps	Delivering geospatial data as web maps with spatial references in a dynamic manner
Web Map Tile Service (WMTS)	OGC API Tiles	Delivering and accessing geospatial data as tiles
Web Feature Service (WFS)	OGC API Features	Delivering and accessing geospatial data as vector data
Web Coverage Service (WCS)	OGC API Coverages	Delivering and accessing geospatial data as raster data
Web Processing Service (WPS)	OGC API Processes	Enables the processing of geospatial information on the web

Table 2.1.: OGC Web Services and OGC API corresponding instances

and modern way to request, retrieve, and interact with coverage data over the web [[OGC API Coverages](#)].

- **OGC API - Tiles:** This standard is focused on retrieving and managing pre-rendered map tiles, similar to the Tile Map Service. It simplifies the process of working with map tiles in web mapping applications, providing resource-based access [[OGC API Tiles](#)].
- **OGC API - Maps:** This standard is designed for serving maps over the web. It introduces a more modern approach to map rendering and interaction. Like other OGC APIs, it uses HTTP and JSON for data access [[OGC API Maps](#)].
- **OGC API - Processes:** This standard enables the execution of geospatial processes and workflows, providing dynamic, interactive capabilities. It is designed for performing geospatial operations in a flexible and resource-based manner [[OGC API Processes](#)].

SensorThings API

SensorThings API is a standard developed by OGC and is essentially a protocol that allows the different IoT devices and applications to connect through the web. It is designed to integrate the sensor data into various applications and platforms [55], while using a standardized way to access the sensor data on a global scale [34]. The SensorThings API is especially useful for managing and accessing sensor data in real-time [55] and can be used to integrate this sensor data with 3D city models [55]. This integration allows for a more extensive understanding of the urban environment and assists in the development of smart city applications [55]. It is flexible, scalable and supports a wide range of IoT technologies [34].

As it can be seen in Figure 2.1, SensorThings API has two main parts, the Sensing part that produces a REST API for managing heterogeneous data through HTTP POST, GET, PATCH,

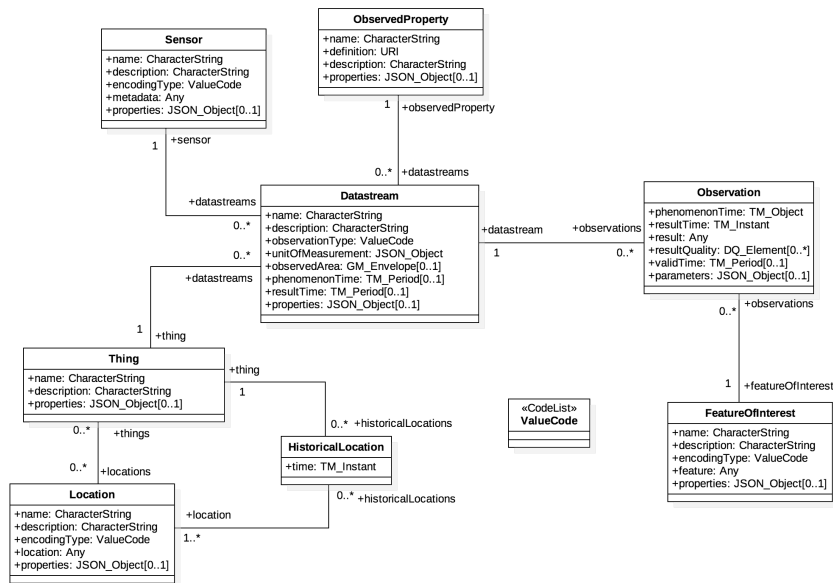


Figure 2.1.: UML model of the sensing entities

and DELETE operations, and the other part is Tasking [34]. REST is a widely used architectural style for building web services [65]. It provides a uniform API for accessing web sensors and other smart things [65]. The API resolves the complexities of sensor communication protocols and makes it easier to develop applications that interact with sensors.

The key entities of the SensorThings API are the following;

Things: A “Thing” represents a physical or virtual object with sensors that collects data. Each Thing can have one or more sensors associated with it, for example temperature or wind sensors [34].

Locations: They represent the location of the Thing they are associated with. In the case of static sensors one Thing is associated with one Location, in other cases the Location represents the last known location of the Thing [34].

HistoricalLocations: Represent the historical Locations of a Thing [34].

Sensor: Sensors are devices that collect the data measurements, such as temperature, humidity, pressure, or any other data [34].

ObservedProperties: They define the phenomenon that is being observed. It can be for example temperature or humidity [34].

Observations: They represent the actual measurements and values for a specific time that are collected by a single Sensor [34].

Datastreams: A Datastream is the grouping of many observations from a specific sensor for a period of time and for a specific ObservedProperty [34].

2. Theoretical Background and Related work

FeatureOfInterest: “An Observation results is a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the FeatureOfInterest of the Observation” [34].

The Sensor Web With Spatial Data Infrastructures (SENSID) research project [2] aims to expand the capabilities of Spatial Data Infrastructures (SDIs) by integrating sensor web technologies, which involves combining geospatial and built infrastructure data and using specific sensor data to create a general sensor web framework. The main challenges involve making SDIs and Sensor Web interfaces semantically compatible. The research plan involves identifying sensor data sources, establishing an open-source SDI, aligning APIs and functions between Sensor Web and SDI, and conducting case studies like those related to hazards and urban applications. The ultimate goal is to create a “Geospatial Web” or “Geosemantic Web” by connecting various OGC standard interfaces of SDI and Sensor Web interfaces.

2.2.2. Smart Emission initiative

The research paper by [24] discusses the Smart Emission initiative, which is a citizen sensor network using low-cost sensors to collect data about environmental quality. The spatial data infrastructure (SDI) used is open and accessible on the internet and follows open geospatial standards. The Smart Emission initiative aims to establish a citizen sensor network in a real-life urban lab setting and involves citizens in data collection and analysis in a low-cost and efficient manner. The Smart Emission initiative has a broad environmental perspective and aims to explore how low-cost sensors can complement high-end sensing methods. Currently, the initiative has been implemented in the city of Nijmegen in the Netherlands and the study focuses on this test-case.

The data collection involves the use of low-cost sensors called “Jose” sensor units located at citizens’ homes, which collect various environmental indicators such as air quality, noise load, humidity, light intensity, carbon monoxide, temperature, and noise load. The Jose sensor unit is connected to a power supply and the internet, allowing for data collection and transmission. The data collected by the Jose sensor units are encrypted and sent to the data production platform hosted by CityGIS every 15 seconds. A dedicated ‘Jose Input Service’ decrypts the data and stores it in a MongoDB database. This database serves as the source production database, where all raw sensor data streams are permanently stored.

The Smart Emission initiative utilizes the FIWARE platform and FIWARE Lab to develop smart applications in various sectors. The distribution database stores Smart Emission data by harvesting and pre-processing the raw sensor data from the CityGIS production platform. A harvesting mechanism collects data every minute using the raw sensor API, and a multi-step Extract-Transform-Load (ETL)-based pre-processing mechanism transforms the JSON-encoded data into the Postgres/PostGIS database. In Figure 2.2, the whole architecture with the ETL steps is described, where the arrows represent the flow of data, the circles depict harvesting/ETL processes, the server instances are in rectangles and the data stores are represented by the DB icons. The pre-processing focuses on calibrating the raw data from air quality sensors to improve interpretability and accuracy. The calibration procedure compares Smart Emission sensor data to high-cost air quality sensor installations operated by the National Institute for Public Health and the Environment in Nijmegen. Post-processing involves transforming the pre-processed values using statistics and spatial interpolations. The paper highlights the importance of open access principles in citizen sensor networks, allowing for the sharing and utilization of the collected data. Therefore, the architecture

leverages the OGC geospatial standards, such as WMS, WFS, SOS, and the SensorThings API, to expose sensor data to the Internet for re-use by different developer communities.

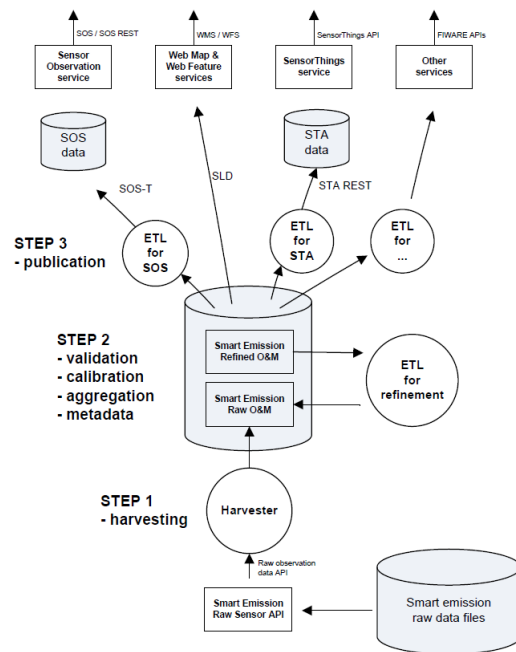


Figure 2.2.: Overall Architecture with ETL Steps. Source: [24]

The data infrastructure of the Smart Emission initiative is based on an open access approach, allowing for re-use and access to data through standardized APIs and client applications. The infrastructure prioritizes the privacy of Smart Emission citizens, ensuring that data remains open while protecting personal information. The software infrastructure largely consists of open source software and documentation. Open access enables easy access to sensor data through various client applications, such as GIS applications, statistical packages, and web applications. The SmartApp, developed during the project, uses the 52°North Sensor Web REST API to provide users with a simple map application displaying the measurements, as it can be seen in Figure 2.3. The system undergoes continuous improvements and innovations based on feedback from citizens and meetings with the consortium. A website dedicated to users has been created, offering data viewers, a forum, and documents exchanged at citizen meetings. Regular meetings with citizens and consortium members allow for insights and lessons to be shared, which inform optimizations to the technical architecture and data processing mechanisms to improve the Spatial Data Infrastructure (SDI). Overall, the initiative strives to create an open and accessible platform for citizen sensor networks while addressing the challenges and constraints faced during implementation.

2.2.3. Integrating Sensor Web With Spatial Data Infrastructures

Another interesting study that explores the integration of sensor web with spatial data infrastructures, is the one by [2]. According to the paper, almost every decision made by individuals or organizations has a geospatial component and the benefits of the technical

2. Theoretical Background and Related work

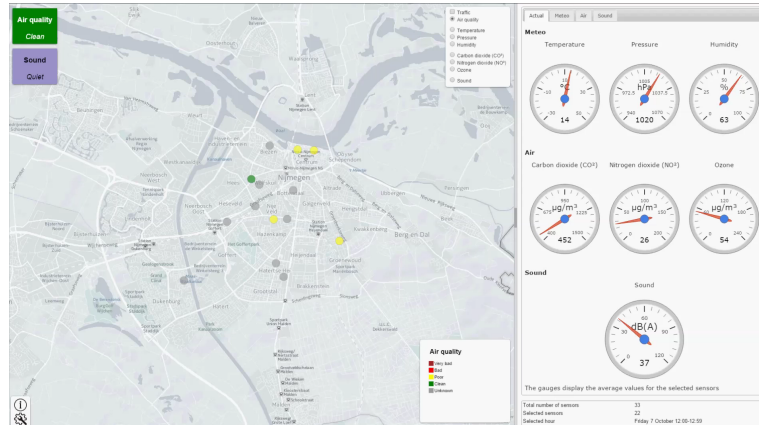


Figure 2.3.: SmartApp. Source: [24]

capability to track, measure, and analyze location-based information, are major. The main goals of this research paper are to extend Spatial Data Infrastructures (SDIs) with sensor web enablement and to converge geospatial and built infrastructure. The paper aims to address several problems, including identifying readily usable sensor data sources and setting up an open source SDI. It also discusses the importance of matching the APIs and functions between the Sensor Web and SDI, and highlights the significance of case studies in hazard applications and urban applications.

The example that is discussed is the called SANY (Sensor Systems Anywhere) and explores the interoperability between in-situ sensors and sensor networks. The methodology proposed in the paper involves understanding the concepts of the Sensor Web, such as the Sensor Observation Service (SOS) for retrieving sensor observation data, the Sensor Planning Service (SPS) for planning and executing tasks, the Sensor Alert Service (SAS) for subscribing to specific alert types, and the Web Notification Service (WNS) for facilitating message interchange. For the SDI components, OGC and ISO international standards are utilized in order to serve geospatial data in all available formats, through different APIs (Figure 2.4).

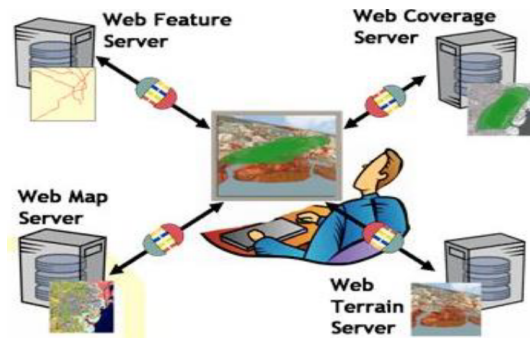


Figure 2.4.: APIs of SDI that integrate to outside frameworks. Source: [2]

To establish interfaces between the Sensor Web and SDI, the paper suggests using GeoServer and GeoNode, which provide an OGC-compatible data store that can communicate through common formats like GML, GeoJSON, KML, and GeoTiff. These components can connect to

different spatial backends, including PostGIS, Oracle Spatial, and ArcSDE. The paper also mentions the use of GeoNetwork, which provides a standard catalog and search interface based on OGC standards, accessed through the CSW interface within GeoNode. The main map interface for GeoNode is the Map Composer/Editor, which communicates with other components via HTTP, JSON, and standard OGC services. The components that are utilized and their connections are described in Figure (2.5).

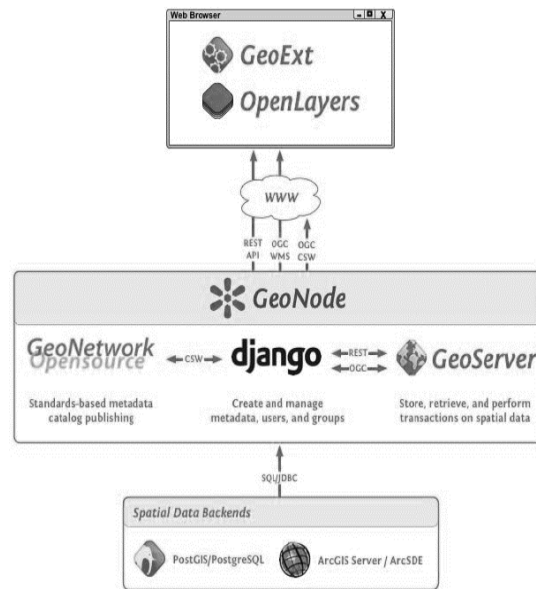


Figure 2.5.: Top-down development structure for open access. Source: [2]

2.2.4. OGC 3D-IoT Platform for Smart Cities

An OGC Pilot [16] was conducted to integrate city models, both indoor and outdoor, with sensor data to define and visualize observable characteristics for various components of buildings and larger city elements. These dynamic sensor observations primarily focused on measuring Particulate Matter (PM), which encompasses solid particles and liquid droplets in the air. The observations covered PM_{2.5} air quality (fine particulate matter with particles generally smaller than 2.5 micrometers) and room occupancy within buildings. PM_{2.5} air quality was measured and synthesized for outdoor locations, while building room occupancy data was synthesized for individual rooms. The detailed architecture of the project is presented in Figure 2.6.

The observations were made accessible through services that implemented the OGC SensorThings API standard. The features, represented using the OGC IndoorGML building model and 3D-Tiles/glTF city model features, were provided through implementations of the OGC API – Features – Part 1: Core standard. IndoorGML, a profile of the OGC Geography Markup Language (GML) standard, was used for this purpose. Multiple types of clients deployed in the pilot, including standalone dashboard applications, applications relying on 3D Portrayal Service (3DPS), and Augmented Reality (AR) visualization tools. Web Processing Service (WPS) and OGC API - Processes implementations were also deployed to

2. Theoretical Background and Related work

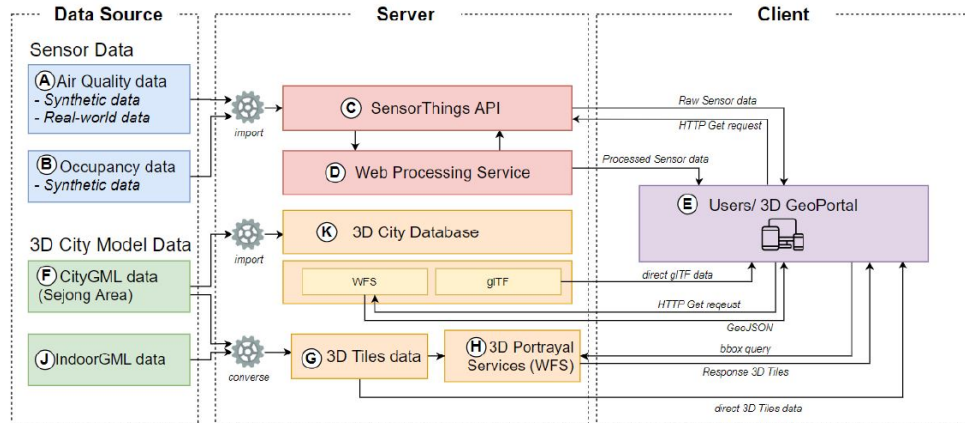


Figure 2.6.: Overall architecture. Source: [16]

combine and interpolate observations from various locations to estimate observable properties of larger city elements, such as the interiors of building floors and the air quality above city streets or blocks. These derived observations could be directly visualized in client applications or added to SensorThings API implementations as new data.

The pilot demonstrated a practical architecture based on standards for connecting real-time sensor data with modeled city features. It emphasized the importance of aligning identifiers for features and sensors, along with other infrastructure data, to ensure the sustainability of robust Smart City 3D-IoT capabilities. Despite the pilot's valuable insight on a system that integrates sensor data and OGC standards, certain aspects of the project can be explored further. Firstly, the pilot uses synthesised observations for the air quality. In the context of the outdoor urban space, is interesting to explore the possibilities and limitations of real sensor data as well as the availability of open access data. Moreover, the pilot only handles real-time sensor data and completely ignores the historical data. In the context of using only synthesised, real-time observations, in the pilot, there is no cleaning process to ensure the high quality of the data. Therefore, in the current thesis the aforementioned limitations will be explored.

2.3. Environmental Sensor Data

Environmental sensor data refers to information or measurements collected by sensors specifically designed to monitor and record various environmental parameters or conditions. These sensors are deployed in natural or built environments to capture data related to factors such as weather, air quality, water quality, soil conditions, and other aspects of the surroundings.

2.3.1. Crowd-sourced data for urban studies

Environmental sensor data are widely used in the field of urban studies. An example of this kind of studies is the Urban heat island (UHI) effect that have a significant impacts in many

socio-economic aspects. Traditional methods for studying UHI often fall short in providing the necessary detailed observations to discern small temperature differences within urban areas. Crowd-sourcing presents a solution to this challenge. In the study conducted by [1] in London in 2018, crowd-sourced data from over 1300 Netatmo personal weather stations were utilized to estimate building cooling and heating loads. The local climate zone (LCZ) scheme was employed to categorize London's diverse urban environments, and the temperature variations between different LCZs generally aligned with their respective temperature definitions.

Crowd-source data allows for a dense network of personal weather stations that can offer insight into the spatial patterns of urban temperature [14]. Additionally to the high spatial resolution of the stations, the data has a high temporal resolution as it can be provided in real time or near-real time [35] and it can be acquired instantly in order to be used in research. This factor also allows for long term environmental monitoring that can provide valuable insight into spatio-temporal variations of a phenomenon. Another important aspect that is making crowd-sourced data so appealing is its low-cost, not only of the sensors themselves but also for the data collection, that makes it more appealing to researchers and projects with narrow means, as personal devices or online tools are used instead of expensive equipment [42].

Despite the fact that crowd-sourced data bring forth major advantages in comparison to more traditional methods of acquiring environmental data, there are also important issues that arise and need to be taken into consideration. As different sensors can cause diverse issues, this section is only focusing on presenting the identified problems of the Netatmo sensors. Firstly, the metadata of the sensors can be wrongly specified in cases where the user chooses not to manually add the coordinates of the sensor and instead they are assigned in an automated way based on the IP address of the wireless network [36]. Additionally, the APIs that use real time data collection, have the limitation of not collecting and therefore missing measurements of sensors that are momentarily not connected to the Wi-Fi or have no more battery power [36]. Lastly, the placement of the sensors is not always optimal, besides the manufacturer's guidelines to locate the outdoor sensors away from the sun, the users in many cases dismiss this principle [36]. This practice can lead to extreme radiative errors for specific sensors and for specific times of the day, which can result in generalized errors when the data from all sensors are processed collectively [39]. In this context, data preprocessing is an important step to address the aforementioned problems and to provide usable and accurate data for research.

3. Methodology

This section provides more insight into the methodology and the steps that were followed in order to acquire environmental sensor data and to process it, using SensorThings API and then visualize it in a 3D WebGIS environment, using CesiumJS. The study case area for this thesis is the city of Rotterdam, the Netherlands and only temperature data will be retrieved and be processed to provide a complete pipeline that can be duplicated for other environmental data in the future.

3.1. Methodology Overview

The methodology graph presented in Figure 3.1 offers a visual representation of the sequential steps involved in the research, spanning from the initial stages of data collection to subsequent phases of analysis, publication, and visualization via a client interface. The research starts by identifying and sourcing the data, focusing on two primary categories: outdoor temperature sensor data and 3D city model data. Distinct approaches were taken for real-time and historical sensor data due to their unique characteristics. Historical data underwent an additional data cleaning step, enhancing data quality for potential, future research. Subsequently, a data interpolation process was applied to both real-time and historical data, harnessing the benefits of a densely distributed network of crowd-sourced sensors. This interpolation process aims to create a detailed and visually useful output, meeting the requirements of decision-makers. Following the data interpolation process, the resulting data is published using the Open Geospatial Consortium (OGC) international standards, enabling standardized access by a wide range of users for diverse applications. In the concluding phase of this thesis, the sensor data is integrated with a 3D city model within a web client. This platform is designed to provide users with a cohesive and interactive experience that combines sensor data and city models, fostering effective data visualization and analysis.

3.2. Sensor data

In the Netherlands the temperature is officially measured by 34 automatic weather stations (on land) that are operated by the Royal Netherlands Meteorological Institute (Koninklijk Nederlands Meteorologisch Instituut - KNMI) and provide measurements every 10 minutes [15]. Even though the placement of these weather stations is in alignment with the World Meteorological Organization (WMO), for Rotterdam there is only one available station [15], that cannot provide detailed temperature data, which are needed for urban studies.

An alternative and highly beneficial approach to the KNMI stations is that environmental data can also be provided by the residents of a city, in the form of open-source, crowd-sourced data. For this study, Netatmo sensors from the already established network were

3. Methodology

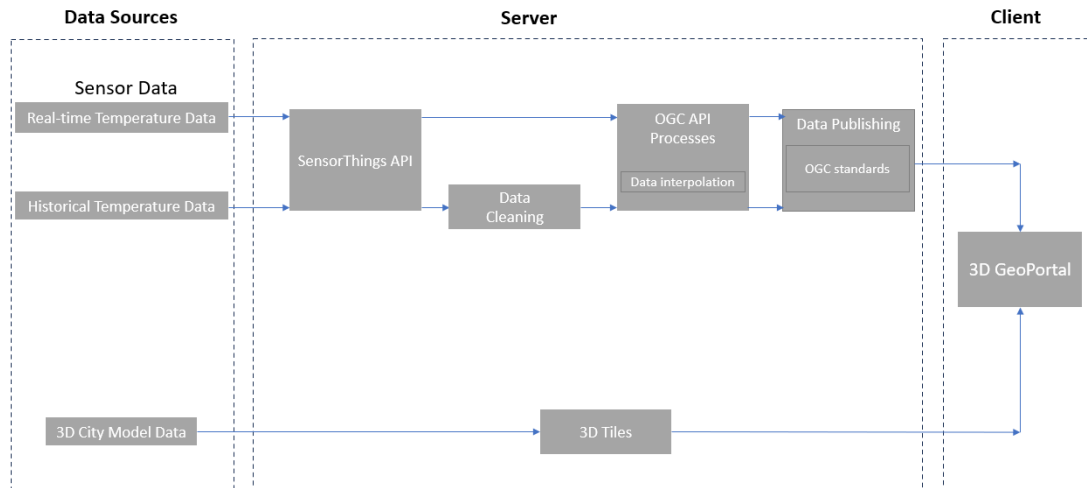


Figure 3.1.: Methodology overview

used in order to retrieve temperature data. Netatmo operates as a commercial manufacturer and data aggregator specializing in citizen weather stations. They provide affordable weather stations to individuals worldwide, facilitating the monitoring of both outdoor and indoor weather conditions, such as temperature and humidity. Netatmo stations utilize Wi-Fi connectivity for seamless data transmission and automated uploads to a dedicated server. Owners can access real-time data visualization through the accompanying application software. Furthermore, these observations are made publicly available via a dedicated Application Programming Interface (API), allowing users to freely access data within the limits set by the provider [52].

Netatmo's products include weather stations, indoor climate monitors, air quality sensors, security cameras, and more, all designed to help users better understand and manage their living spaces and outdoor surroundings [40]. The Netatmo stations comprise both indoor and outdoor modules. The outdoor module is responsible for collecting real-time weather data, which includes parameters like temperature, humidity, and barometric pressure. This data is made available on the Netatmo Weathermap web portal if the user consents to sharing. Netatmo provides three APIs that can be used to collect the measured data: `"/getpublicdata"`, `"/getstationdata"`, `"/getmeasure"` [40].

The outdoor temperature and humidity sensors exhibit a high level of precision, with an accuracy of $\pm 0.3^{\circ}\text{C}$ over a temperature range spanning from -40°C to 65°C , and an accuracy of 3%, respectively. The accuracy of the temperature measurements has been verified and validated in a study conducted by [36] within a temperature range of 0°C to 30°C . However, it's worth noting that the placement conditions of the outdoor module can significantly affect temperature readings [51].

3.2.1. Real time sensor data

The measurements that are available to the public can be accessed and retrieved through the `"/getpublicdata"` API that returns publicly shared data about the devices, such as device id, mac address and location, along with temperature, pressure, humidity, rain and wind

Product	Product Types	Data Available
Smart Home Weather Station Smart Outdoor Module	NAMain NAModule1	Device information (FW version, Radio/WiFi status & battery level)
Smart Anemometer Module	NAModule2	Sensors data (temperature, pressure, CO2, noise, rain, wind, ...)
Smart Rain Gauge Module Smart Indoor Module	NAModule3 NAModule4	

Table 3.1.: Netatmo Weather Station modules[40]

measurements, for a user defined area [40]. The data is supplied near real-time and can be acquired through the API every ten minutes, therefore there is no access to historical data through this specific endpoint. the `"/getpublicdata"` endpoint provides the latest measurement of all the sensors that are active the time the request is made, within the user-defined area.

3.2.2. Historical sensor data

In order to access the historical data provided by Netatmo, the `"/getmeasure"` endpoint. This can be used only for a specific device id and for a specific time-frame. Therefore, the `"/getmeasure"` endpoint retrieves multiple measurements of one sensor device for a period of time for a certain time interval (30 minutes, 1 hour, 3 hours, 1 day, 1 week, 1 month) between two measurements [40] and for a specific data type. The data types that can be returned from the endpoint are the following:

- Temperature data (°C) = temperature, min-temp, max-temp, date-min-temp, date-max-temp
- Humidity data (%) = humidity, min-hum, max-hum, date-min-hum, date-max-hum
- CO2 data (ppm) = co2, min-co2, max-co2, date-min-co2, date-max-co2
- Pressure data (bar) = pressure, min-pressure, max-pressure, date-min-pressure, date-max-pressure
- Noise data (db) = noise, min-noise, max-noise, date-min-noise, date-max-noise
- Rain data (mm) = rain, min-rain, max-rain, sum-rain, date-min-rain, date-max-rain
- Wind data (km/h, °) = windstrength, windangle, guststrength, gustangle, date-min-gust, date-max-gust

3.3. SensorThings API standard for data accessing

Following an implementation that is based on the OGC standards, the most suitable way to process and store the data from the Netatmo, is SensorThings API. This standardization ensures interoperability and consistency across diverse systems, reducing the need for custom integrations. This standardization reduces the complexity of integrating various sensor devices while minimizing the dependence on custom solutions. The API follows RESTful principles and leverages HTTP methods, simplifying application development and allowing developers to apply familiar web development tools and techniques. Its advanced query capabilities allow users to filter and aggregate sensor data based on specific criteria, including time, location, and sensor characteristics. With support for both real-time and historical data, the API can be used in a broad spectrum of applications, and scenarios that require immediate monitoring as well as in-depth historical trend analysis. Importantly, the SensorThings API can interoperate with other OGC standards and specifications creating a complete solution for data management and its flexibility allows it to be adjusted for diverse sensor types, situations and purposes, making it a useful tool for many different IoT applications that potentially require the harvested sensor data.

The data from these sensors will be gathered and managed in the FROST-Server (FRaunhofer Opensource SensorThings) server, which is an open - source server that was developed according to the implemented SensorThings API standard. It is characterized by high performance and has also been used in numerous use cases, such as the smart city of Hamburg and the European "beAWARE" project [21]. FROST-Server also supports a PostgreSQL database with the PostGIS extension in order to store the data. FROST-Server, as a RESTful API supports not only the basic "GET" request but also the "PATCH" and "DELETE" requests to update and delete its contents.

Once the server is deployed and running (e.g. on localhost:8080), the FROST API has the format described in Figure 3.2, where each SensorThings API entity can be accessed by adding the relevant entity keyword at the end of the main FROST API.

3.4. Historical sensor data pre-processing

As it was thoroughly explained in section 2.2.1, while using crowd-sourced sensor data many issues can occur. A complete solution to address the aforementioned problems was developed through the "CrowdQC+", R package that incorporates a five step data correction process. The advantage of the tool is that it does not need to use external data, for example from other weather stations, as reference and it can fully operate with only as input the crowd-sourced data [19]. On the downside, the tool needs hourly data between 7 days and a month to be able to perform adequately. In the study that was conducted [19] in order to evaluate the tool, temperature data from Netatmo sensors were used for the time period of one year and for the cities of Amsterdam, Netherlands, and Toulouse, France.

The tool's main quality control levels are the following:

1. Metadata check: Eliminates sensors with identical latitude and longitude values. This error occurs when geographic coordinates are automatically assigned based on the user's IP address.

3.4. Historical sensor data pre-processing

```

{
  "value": [
    {
      "name": "Datastreams",
      "url": "http://localhost:8080/FROST-Server/v1.1/Datastreams"
    },
    {
      "name": "FeaturesOfInterest",
      "url": "http://localhost:8080/FROST-Server/v1.1/FeaturesOfInterest"
    },
    {
      "name": "HistoricalLocations",
      "url": "http://localhost:8080/FROST-Server/v1.1/HistoricalLocations"
    },
    {
      "name": "Locations",
      "url": "http://localhost:8080/FROST-Server/v1.1/Locations"
    },
    {
      "name": "Observations",
      "url": "http://localhost:8080/FROST-Server/v1.1/Observations"
    },
    {
      "name": "ObservedProperties",
      "url": "http://localhost:8080/FROST-Server/v1.1/ObservedProperties"
    },
    {
      "name": "Sensors",
      "url": "http://localhost:8080/FROST-Server/v1.1/Sensors"
    },
    {
      "name": "Things",
      "url": "http://localhost:8080/FROST-Server/v1.1/Things"
    },
    {
      "name": "MultiDatastreams",
      "url": "http://localhost:8080/FROST-Server/v1.1/MultiDatastreams"
    }
  ],
  "serverSettings": {
    "conformance": [
      "http://www.opengis.net/spec/iot_sensing/1.1/req/batch-request/batch-request",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/create-observations-via-mqtt/observations-creation",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/create-update-delete",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/data-array/data-array",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/datamodel",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/multi-datastream",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/receive-updates-via-mqtt/receive-updates",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/request-data",
      "http://www.opengis.net/spec/iot_sensing/1.1/req/resource-path/resource-path-to-entities",
      "https://fraunhoferiosb.github.io/FROST-Server/extensions/DeepSelect.html",
      "https://fraunhoferiosb.github.io/FROST-Server/extensions/GeoJSON-ResultFormat.html",
      "https://fraunhoferiosb.github.io/FROST-Server/extensions/JsonBatchRequest.html",
      "https://fraunhoferiosb.github.io/FROST-Server/extensions/ResponseMetadata.html",
      "https://fraunhoferiosb.github.io/FROST-Server/extensions/SelectDistinct.html",
      "https://github.com/INSIDE-information-systems/SensorThingsAPI/blob/master/CSV-ResultFormat/CSV-ResultFormat.html"
    ],
    "http://www.opengis.net/spec/iot_sensing/1.1/req/create-observations-via-mqtt/observations-creation": {
      "endpoints": [
        "mqtt://localhost:1883"
      ]
    },
    "http://www.opengis.net/spec/iot_sensing/1.1/req/receive-updates-via-mqtt/receive-updates": {
      "endpoints": [
        "mqtt://localhost:1883"
      ]
    }
  }
}

```

Figure 3.2.: FROST-Server API response at <http://localhost:8080/FROST-Server/>

3. Methodology

2. Distribution check: Removes statistical outliers at both the lower and upper ends of the distribution. This targets extreme values caused by radiative errors and errors in indoor placement that result in abnormally low temperatures. Elevation corrections are performed in the CrowdQC+ process, but the impact is minimal in regions with flat topography. In this step, the outliers are detected through a modified z-score approach that is followed by masking the suspicious data [23]. For the calculation of the z-score the more efficient Qn estimator is used instead of the median absolute deviation (MAD) as is defined by [54] as:

$$Q_n = \text{median}(|x_i - x_j|; i < j) \quad (3.1)$$

where:

median is the median function,
 x_i and y_i are data points.

The elevation corrections are calculated by [23] as:

$$T'_{M1} = T_{M1} + 0.0065(z - \text{mean}(z)) \quad (3.2)$$

where:

0.0065 corresponds to the usual standard atmospheric lapse rate.
 z corresponds to the elevation of the stations.
 T_{M1} corresponds to the filtered data from level 1.

The z-score Z is calculated as:

$$Z = \frac{T'_{m1} - \text{median}(T_{m1'})}{Q_n(T_{m1'})} \quad (3.3)$$

Values that lead to the null hypothesis being rejected with a significance level of 0.05 in the upper tail and 0.01 in the lower tail of the distribution are regarded as problematic and are thus concealed. This can be formally expressed as follows:

$$T_{M2}[i, j] = \begin{cases} T_{M1}[i, j], & \text{if } -2.32 < Z < 1.64 \\ NaN, & \text{otherwise} \end{cases} \quad (3.4)$$

where: i and j correspond to the time step and each of the stations, respectively.

3. Data validity: Checks each station to determine the frequency of flagged values from step 2 over a specific period of time. If the false flags are above a user defined threshold, the station is considered erroneous and will be excluded.

4. Temporal correlation: Assess the temporal correlation, by utilizing the Pearson correlation coefficient (R), between each station and the median of all stations for a specific time period. The Pearson correlated coefficient can be described as [66]:

$$\mathbf{R} = \begin{bmatrix} 1 & r_{12} & r_{13} & \dots & r_{1n} \\ r_{21} & 1 & r_{23} & \dots & r_{2n} \\ r_{31} & r_{32} & 1 & \dots & r_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & r_{n3} & \dots & 1 \end{bmatrix} \quad (3.5)$$

where:

R is the correlation matrix.

r_{ij} is the correlation between variables i and j .

n is the number of variables.

For this step in the cleaning processes the following is applied:

$$T_{M2}[i, j] = \begin{cases} T_{M1}[i, j] & \text{if } -2.32 < Z < 1.64 \\ \text{NaN} & \text{otherwise} \end{cases} \quad (3.6)$$

5. Spatial Buddy Check: Detects outliers within the vicinity of each station. This step focuses on identifying single, unrealistically high values caused by radiative errors.

For this step the z-score is calculated as:

$$Z = \left| \frac{ta_{i,j} - \text{median}(ta_{i,\text{buddies}})}{Q_n(ta_{i,\text{buddies}})} \right| \quad (3.7)$$

where:

$ta_{i,j}$ is the t_a value at time i and station j ,

$ta_{i,\text{buddies}}$ are the t_a values of the buddies at time i .

For each of the quality control levels, specific values of correlation coefficient can be set by the user, that will identify the flagged data, along with other arguments such as the time frame for which the user wants to perform the quality control or the minimum number of neighbouring stations that are within a certain radius of each station to perform the "spatial buddy check".

In conclusion, CrowdQC+ is a valuable tool for quality control of crowd-sourced air-temperature observations. By ensuring the accuracy and reliability of the data, CrowdQC+ contributes to the advancement of urban climate research and supports evidence-based decision-making for urban planning and design.

3.5. Sensor data interpolation

After cleaning, the raw historical sensor data needs to undergo a process to be transform from a point dataset to a raster dataset that will give a better visualization of the temperature patterns across the area. This process is called interpolation and it involves estimating values at unobserved locations within a raster dataset based on the values of surrounding observed locations.

Interpolation, widely explored in fields like AI, data mining, statistics, and signal processing, often overlooks the simultaneous integration of spatial and temporal data [25]. Additionally, these methods frequently fail to address the demands of real-time streaming scenarios, where data is continuously generated and processed [25]. The study carried out by [25], introduces an advanced Kriging interpolation algorithm designed to address the aforementioned research challenges. In this study, a comparison is made between the interpolated data obtained using the developed Kriging algorithm (TreCK), Inverse Distance Weighted (IDW), and Nearest Neighbor (NN). The research results, as evaluated using the root mean square error, indicate that Kriging performs slightly better than the other two methods, particularly in scenarios with a substantial percentage of randomly switched-off sensors (see Table 3.2). Given the marginal improvement in accuracy and the increased complexity associated with the Kriging algorithm, this thesis opts for the simpler Nearest Neighbor approach.

% Switched-off Sensors	TreCK	IDW	NN
0%	1.94	2.52	2.59
10%	2.13	2.91	2.83
20%	2.12	2.54	2.65
50%	2.08	2.67	3.01

Table 3.2.: Root Mean Square Error (RMSE) for Kriging, IDW, NN [25]

For the scope of this thesis a library that performs interpolation of scattered data will be applied. The SciPy library, a Python library, that provides the "scipy.interpolate.griddata" function that can be used for interpolating scattered data on a regular grid.

The available interpolation methods that the package provides are the following and the results can be visualized in Figure 3.3:

- Nearest Neighbor: In N 1 dimensional scenarios, nearest-neighbor interpolation identifies the data point that is closest to the specified interpolation point and returns the value associated with that nearest data point [56].
- Linear: The process involves creating an interpolant by partitioning the input data into triangles (triangulation) using Qhull, and then, within each of these triangles, applying linear barycentric interpolation to determine the desired values [56]).
- Cubic: The process involves a piecewise cubic interpolant that is C1 smooth and aims to minimize curvature. This interpolant guarantees continuity in its derivatives, ensuring a smooth transition between adjacent triangles [56]).

The interpolation needs to be performed for every hour for a specified time period (1 hour intervals) on the clean data. With this requirement in mind, the outcome of the interpolation process generates multiple raster files, each depicting temperature variations across

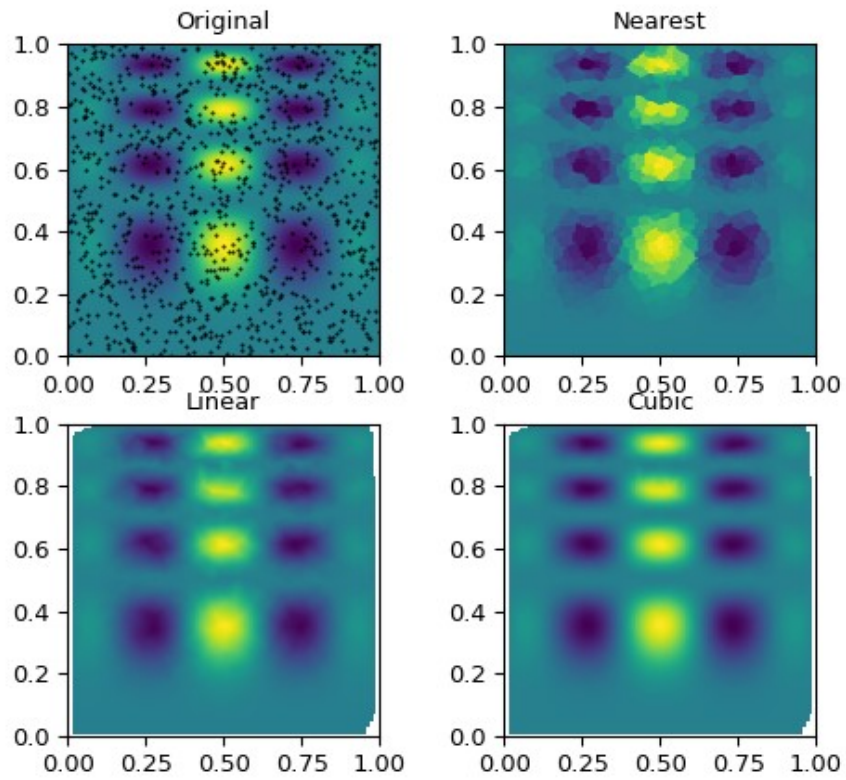


Figure 3.3.: `scipy.interpolate.griddata` different methods results

Source: [56]

3. Methodology

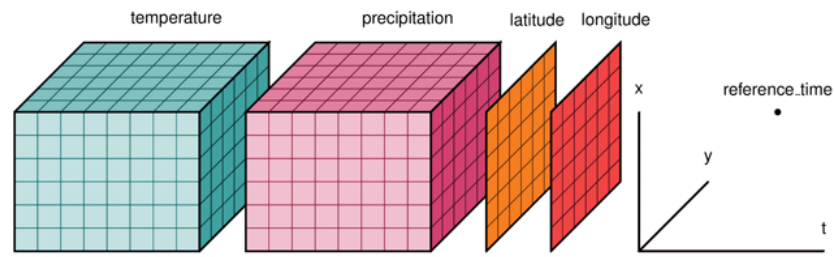


Figure 3.4.: netCDF or xarray structure

Source: [27]

a consistent grid, all within the same area but spanning various time instances. To effectively manage this spatio-temporal data new data formats such as xarray and NetCDF can be used. By doing so, we can introduce time as a third dimension, thereby improving the representation and accessibility of this spatio-temporal dataset, making it more structured and efficient for analysis and storage.

Xarray is a powerful Python library that provides N-D labeled arrays and datasets, allowing for efficient and flexible handling of multi-dimensional data. It provides a powerful indexing system that allows for easy slicing and subsetting of data based on dimensions and coordinates. Xarray also supports advanced operations such as mathematical computations, aggregation, and resampling of data [27]. One of the key features of xarray is its ability to work with NetCDF files, which are commonly used in the geoscience and atmospheric studies fields for storing and sharing scientific data. NetCDF (Network Common Data Form) is a file format that is widely used for storing and exchanging scientific data. It provides a self-describing and machine-independent format that allows for efficient storage and retrieval of multi-dimensional arrays. NetCDF files can store various types of data, including gridded data, time series, and metadata [27]. As it can be seen in Figure 3.4, within the dataset, there exist two three-dimensional data variables, temperature and precipitation. Additionally, the dataset has two-dimensional coordinates, latitude and longitude, alongside a reference time that serves as a zero-dimensional, scalar coordinate.

3.6. Publishing the interpolation results using OGC standards

The OGC standards are widely accepted and used in the geospatial community. They provide a framework for describing sensors and observations, enable interoperability and integration of sensor networks, and ensure the seamless integration and sharing of geospatial information and services. In the context of this thesis, OGC API standards will be used to serve the interpolation results through web services. The historical interpolated results that is the product of the clean data can be interpolated locally and then served through OGC API Coverages in a raster format or through OGC API Map for visualization purposes. On the other hand, the real time sensor data need to be interpolated on the fly, therefore using OGC API Processes is a solution that will provide instant results to the client.

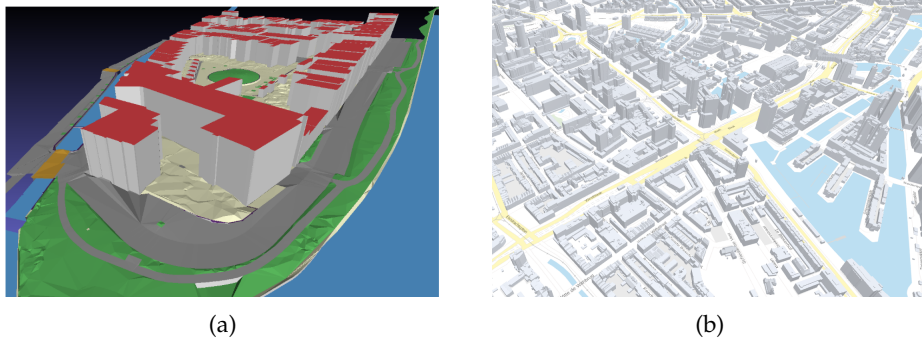


Figure 3.5.: 3D city models for the Netherlands. (a) 3D-Basisvoorziening (Geonovum, 2021)
(b) 3D BAG, LoD 2.2 (<https://3dbag.nl/en/viewer>)

3.7. Preparation of the 3D city model for CesiumJS

3.7.1. 3D city models

3D city models are created and used for various purposes, including urban planning, visualization, simulation, and analysis [60]. These models provide a digital representation of the Earth's surface and its related objects in urban areas. In the Netherlands there are available datasets for 3D city models, such as the 3D BAG in CityJSON [33] which is a homogeneous dataset for all of the Netherlands and the 3D-Basisvoorziening (Geonovum, 2021) dataset that is provided in CityJSON by the Kadaster and covers the entirety of the Netherlands as well. A visual representation of both datasets and their elements can be seen in Figure 3.6 .

3.7.2. OGC 3D Tiles

In order to be able to handle, import and visualize efficiently the dataset, it is need to be converted to a format that is appropriate for rendering large datasets. "3D Tiles is an open specification for sharing, visualizing, fusing, interacting with, and analyzing massive heterogeneous 3D geospatial content across desktop, web, and mobile applications" [11]. It is built on GL Transmission Format (glTF), another open specification developed by Cesium, and is optimized for streaming and rendering 3D geospatial content [11]. A tileset is a collection of tiles arranged in a hierarchical structure resembling a tree. It starts with a root tile, and each tile can have child tiles. Additionally, a tile can reference another tileset, allowing for flexible and hierarchical combinations of tilesets.

Tiles within a tileset can contain various types of renderable content, such as textured terrain, 3D models, or point clouds, each described in JSON format. The tileset JSON file contains essential information about the tileset and tile descriptions, including properties like geometric error, which measures the visual error if the tileset is not rendered. When this error surpasses a certain threshold, the tileset and its tiles become candidates for rendering. Tile properties can include additional attributes related to the content, like building heights in the case of buildings. Metadata about the 3D Tiles version and application-specific details can be stored in the asset property of a tileset. Regarding tile properties, content is the actual

3. Methodology

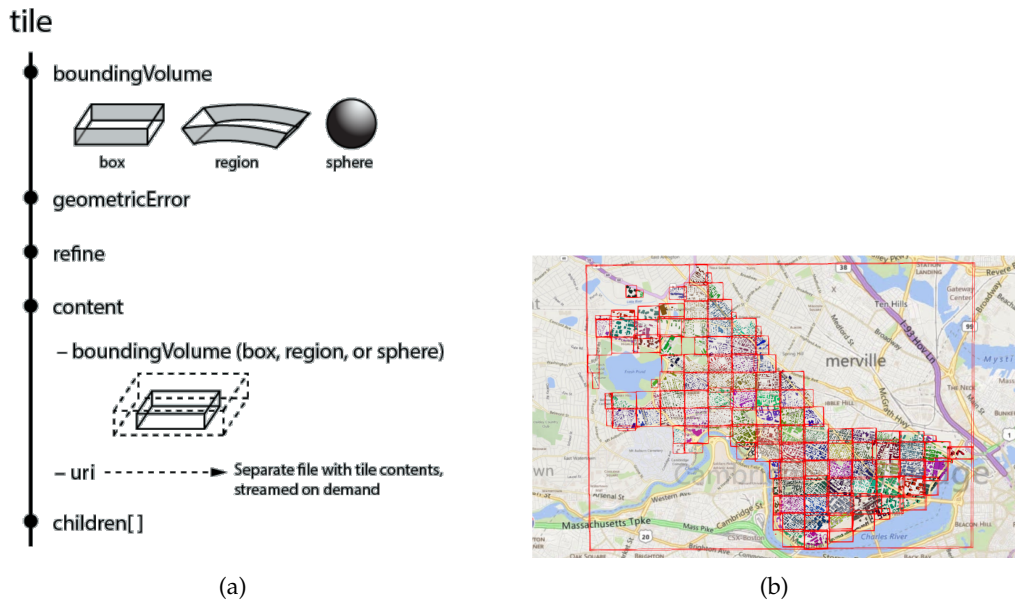


Figure 3.6.: 3D Tiles. (a) Elements of a tile JSON object (Open Geospatial Consortium, 2019)
(b) A tileset with an overlapping grid spatial data structure ([44])

renderable content referred to by a URI in the content property. Tiles form a hierarchical tree structure with children tiles nested within each tile. The refinement strategy determines how child tiles are used when the visual error of a tile with a certain level of detail exceeds a threshold, and the refine property specifies this strategy. Each tile has a bounding volume associated with it, described in the "boundingVolume" property, which can take various forms (sphere, box, region) and enclose both the tile's content and the content of its children, creating a spatially coherent hierarchy of bounding volumes. To manage different levels of detail in the renderable content of tiles, the "geometricError" property quantifies the simplification of the content compared to the highest level of detail. It's used to determine when child tiles should be considered for rendering, as detailed in the Geometric Error section. [11]

3.8. Visualization in CesiumJS

CesiumJS is a JavaScript based, virtual globe, platform for dynamic spatial data visualization [13]. It has grown into a leading platform for 3D web visualization and provides excellent support for the online visualization of geographic information in the 3D Tiles format [32]. Cesium enables the loading and rendering of 3D tiles. When it comes to its JavaScript-based platform, CesiumJS, it inherently accommodates all the formats, such as Batched 3D models (.b3dm), Instanced 3D models (.i3dm), Point Cloud tiles (.pnts) and Composite tiles (.cmpt). These formats can seamlessly integrate into the JavaScript application via a web-based hosted server, Cesium ion, responsible both for serving the tiles and for tiling datasets.

3.8.1. Loading 3D city models in CesiumJS through Cesium ion

Cesium ion provides a robust data pipeline that allows for the user to upload data in various formats and transform it to five types of assets (Table 3.3). CityJSON is not a directly supported format from Cesium ion, therefore the 3D city dataset (e.g. 3D BAG) needs to be first converted in CityGML and then uploaded in Cesium ion to be tiled directly. These are the main approaches that can be followed:

- Convert CityJSON to Cesium 3D Tiles using FME
- Convert CityJSON to CityGML using FME
- Convert CityJSON to CityGML using citygml-tools

Alternatively, other 3D city datasets can be used in CesiumJS to visualize the static elements of the city. Cesium OSM Buildings is a global 3D building dataset that is directly accessed through Cesium ion, making it easily integrable into custom applications using CesiumJS or any client that can handle 3D Tiles [13]. This dataset is sourced from OpenStreetMap and boasts a vast collection of more than 350 million buildings, each accompanied by detailed metadata. This metadata encompasses fundamental details such as building names and heights, as well as more comprehensive information like addresses, operating hours, and even specifics about the materials used in various parts of the buildings [13]. The advantage that offers is that the user does not to acquire, convert and upload datasets into Cesium ion or fit the data to the terrain.

For the context of this thesis, the 3D BAG will be used and the direct approach of converting CityJSON to CityGML using citygml-tools, as the tool provides the option to convert all the tile files with a single command and then to upload them in cesium ion directly.

3. Methodology

Format	3D Tiles	Terrain	Imagery	glTF	Native
Zip Archive (.zip)	X	X	X		
glTF (.gltf, .glb)	X			X	
Filmbox (.fbx)	X			X	
CityGML (.citygml, .xml, .gml)	X				
CZML (.czml)					X
GeoJSON (.json, .geojson, .topojson)				X	
KML (.kml, .kmz)	X				X
LASer (.las, .laz)	X				
COLLADA (.dae)	X			X	
Wavefront OBJ (.obj)	X			X	
Floating Point Raster (.flt)		X	X		
Arc/Info ASCII Grid (.asc)		X	X		
Source Map (.src)		X	X		
GeoTIFF (.tiff, .tif)		X	X		
Erdas Imagine (.img)		X	X		
USGS ASCII DEM and CDED (.dem)		X	X		
JPEG (.jpg, .jpeg)			X		
PNG (.png)			X		
Cesium Terrain Database (.terraindb)	X				

Table 3.3.: Cesium ion accepted data formats and asset types [13]

4. Implementation

In this section of the current thesis, the implementation steps will be explained. The main goal is to create a pipeline that will be as automated as possible, with minimal user interactions. The starting point of the research is Node0red, a JavaScript tool that allows the creation of HTTP requests, processes running on specific time frames and it also allows for the integration of various packages. Even though node-red is based on JavaScript, other programming languages were integrated as well, such as Python, by creating virtual environments and connecting them in the application. Besides node-red, several other servers needed to be deployed and run, such as GeoServer, FROST-Server, pygeoapi server and the Apache Tomcat to host the web application for CesiumJS.

In the following Figure 4.1 the main components and applications are represented through a graph.

4.1. Tools and datasets

The entirety of this thesis was developed and hosted locally and the hardware used consists of a personal computer, with CPU AMD Ryzen 5 4500U, clocked at 2.38 GHz and 16GB of RAM. All the datasets and tools that were used in this thesis are presented as follows:

- **Node-Red:** The backbone of the whole application. It is a JavaScript application that is used to host the pipeline from making HTTP requests to Netatmo APIs and storing the data as SensorThings API entities. Also, used to update the real-time measurements, collect the historical data and perform data cleaning for the historic measurements.
- **Docker:** Is an application that offers an efficient and easy way to deploy and run applications. In this research it is used to run FROST-Server, but it can also be used to host Node-red, R-Server, pygeoapi and many other applications.
- **Netatmo:** Netatmo temperature data is retrieved from the Netatmo public APIs.
- **FROST-Server:** Is used for implementing the SensorThings API standard.
- **PostgreSQL & PostGIS:** Are used in combination with the FROST-Server to store all the measurements received from the Netatmo sensors.
- **QrowdQC+:** An R tool that provides a complete solution to temperature crowd-source data cleaning.
- **rpy2:** It is a Python package that allows the user to run any R code from Python. It is used to run QrowdQC+ within the pipeline of Node-Red, using a Python virtual environment.
- **SciPy:** This Python library is used to interpolate the scattered sensor data to a regular grid.

4. Implementation

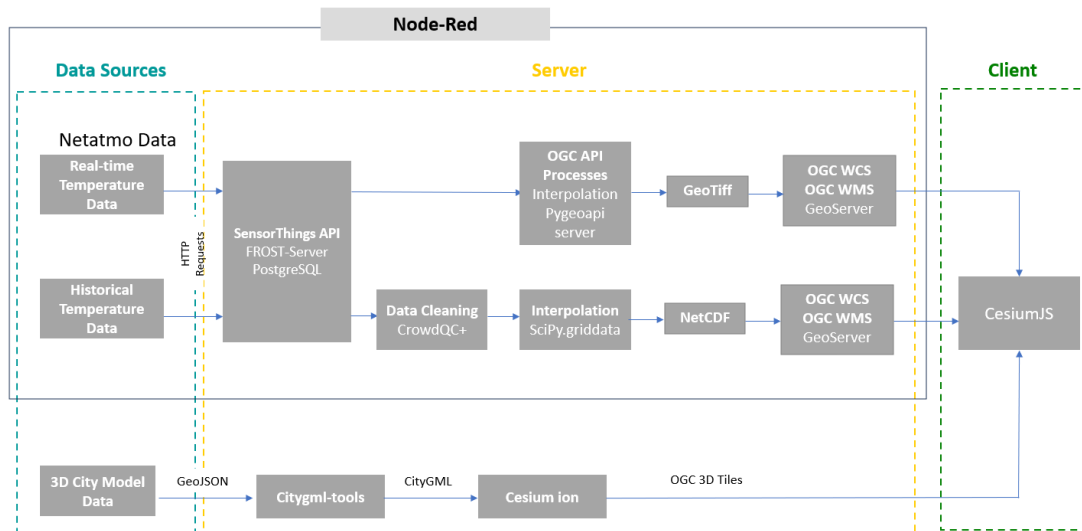


Figure 4.1.: Implementation overview

- **Xarray:** It is a Python library that is used to handle the spatio-temporal interpolation results and interacts with the final NetCDF format of the data.
- **pygeoapi:** It is a modern, python implementation of OGC API standards and is the server that hosts and publishes the interpolation process and results.
- **GeoServer:** It is a well known and widely used, server for hosting and publishing geospatial data. The formats that it accepts vary between raster data, vector data, spatio-temporal data. It supports the first generation of OGC web standards.
- **3D BAG:** Tiles from Rotterdam were used to be imported in the final 3D visualization of the application
- **citygml-tools:** It is used to convert CityJSON to CityGML to be used as input in Cesium ion.
- **CesiumJS:** It is hosting the final application that visualizes the temperature results, based on the time.
- **Cesium ion:** It is the application that is responsible for the tiling of the 3D dataset of Rotterdam and also hosts the CityGML dataset. Additionally it can be used to store raster data.
- **Apache Tomcat:** Is used to host the CesiumJS application and to serve it in a web environment.
- **Visual Studio Code:** Used for implementing the needed coding for the final CesiumJS application, using JavaScript
- **PyCharm:** Used to host and debug the code that was needed for the data cleaning process.
- **QGIS:** Used for visualizing the location of the sensors on the map in different stages of the process, as well as the results of the interpolation process.

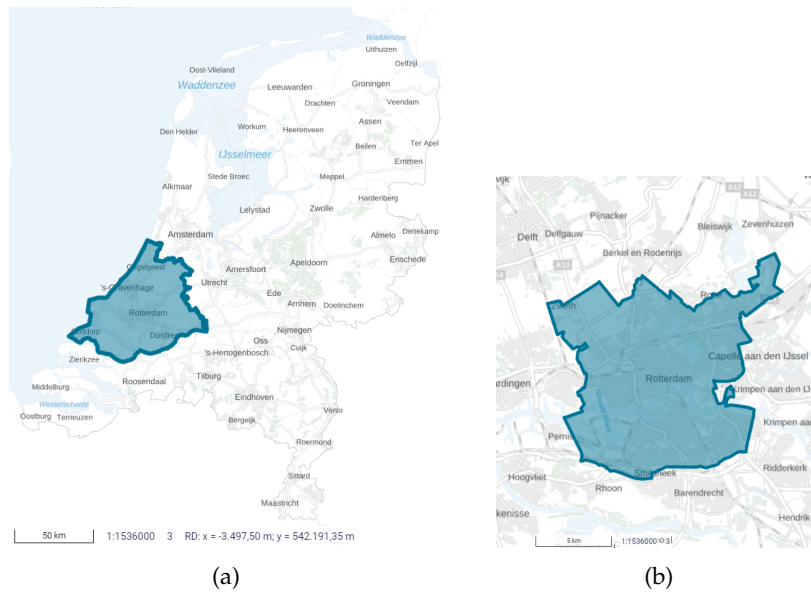


Figure 4.2.: Study area location (a) South Holland (Source: PDOK viewer) (b) the city of Rotterdam (Source: PDOK viewer)

4.2. Study Area

The current research focuses on providing temperature data for the city of Rotterdam, in the Netherlands (4.2, with a population of 663.900 inhabitants [10] Rotterdam is taking resilience and sustainability into consideration as it is one of the cities that are included in the 100 Resilient Cities Programme [61]. Urban resilience is for a city to “be able survive, adapt, and grow regardless of the kinds of chronic stress and acute shocks it experiences” [61]. One major aspect of this is proper planning for climate adaptation, including temperature in the city. Therefore, by researching and providing temperature data for Rotterdam, this thesis explores the possible opportunities for the city to plan and prepare for future heat related events, by providing a user-friendly tool.

4.3. Real-time sensor data pipeline

This section focuses on the acquisition, handling and serving of the real time temperature data from the Netatmo sensor network. This pipeline differs from the historical data pipeline, mainly due to the fact that real time sensor data cannot be cleaned in an efficient way, as the available cleaning methods require a dataset that expands through a large time period of time, for example, at least 7 days of measurements in an hourly rate. Additionally, the real time sensor data can be acquired from Netatmo API every ten minutes and do not need to be stored in the same way as the historical data, as this data are not reliable enough due to the known crowd-sourced data errors and cannot be used for future applications.

4. Implementation

4.3.1. Acquisition of real-time sensor data

As long as the Netatmo network is used, the user needs to create an account and create an application. Through this process a client ID and a client secret are generated that will be used as credentials to access the APIs. The public, real-time (every ten minutes) temperature sensor data is accessed through the Netatmo's API `"/getpublicdata"` by making an HTTP request. Following, the body of the request will be explained. In Figure 4.3 the response schema is presented. By parsing the JSON response through node-red, the entities on the server are created.

- **lat-ne:** This parameter describes the latitude of the north east corner of the area of interest and accepts values between -85 and 85.
- **lon-ne:** This parameter describes the longitude of the north east corner of the area of interest and accepts values between -180 and 180.
- **lat-sw:** This parameter describes the latitude of the south west corner of the area of interest and accepts values between -85 and 85.
- **lon-sw:** This parameter describes the longitude of the south west corner of the area of interest and accepts values between -180 and 180.
- **Filter:** This parameter performs a first stage data cleaning. By default, stations with abnormal temperature readings are included in the APIs response, but setting this parameter to 'true' they are excluded.

4.3.2. Integrate the SensorThings API standard using FROST-Server

This is a crucial step of the methodology and the pipeline of the thesis as the main SensorThings API entities are created. First and foremost, the whole implementation takes place through node-red and the entities are created on a locally hosted and deployed instance of the FROST-Server that can be accessed through `"http://localhost:8080"/FROST-Server/v1.1/"`. along with the server, a PostgreSQL database is also deployed and connected to store the entities that are created.

The first entity that is created in the FROST-Server is the "Thing" entity which is assigned a unique "@iot.id" automatically from FROST-Server. Each "Thing" represents a Netatmo station and it is uniquely identified by a MAC address that is returned from the Netatmo endpoint and is stored in the FROST-Server as the "name" property of the "Thing". Each "Thing" is associated with the "Locations" and "Historical Locations", expressed as point with latitude and longitude coordinates. Also, each "Thing" entity is associated with specific "Datastreams". In the case of the real-time data, the measurements are stored in the "Outdoor Temperature" "Datastream". By selecting a specific datastream is easier to distinguish the real-time, raw data with the historical and pre-processed data. For the current implementation, the "Observed Property" entity is only the outdoor temperature and the "Sensor" entity is also only one as only the temperature module of the Netatmo station is taken into consideration in this research.

Every time a get request to the Netatmo endpoint is made, another request to the FROST-Server is made in order to check if the MAC addresses in the Netatmo response are already stored in the FROST-Server. This would mean that the "Thing" entity is already present and only the "Observation" needs to be updated with a new measurement. On the other hand, if

4.3. Real-time sensor data pipeline

```
{
  "status": "ok",
  "time_server": 1695629535,
  "body": [
    {
      "_id": "70:ee:50:52:f0:c0",
      "place": {
        "location": [
          4.355137,
          52.00447
        ],
        "timezone": "Europe/Amsterdam",
        "country": "NL",
        "altitude": 1,
        "city": "Delft"
      },
      "mark": 10,
      "measures": {
        "02:00:00:53:00:58": {
          "res": {
            "1695629034": [
              15,
              77
            ]
          },
          "type": [
            "temperature",
            "humidity"
          ]
        },
        "70:ee:50:52:f0:c0": {
          "res": {
            "1695629052": [
              1022.4
            ]
          },
          "type": [
            "pressure"
          ]
        }
      },
      "modules": [
        "02:00:00:53:00:58"
      ],
      "module_types": {
        "02:00:00:53:00:58": "NAModule1"
      }
    }
  ]
}
```

Figure 4.3.: Netatmo's "/getpublicdata" endpoint response schema

4. Implementation

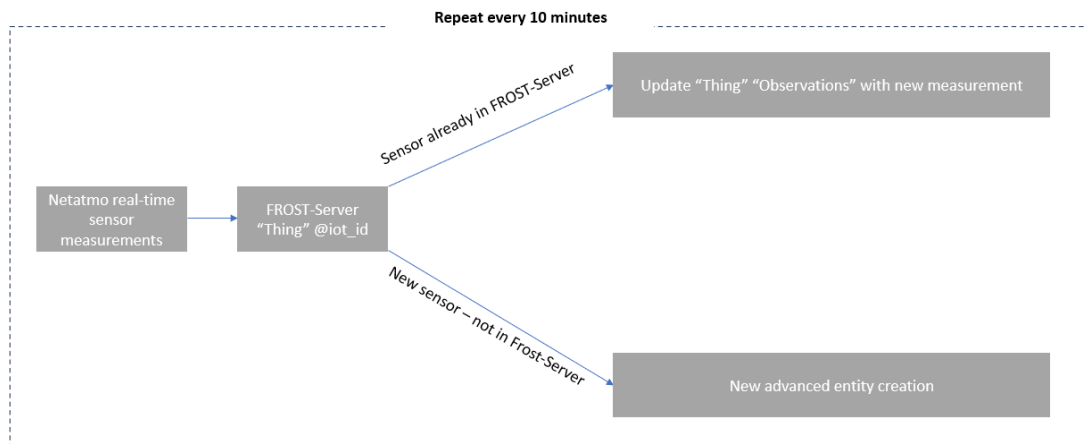


Figure 4.4.: Real-time, raw data to SensorThings API methodology, part 1

the MAC address is not found in the FROST-Server, then a new "Thing" entity is created along with its relations with one POST request, such as "Locations", the three potential "Datastreams", "Sensor", "Observed Property". As soon as the new entity is created the "Observation" is also updated. A graphic representation of the methodology is depicted in Figure 4.4.

4.3.3. Data interpolation

For the real-time data interpolation, the latest sensor measurements are collected from the FROST-Server and the whole process takes place in node-red. First a "date" JavaScript object is created that holds the current time and then an HTTP GET request is made to FROST-Server that requests the latest "Observations" by filtering based on the current time and the observation's "Phenomenon Time". On the same query, the location of the sensor is also collected by using "&\$expand=FeatureOfInterest", which contains the latitude and longitude information. This addition to the pipeline is visualized in Figure 4.5.

For the real-time data interpolation, the SciPy library is used through the "scipy. interpolation. griddata". This library gives three options for interpolation: linear, cubic and nearest neighbor. As it is also mentioned in the methodology section, the nearest neighbor interpolation approach will be followed for the current thesis. First, the data for the interpolation is retrieved from the FROST-Server through an HTTP GET request in the node-red environment. Next, the interpolation is performed through another HTTP POST request that executes and runs the developed OGC API Process, which is presented in detail in the next section.

4.3.4. OGC API Processes

In order to perform the real-time data interpolation, a custom OGC API Process is developed and published in the locally deployed pygeoapi server. Pygeoapi is an open source, RESTful, Python server implementation that supports the new generation of OGC standards, the OGC

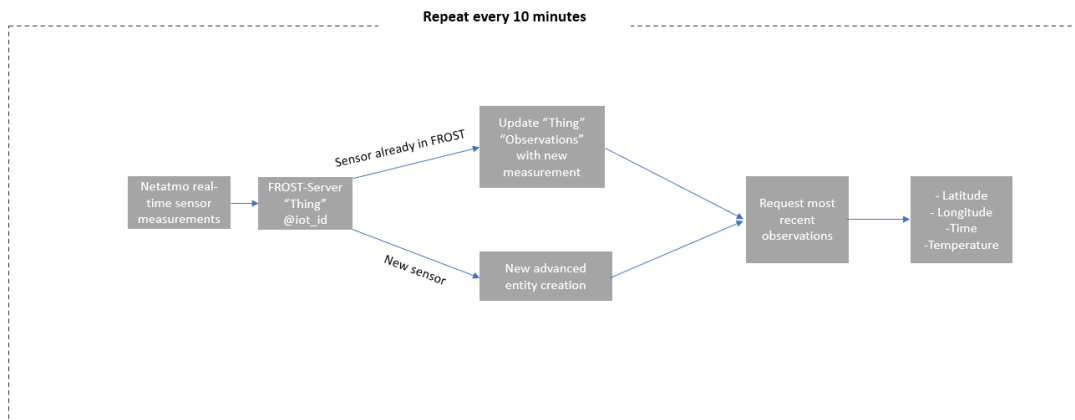


Figure 4.5.: Real-time, raw data to SensorThings API methodology, part 2

API standards [53]. Pygeoapi supports API Processes in the form of plugins. The server was deployed using a Python virtual environment and the process is developed also in Python. In Figure 4.6 the process's landing can be seen along with the relative links to its jobs or its description in JSON. This processes can be either synchronous or asynchronous as it is a geospatial process that is possible to be long and therefore the asynchronous functionality is an advantage.

Since the interpolation is performed in the pygeoapi server, an HTTP request needs to be made to the server that specifies the process that needs to be called with the `"/execute"` parameter to actually execute it. The input of the request is a JSON body and an input example with only a few points can be seen in Figure 4.7. The result of the process is a GeoTiff file and is received a Base64 encoded format.

4.3.5. Publishing the interpolation results

After receiving the server's response with the interpolated results, the data needs to be transform to a format that can be used for web map visualization or any other further use. To perform this step, a Python script is added in the pipeline and the Base64 scheme along with "rasterio" are used to decode the string and create the tiff file. The resulted file is named according to the time instance it was created and it is stored in GeoServer. The next step is to upload this result to the locally deployed GeoServer. Uploading the raster as GeoTiff files is done also through node-red and an HTTP POST request, as GeoServer provides a RESTful interface that is used to create the store for the real-time interpolation dataset and later update it through the base URL <http://localhost:8082/geoserver/rest/workspaces/currentWorkspace> and can be expanded as needed to access the different Stores. In Figure 4.9, the whole pipeline for handling the real-time data is presented.

4. Implementation

The screenshot shows the landing page for a custom API process named "Temperature Interpolation" in the pygeoapi framework. The page includes a header with the pygeoapi logo and a "Contact" link. Below the title, a description states: "A process that performs temperature interpolation based on provided observations." There are three tags: "temperature", "interpolation", and "grid".

Id	Title	Data Type	Description
lon	Longitude	array	Array of longitude values
lat	Latitude	array	Array of latitude values
temp	Temperature Values	array	Array of temperature values

Inputs

Id	Title	Description
geotiff	Interpolation Result	A JSON object containing the result as a base64-encoded GeoTIFF.

Outputs

Execution modes

- Synchronous

Jobs

[Browse jobs](#)

Links

- [Process description as JSON \(application/json\)](#)
- [Process description as HTML \(text/html\)](#)
- [jobs for this process as HTML \(text/html\)](#)
- [jobs for this process as JSON \(application/json\)](#)
- [Execution for this process as JSON \(application/json\)](#)

Figure 4.6.: Custom API Process landing page

```
{
  "inputs": {
    "lon": [4.359311, 4.352610, 4.355137, 4.365177, 4.359705,
            4.372800, 4.367903, 4.367653, 4.354034, 4.355313, 4.368141],
    "lat": [52.011700, 52.007328, 52.004470, 52.009139, 52.009701,
            52.004501, 52.009334, 52.004136, 52.008038, 52.005088, 52.009402],
    "temp": [15.20, 16.23, 17.25, 17.90, 18.40, 17.17, 16.87,
             16.10, 16.37, 14.50, 15.47]
  }
}
```

Figure 4.7.: Example of the JSON body of the process request

4.3. Real-time sensor data pipeline

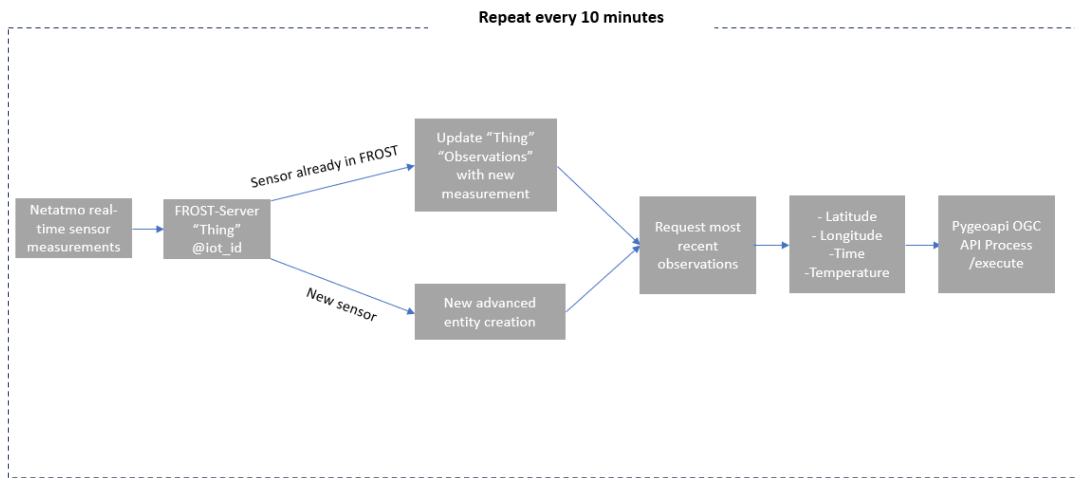


Figure 4.8.: Real-time, raw data methodology, part 3

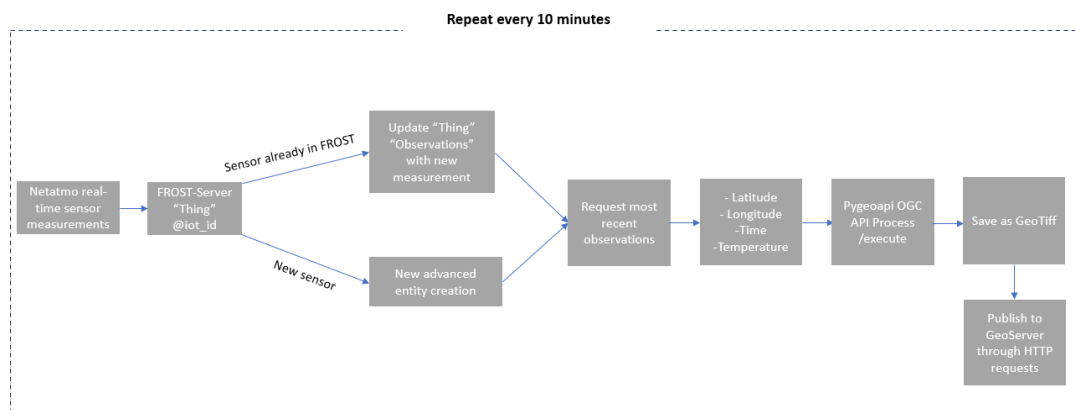


Figure 4.9.: Real-time, raw data methodology, part 4

4. Implementation

4.3.6. Accessing the interpolated data through CesiumJS

In spite of the fact that the interpolated results are delivered through GeoServer as a Web Coverage Service, CesiumJS is unable to handle this particular format. Consequently, the grid is presented as a Web Map Service using the "WebMapServiceImageryProvider" option within the CesiumJS library. This option enables the display of tiled imagery from a WMS service in Cesium. Cesium is locally deployed via Apache Tomcat, and the CesiumJS library is employed to create a 3D geospatial application for visualizing the interpolation results.

Within this setup, Cesium sends a direct WMS request to GeoServer, resulting in the addition of a new layer to the viewer. When the real-time interpolated layer is integrated into Cesium, the layer's "time" parameter is synchronized with the layer's name. Consequently, the layers are dynamically switched in accordance with a looping mechanism that retrieves information for the "currentTime" and imports the corresponding layer from GeoServer

4.4. Historical sensor data pipeline

This section focuses on the acquisition, handling and serving of the historical temperature data from the Netatmo sensor network. In the context of developing a tool that provides insight into the patterns of the city's temperatures in a detailed level, it is important to have historical data to provide this information. The pipeline for the historical data differs from the one for the real-time data, as historical data can go through an effective cleaning process of the raw measurements and then be stored and accessed through spatio-temporal datasets and formats. The acquisition of the historical data comes as a second step after storing the real-time data in the FROST-Server.

4.4.1. Acquisition of historical sensor data

The sensor data is accessed through the Netatmo's API "/getmeasure" by making an HTTP request. Following, the body of the request will be explained and how specific information from this request is obtained.

- **Device Id:** This is the weather station's mac address. In order to get this specific mac address, a request to the FROST-Server has been made to retrieve the corresponding "id" element from all the "Thing" entities that are stored through the real-time data pipeline. This can cause to potentially have less stations for the historical data as some sensors which are retrieved real-time might be new and not up and running the past months. On the other hand, some sensors that were active in the past might not be anymore, and there their mac address is not known and their data cannot be retrieved.
- **Scale:** This refers to the time intervals of the measurements and it can be 30 minutes, 1 hour, 3 hours, 1 day, 1 week and 1 month. Considering also the requirements for the data cleaning tool, measurements of 1 hour intervals are selected.
- **Type:** The type of the data that the API returns can be defined here. This can be an array containing different environmental variables, but for this thesis only the temperature is selected.

- **Date Begin and Date End:** Defines the timestamp (Local Unix Time in seconds) of the first measurement and the last measurement to be returned. For this implementation these timestamps are set to represent a month, in order to not exceed the APIs user and application limits.

4.4.2. Storing in FROST-Server the historical sensor data

Integrating the response of the Netatmo's API with the SensorThings API standard, for this step is translated to storing the measurements in a suitable way to FROST-Server. As soon as the data is requested from the Netatmo's API, with the use of node-red, a dictionary is created that correlates the "device id" with the "measurement". This is used to trace the corresponding "thing" entity and "datastream" to store the historical data in FROST-Server. The important part in storing this historical data is that a new "datastream" is created for each "thing" entity, called "Historical Outdoor Temperature" and contains all the available raw historical measurements.

4.4.3. Historical data cleaning

The next step after obtaining the sensor historical data is to perform the cleaning process. In this stage, the CrowdQC+ tool is employed. CrowdQC+ is an R-based tool, and to seamlessly incorporate it into the Node-RED flow, two primary alternatives exist. The first choice is to utilize the Node-RED package known as "node-red-contrib-R-nodes," which interfaces with an R server. The second option involves harnessing the "rpy2" tool in Python by creating a Python virtual environment equipped with all the necessary dependencies and executing the R tool through a Python script within Node-RED. For the current thesis project, the second option was selected, as it presented a more streamlined approach, eliminating the need to integrate an additional server into the overall implementation.

CrowdQC+ requires a specific format of a data table as input that contains the following columns (Fenner et al,2021):

- **p-id:** Unique ID of each station. Data format: Integer or character
- **time:** Time. Data format: POSIX.ct
- **ta:** Air-temperature values. Data format: Numeric/double
- **lon:** Longitude of the station (WGS 84). Data format: Numeric/double
- **lat:** Latitude of the station (WGS 84). Data format: Numeric/double

To be able to deliver the data in the format that is required from the R tool, the historical data is requested from the FROST-Server and processed through SQL queries for a monthly time frame. As soon as the data is imported to CrowdQC+, a check is performed with an additional data padding function, that fills the gaps per station, in order to have a table with hourly measurements for every station.

The crowdQC+ tool has five stages of quality control. Based on the documentation the following is used:

- **Main QC step m1:** The number of stations that are allowed to have the same location is set to 1, therefore eliminates the stations with false latitude and longitude values.

4. Implementation

- **Main QC step m2:** Checks for the stations that fall outside of the low and high boundaries of the robust z-score. Low (=0.01) and high (=0.95) are kept default as it is recommended in the documentation. The "height correction" is manually set to False as it is needed for areas with flat geography, such as Rotterdam. As long as there are more than 100 sensors, the "t-distribution" is set to false.
- **Main QC step m3:** The "cutoff" value that determines the percentage above of which data that were flagged as erroneous in step m2 is set to 20%. This check ensures that if too many measurements of a sensor may be wrong, then the whole sensor should be removed. It is recommended to apply this check for the duration of 1 month.
- **Main QC step m4:** The "cutoff" value in this step is set to 0.9 and represents the correlation coefficient between a specific sensor and the median of all stations. Here, data for 1 month are considered.
- **Main QC step m5:** The spatial check is performed in this step, by checking for spatial outliers by comparing a station's values with those of its neighboring stations within a specified radius. Stations with too few neighbors or large elevation differences may be flagged as erroneous. The default radius is 3000m and the minimum number of neighbouring stations is set to 5. Considering the area's size and number of stations, the default setting should give a trusted result.

Storing in FROST-Server the clean historical sensor data

The output of the cleaning process is used to update the entities on the FROST-Server, by adding the resulted measurements in the "Pre Processed Historical Outdoor Temperature" data stream, of the "things" that correspond to the sensors that made it through the process. By adding this extra "Datastream", the possibility of more efficient and advanced queries and data processing is given to the users of the server. For example, in order to retrieve observations of the clean sensor data but for a specific time this query can be used:

```
http://localhost:8080/FROST-Server/v1.0/Observations?$filter=Datastream/name eq 'Pre Processed Historical Outdoor Temperature' and phenomenonTime ge 2023-05-01T00:00:00Z and phenomenonTime le 2023-05-17T10:00:00Z
```

4.4.4. Data Interpolation

Considering the fact that the historical data are not dynamic, the interpolation does not need to be executed on the fly, like real-time data. Therefore, through node-red, an HTTP request is made to the FROST-Server to return the measurements from the "Pre Processed Historical Outdoor Temperature" datastream. The return of the request is used as an input for the interpolation. The historical data interpolation process can handle as input measurements of different sensors for a quite large time-frame (e.g. 1,2 or 3 months) and performs sequential interpolations of data that represent the same time. In this thesis the clean data are returned in 1 hour intervals and therefore the interpolations are performed for each hour within the available dataset. As a last step of the interpolation process, the output is a spatio-temporal dataset in the format of NetCDF that encapsulates the different rasters and correlates them with a timestamp. As it can be seen in the visualization in QGIS in Figure 4.12, a NetCDF treats the data series as different bands and therefore for a specific latitude and longitude pair, there are different temperature values across time.

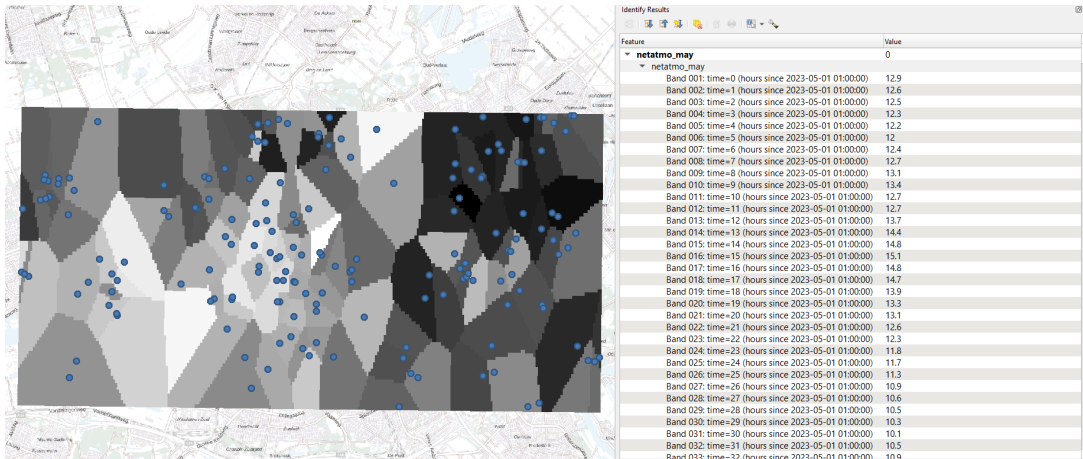


Figure 4.10.: NetCDF interpolation result on QGIS

There are many available libraries and techniques that interpolate scattered data to a regular grid, but for the scope of this thesis the SciPy library is utilized and the “nearest neighbor” interpolation method is selected. This might be a quite simple and straightforward approach, compared to the linear and cubic interpolation methods that this library also provides, but it is the only one out of the three that performs an extrapolation. In the case of historical measurements from sensors, there is always the possibility that certain sensors will not be present throughout the course of time and therefore the resulted interpolated raster will have different extents for different timestamps. This could cause an issue as long as the results are stored in a NetCDF, that requires specific spatial extents for all the rasters at all time instances.

Export to NetCDF data format

In an effort to standardise, simplify and make the whole process easily reproducible, the “geocube” package is used to perform the interpolation based on the “scipy.interpolate.griddata” and later transform the result into an xarray that can be exported directly into a NetCDF file. The resulted file has three dimensions: time, latitude and longitude and the meta-data attributes that were assigned to describe these variables are based on the CF Metadata Conventions [?].

- **Time:** Follows the UDUNITS documentation. For our data the “hours since” is selected as there hourly data and the format of the time is based on the ISO8601 standard. The “standard name” of the variable is based on the CF Standard Names and “time” is selected. The “coverage-content-type” variable is set to “coordinate”, based on the Attribute Convention for Data Discovery (ACDD).
- **Latitude:** “Units” are set as “degrees-north”. The “coverage-content-type” variable is set to “coordinate”.
- **Longitude:** “Units” are set as “degrees-east”. The “coverage-content-type” variable is set to “coordinate”.

4. Implementation

```
netatmo-temperature:
  type: collection
  title: Netatmo temperature
  description: Outdoor temperature from Netatmo sensors
  keywords:
    - Netatmo
    - sensors
  links:
    - type: text/html
      rel: canonical
      title: information
      href: https://dev.netatmo.com/apidocumentation/weather
      hreflang: en-US
  extents:
    spatial:
      bbox: [4.3598377, 51.87996, 4.63425776535564, 51.965575]
      crs: http://www.opengis.net/def/crs/EPSG/0/4326
    temporal:
      begin: 2023-05-01T01:00:00Z
      end: 2023-05-30T23:00:00Z
  providers:
    - type: coverage
      name: xarray
      data: data/interpolated_temperature.nc
      format:
        name: netcdf
        mimetype: application/x-netcdf
```

Figure 4.11.: Xarray provider in pygeoapi

- **Temperature:** "Units" are set as "degrees Celsius". The "coverage-content-type" variable is set to "modelResult", as it is the interpolation result. The "standard name" is set to "air-temperature" according to the the CF Standard Names.

4.4.5. Publishing the interpolation results

In order to make the interpolation results available for further use, the pygeoapi server is used. Since the pygeoapi server follows the new OGC API standards, the API Coverages would be a good alternative to store raster data as it enables a Web API access to gridded data as well as data cubes. To be able to do this, pygeoapi utilizes two main providers: rasterio and Xarray. As long the interpolation results are stored in a NetCDF file, Xarray needs to be added as a plugin to the pygeoapi configuration as it is described in Figure 4.11. This data can later be accessed through the following URL: <http://localhost:5000/collections/netatmo-temperature/coverage>.

Taking into account the goal of visualizing the result in Cesium, another approach must be taken in terms of publishing the interpolation results. An apt solution for this purpose would involve the utilization of the OGC API Maps (draft) standard, which is currently in the process of implementation within pygeoapi. This implementation employs the Mapscript and WMS facade providers to enable the direct publication of data as a Web Map Service (WMS) accessible by Cesium. However, it's worth noting that the OGC API Maps implementation in pygeoapi does not provide support for NetCDF files. Consequently, in

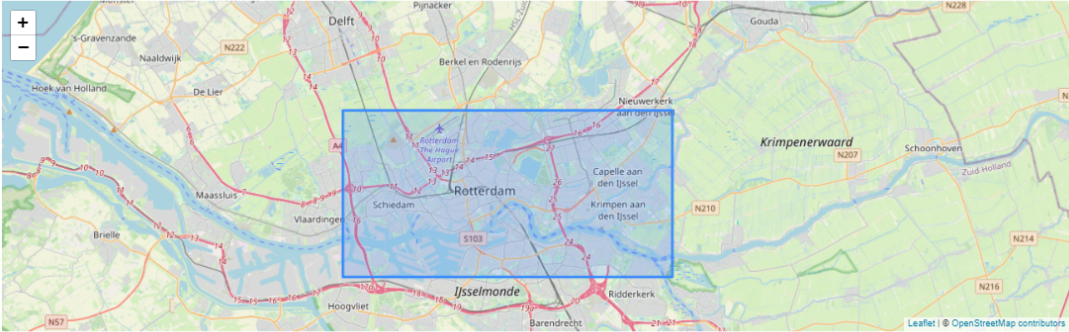
pygeoapi Contact

Home / Collections / Netatmo temperature json jsonld

Netatmo temperature

Outdoor temperature from Netatmo sensors

Netatmo sensors



Leaflet | © OpenStreetMap contributors

Links

- [information \(text/html\)](#)
- [The landing page of this server as JSON \(application/json\)](#)
- [The landing page of this server as HTML \(text/html\)](#)
- [This document as JSON \(application/json\)](#)
- [This document as RDF \(JSON-LD\) \(application/ld+json\)](#)
- [This document as HTML \(text/html\)](#)
- [Detailed Coverage metadata in JSON \(application/json\)](#)
- [Detailed Coverage metadata in HTML \(text/html\)](#)
- [Coverage domain set of collection in JSON \(application/json\)](#)
- [Coverage domain set of collection in HTML \(text/html\)](#)
- [Coverage range type of collection in JSON \(application/json\)](#)
- [Coverage range type of collection in HTML \(text/html\)](#)
- [Coverage data \(application/prs.coverage+json\)](#)
- [Coverage data as netcdf \(application/x-netcdf\)](#)

Figure 4.12.: OGC API Coverages landing page in pygeoapi server

4. Implementation

the current thesis, the decision to create NetCDF files, because they are more efficient in terms of size and data structure, in comparison to multiple, individual raster images.

Hence, an alternative strategy was adopted, involving GeoServer and the utilization of GeoServer's NetCDF plugin instead of the pygeoapi server. GeoServer is an open-source server software designed for the management, processing, and distribution of geospatial data over the internet [22]) It implements mainly the first generation of OGC Web Standards (WCS, WFS, WPS, WMS, WMTS) and various geospatial data formats. After publishing the NetCDF layer in GeoServer as WMS, an SLD file is uploaded and selected as the default styling format, in order to be able to render the data on a web map.

GeoServer is deployed locally and due to its Restful configuration, a Python script is executed and requests are made through the node-red pipeline, to import, update and delete the NetCDF files or stores. In the pipeline, after interpolating the data and storing it locally as a .nc file, the data is zipped to be able to transferred through type "application/zip". An HTTP POST request is made to "http://localhost:8082/geoserver" to create a new store, then a request to add a layer and lastly a request to add the selected styling file (.sld). After successfully publishing the data to GeoServer, the pipeline for the acquisition, processing and making the data available through web services is complete and the data can be used in a web client, such as Cesium.

4.4.6. Accessing the interpolated data through CesiumJS

After deploying Cesium and GeoServer locally through Apache Tomcat, the data can be accessed and visualized in the client. The data is requested as a WMS from GeoServer, directly through the CesiumJS library and the "WebMapServiceImageryProvider" option, with the specified timing parameter. This time parameter is used to display the corresponding raster for each timestamp in the geoportal.

4.5. 3D city models to 3D Tiles

In the current thesis a direct and minimal approach is needed, that fits into the entire pipeline and does not add many extra steps into the whole process. Therefore, the conversion of CityJSON to CityGML is done using citygml-tools that can convert multiple CityJSON files at once and then the converted files can be zipped and uploaded directly in Cesium ion, which will be responsible for tiling the dataset in an automated way. The original tiles are acquired from 3D BAG. The data flow from Cesium ion to CesiumJS can be seen in Figure 4.13.

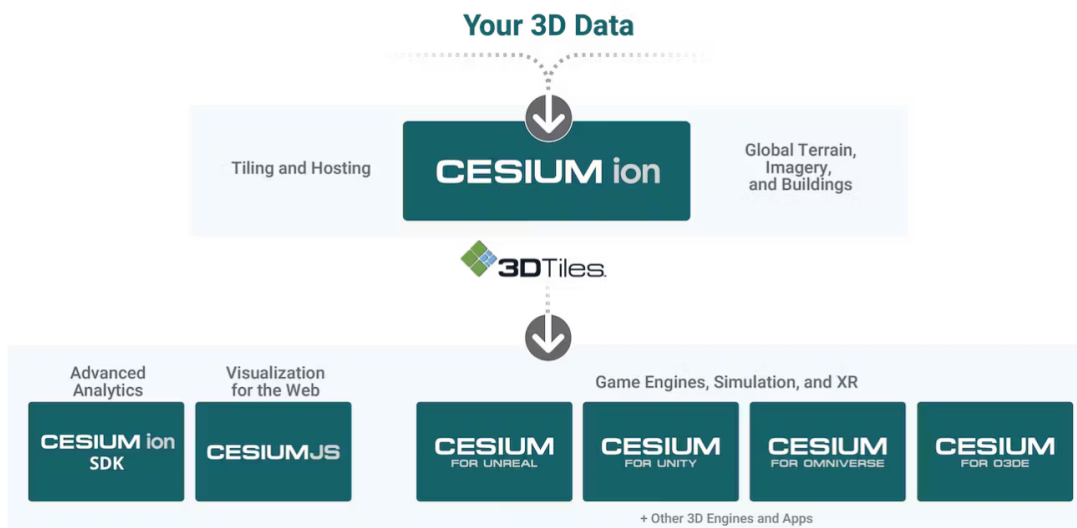


Figure 4.13.: Cesium platform. Source: Cesium, <https://cesium.com/platform/>

5. Results and Discussion

In this section an effort to present the results of the current research will be made. This includes both the implementation for the real-time temperature data and the historical data, as well as the final visualization in CesiumJS. In this section also the interpolation results will be presented and discussed. Each section focuses on one aspect of the application but in many cases the steps are closely correlated and therefore some overlaps may occur.

5.1. Netatmo data acquisition

Regarding the real time data the available sensors for the area of Rotterdam are approximately 200 sensors. This number cannot be definitive as it fluctuates because not all the sensors are available at all times. This specific area is selected as it covers the more central areas of the city, along with some of the lower rise-suburban areas. In Figure 5.1 the locations of the stations for the 15th of September 2023 are depicted on a map. Indicatively, the real-time API retrieved 208 stations at 18:20 on the 15/09/2023. During the next one hour the number of sensors was remaining the same with maximum 2 sensors that were occasionally not detected or newly added. This number of sensors provides a good number of data to perform the interpolations as well as the cleaning process.

The sensors that were detected as active during the server's running hour on the 15/09/2023, were used to retrieve the historical measurements. The historical measurements were retrieved for May, 2023 and the active sensors in total were 197. This number does not mean that all of these sensors were active and were providing measurements for all the 1 hour intervals for the whole month. In Figure 5.2 the locations of the detected sensors for the whole month are presented. Even though there are fewer sensors, the network density is still way higher than the available KNMI weather stations.

5.2. Historical data cleaning

After performing the data cleaning based on the criteria described in the implementation section, 19 sensors were removed. In Figure 5.3 the sensors that are depicted with the blue color symbolize all the valid sensors that passed the CrowdQC+ tests and with white colour, the sensors that were eliminated are depicted. The parameters described in every step of the data cleaning tool can be modified further to give priority either to maintaining the number of sensors or focusing on the measurement precision.

5. Results and Discussion

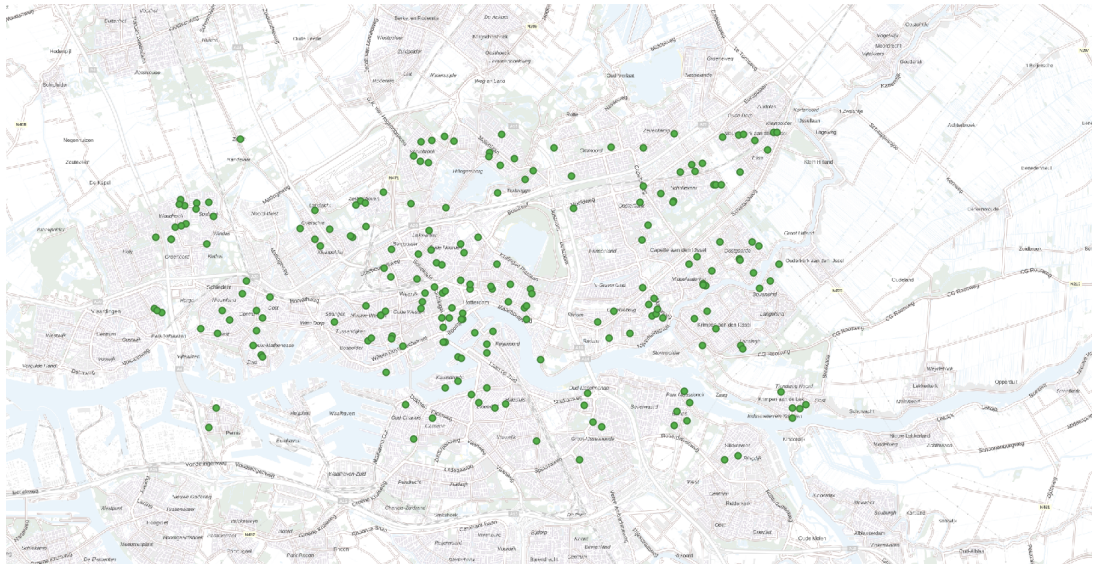


Figure 5.1.: Real-time sensor locations in Rotterdam (September 15, 2023)

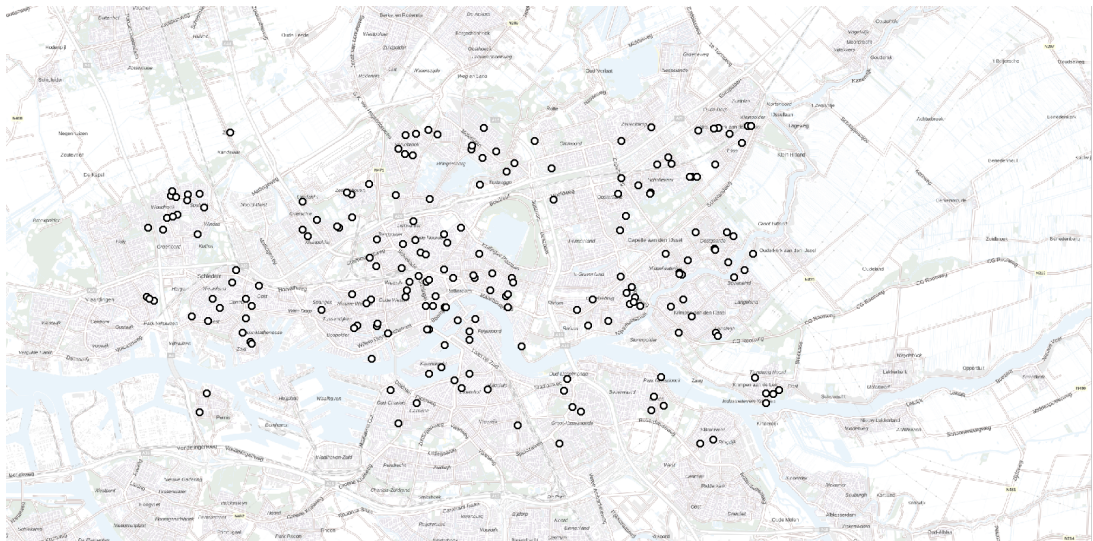


Figure 5.2.: Raw historical sensor locations in Rotterdam (May 2023)

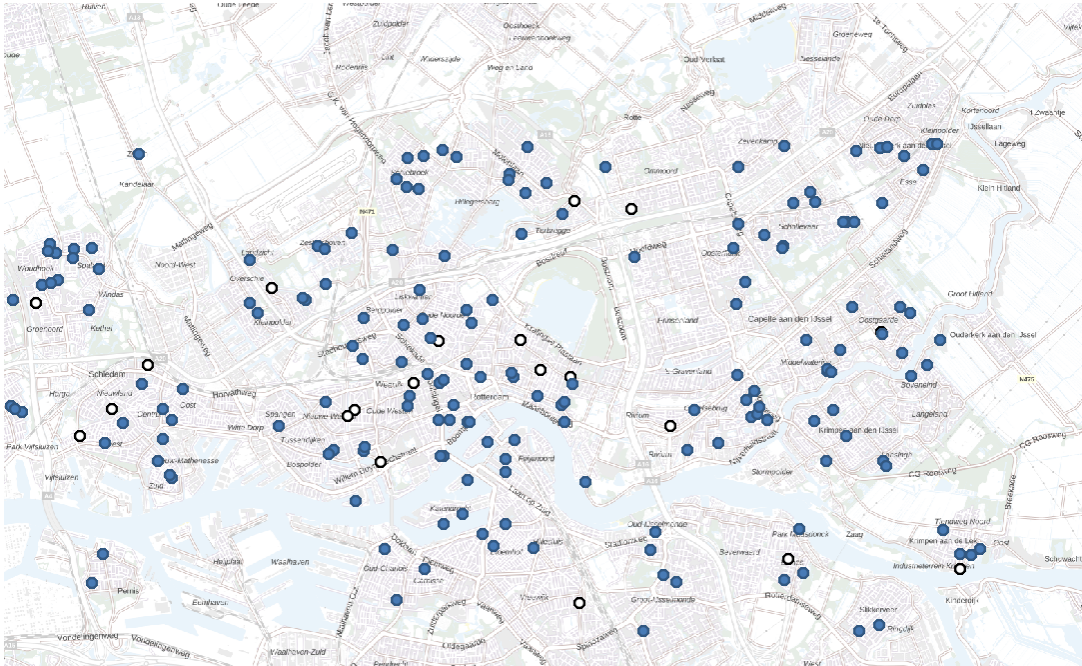


Figure 5.3.: Eliminated historical sensors (white dots) and the accepted sensors after the cleaning process (blue dots) for May 2023

5.3. Data interpolation

The interpolation method that was chosen is the nearest neighbor, mainly due to its extrapolating properties that provide a raster with specific extents, no matter the sensor availability. Additionally, it can be seen from the interpolation results that the smallest difference between the highest and lowest temperature values was given by the linear interpolation (15 degrees Celsius). In all of the three methods, it is obvious that there is one sensor that might be not set up properly, as it shows very low measurement values compared to the other sensors. This behaviour is expected because when working with real-time data, the observations are raw as the data cannot go through a cleaning process. For the real-time data, and specifically for the 15th of September at 17:00, the different interpolation results can be seen in the Figure 5.4.

The same interpolation methods were tested for the clean, historical sensor data. The results of the different interpolation methods can be seen in Figure 5.5. In this case the smallest difference between the highest and lowest recorded temperatures is given by the nearest neighbor and linear interpolation (2 degrees Celsius and 2.25 degrees Celsius).

5.4. Publishing the data

First this section will focus on the real time data and the pygeoapi process and on the second half of the section, the historical data pipeline will be discussed.

5. Results and Discussion

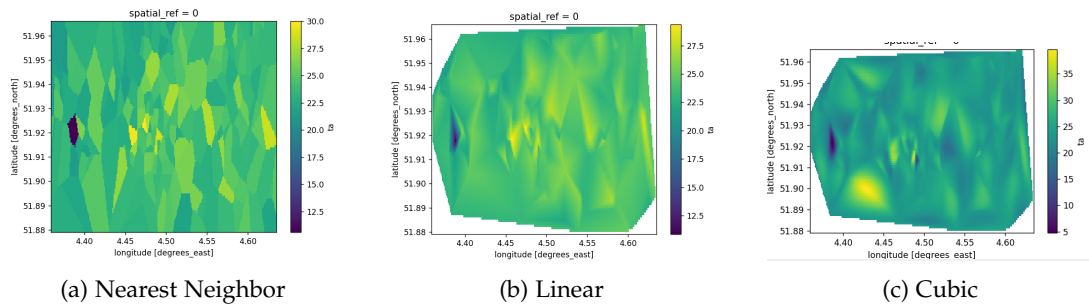


Figure 5.4.: Interpolation results using SciPy for raw real-time measurements.

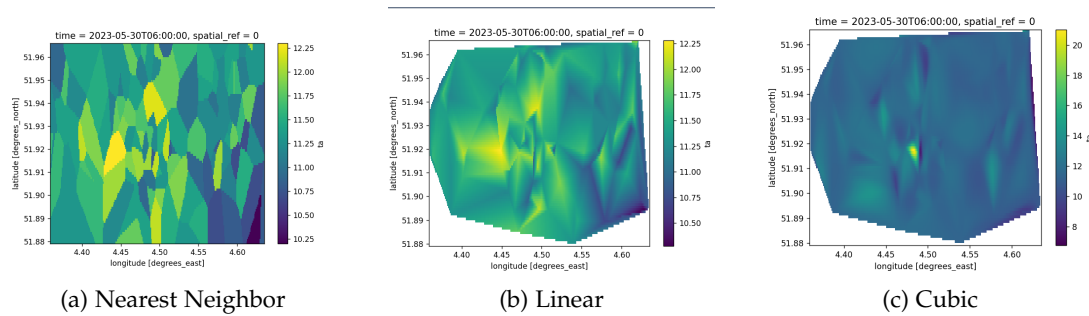


Figure 5.5.: Interpolation results using SciPy.

As it was described in the implementation section, the result of the OGC API Process required many additional steps in order to be available to be loaded into Cesium or to be retrieved by any other application. The base64 result needed to go through a transformation method to have a geospatial format (GeoTiff). This is a limitation that needs to be considered when using the API Processes as only the core standards are developed by now. Additionally, the API Processes could/will support process chaining that would make the whole procedure of calling the process and uploading the result as another OGC API standard, for example as OGC API Coverages, a lot more seamless and modern, potentially eliminating the need to utilize node-red to publish the interpolation result in an OGC API implementation server.

Regarding the historical, pre-processed data, because the time was a major aspect, a proper spatio-temporal format needed to be utilized. Pygeoapi offers the option to upload data in a NetCDF format, but an issue occurs when these data need to be accessed by CesiumJS to be visualized. To overcome this limitation, GeoServer was used with the NetCDF plugin that allows the users to upload spatio-temporal data and then be requested in various formats, such as Web Map Services.

5.5. Visualizing the interpolation results in CesiumJS

A sample of the visualization is shown in Figure 5.6 for different time stamps within a day. It can be seen that as the time passes the values on the grid are changing and during afternoon

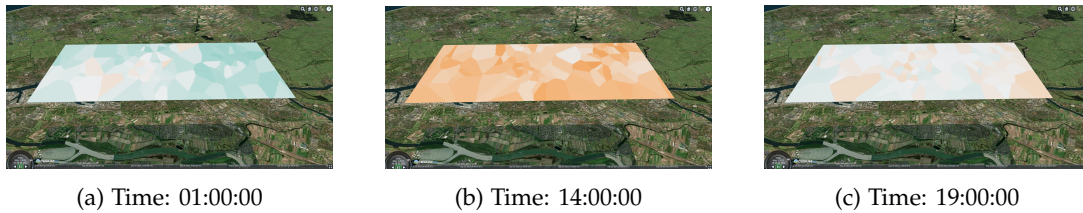


Figure 5.6.: Cesium visualization of interpolated data at different time stamps



Figure 5.7.: 3D Tiles Cesium Visualization)

time (at 14:00:00) the temperature is significantly higher than the other time stamps. The web application is developed in a way, where the raster changes in hourly intervals. This is the implementation for the historical data, while in the same script, the real-time visualization also takes place as soon as the time is set to the current time.

5.6. 3D city model

The integration of the 3D BAG city model to Cesium was an extremely straightforward process that was completed through specific steps. The first concern is to transform the 3D city model into a format that is accepted by Cesium ion, and then the tiling process is completed by Cesium ion. For the current research, the addition of 3D Tiles was not entirely necessary as the area of interest is relatively small, but it is important to highlight the trend towards 3D representations of the urban environment. An overview of the 3D Tiles as loaded in CesiumJS can be seen in Figure 5.7.

The final results of visualization of the application can be seen in a static way in Figures 5.9 and 5.9. It needs to be noted that as the application is deployed in a web browser, the interpolated raster changes dynamically with each timestamp.

5. Results and Discussion

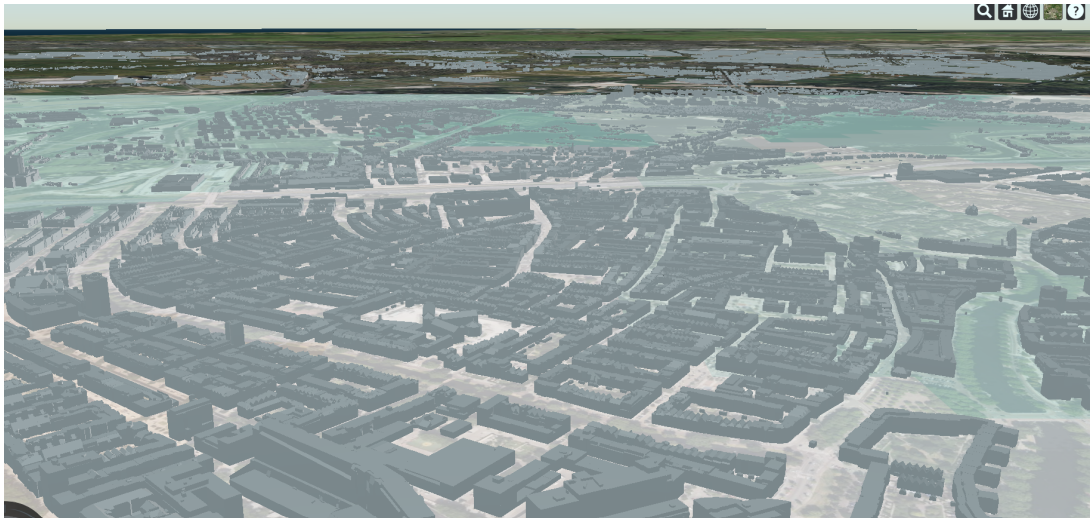


Figure 5.8.: Cesium visualization

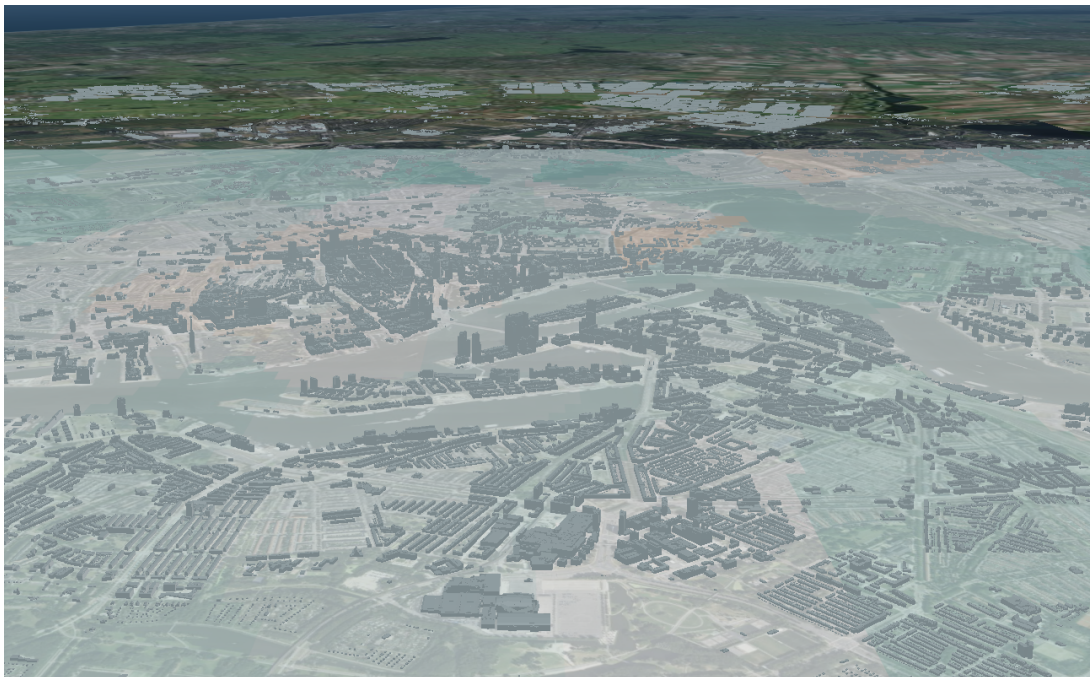


Figure 5.9.: Cesium visualization

6. Conclusions, limitations and future work

In this section the research and its results are concluded, through the scope of main objectives of the thesis and the research questions that were presented in the beginning of this document. This thesis was aiming towards creating a web mapping tool that would integrate efficiently and in an automated way, sensor data to provide insight to the dynamic patterns of the city. Special attention is given to an effort to overcome the documented challenges in the field, while utilizing the latest available applications and following the global trends.

The central challenge facing urban digital twins is the need for seamless interoperability, where these systems must effectively communicate with one another and facilitate the exchange and collaboration of data. To address this challenge in the scope of the present thesis, internationally recognized standards were incorporated, and more specifically the OGC standards. The OGC standards, a well-established and comprehensive set of guidelines, play a pivotal role in enabling data acquisition, publication, and access. Consequently, this research has established a complete data pipeline that encompasses data collection, storage, manipulation, and publication, with the consistent application of these standards at each stage of the process. Notably, the recent introduction of the second generation of OGC standards provides developers and users with increased flexibility, reflecting the evolving landscape of geospatial technologies. In summary, the emphasis on standardization remains at the core of the decision-making process in this research, ensuring a solid foundation for the development and deployment of urban digital twins.

Besides the standardization this research also focuses on sensor data acquisition and especially on data that can provide a more concrete and detailed picture of the city's dynamics and changes on a smaller scale. In order to address this ambition, crowd-source environmental data and more specifically outdoor temperature data is harvested from an open-access API. The crowd-source data is an alternative to the expensive and sparse environmental sensors that allow an abundance of observations to the researchers. One of the main reasons Netatmo is used is the high density of its network and the high frequency of the data update in combination to the access in historical data.

Moving on from the general overview of the scope of the thesis, it is beneficial to address the sub-questions that complement the main research question and provide an in depth exploration of the components of the thesis. Firstly, the question on *How can international standards be employed to acquire, store, and manipulate real-time and historical sensor data, ensuring data consistency and interoperability?* is approached through the SensorThings API standard. SensorThings API provide a complete and highly adjustable framework that handles both real-time and historical data. By being able to define the different "Datastreams" of each sensor, the Netatmo data is stored in a way that are easily accessible not only for the current application but also for users and developers that will use this data in future projects. By storing separately the real-time raw data, the raw historical data and the clean, pre-processed historical data (but all within the same SensorThings API server), the advanced filtering techniques that are provided by the standard can refine the sensor data in an effective and user-friendly way. To sum up, the Sensor Things API standard is a fully developed, mature

6. Conclusions, limitations and future work

and explored in depth standard that provides endless possibilities for storing and handling sensor data. This standard is the first and most important step on which the entire research is built upon and based on the current implementation, no limitations in its use or capabilities were detected in the context of this thesis.

Following, the second sub-question on *How can environmental sensor data be effectively processed to provide detailed spatio-temporal information for further analysis?* will be explored. This question consists of many components that affect not only the selected algorithms for the interpolation but also the original data quality. Despite the fact that crowd-source data provide many observations, as it is shown in the results of the research, they sensors are very prone to errors because of their placement and their availability number fluctuates as not all sensors are available at all times. These limitations, make the real-time sensor data, low quality and unreliable. On the other hand, for the historical data there is the possibility to pre-process the data and eliminate any outliers that affect the resulted product. In this thesis, the CrowdQC+ tool is used, because it offers the advantage of only using the crowd-source data as input and not needing external observations to clean the data effectively. The downside of the tool is that it needs observations of a month in order to provide high-quality results and with this condition it cannot be used to clean the real-time data, where a single measurement of each sensor is taken. Taking into consideration the nature of each application, though, it could possibly be argued that in cases where seasonal trends need to be identified, for example for urban planning projects, the real-time data do not provide any significant insight to the decision makers. The other aspect of the second sub-question is the detailed spatio-temporal information that are provided through the interpolation. The nearest neighbor algorithm is used for this research as it provides, based on the literature review, adequate results in terms of accuracy. The interpolation is a process that creates observations both in the spatial as well as the temporal realm of the data and therefore it contributes to a more detailed depiction of the urban space, in comparison to the scattered sensor observations.

Finally the third sub-question on *How can environmental sensor data be made available and accessed in a standardized manner to enable effective utilization in geospatial applications?* will be scrutinized. Also, in this part of the research, different paths were used for the real-time and for the historical data, mainly due to the obstacles that occurred while trying to incorporate to the pipeline the latest OGC API standards. For the real-time data the interpolation is conducted through an OGC API process that is hosted in the pygeoapi server. The processes is executed and the result is returned to the local machine and then is published to another server, GeoServer, through HTTP POST requests. This implementation, even though it integrates the newly developed OGC API Processes standard, it fails to publish directly to another OGC API standard format, for example OGC API Coverages or WCS. Additionally, for the historical data, the interpolation is performed locally because the data input are the hourly observations of a whole month and the result is a NetCDF file that needs to be stored locally, zipped and then published. Since, only the core part of the OGC API Process standard is published and process chaining is not possible currently, the option of minimizing the use of node-red and JavaScript in handling the interpolation and the interpolation results needs to be revisited. Moreover, while exploring the possibilities of the standards, it is important to keep in mind the applications that will use the data and their compliance with OGC API standards. In this research, CesiumJS is used to visualize the interpolation result in a 3D globe, while integrating a 3D city model to provide a more complete and extensive visualization of the world. By using CesiumJS, there is the major advantage of tiling large datasets in an extremely effective way, but the capabilities of CesiumJS on data integration that is not focused on visualization but also on data analysis, are limited. This became ev-

ident, when the interpolation data could be added in the web portal only as a WMS. It needs to be mentioned that even though GeoServer gives the possibility to serve the data in a WCS format that can be used for further geospatial analysis, Cesium does not support this format. All in all, there are difficulties on integrating the newer generation of OGC standards with the already established geospatial applications, which are currently starting to integrate and adapt to these new standards. Also, the second generation of OGC standards is not mature enough yet to be used seamlessly with the already established standards or with other components of the OGC API family standards.

An effort was made to select a larger area that would give a better understanding of how the temperature changes between the urban and non-urban areas but due to the fact that the whole application runs locally with limited CPU and GPU resources, issues were occurring and this direction was never taken. Therefore, this research should be viewed as a case study that tries to integrate the new generation of OGC API Standards into a pipeline that includes different processes, data formats and specifications.

6.1. Future work

Building upon the conclusions drawn from this research, there are several promising opportunities for future work that can contribute to the advancement of urban digital twin systems, particularly in the context of sensor data integration and international standards.

Enhanced Sensor Data Acquisition and Quality Assurance: The research can be extended to include a broader spectrum of sensor types, beyond environmental data, such as air quality, traffic, noise, and more. Additionally, combining data from multiple sensor types will provide a more comprehensive understanding of urban dynamics, which can lead to more informed decision-making. Moreover, there could be an opportunity to research algorithms for real-time data quality assessment and outlier detection, particularly for crowd-sourced data. These efforts can enhance the reliability of real-time sensor data, making it more suitable for immediate decision-making in urban planning and management. Additionally, advanced data cleaning tools should be researched, especially for real-time data, which will further enhance the quality of sensor data.

Spatio-Temporal Data Analysis: Advancing the research in spatio-temporal data analysis, it is crucial to explore more sophisticated interpolation techniques beyond the nearest-neighbor algorithms. This might involve the investigation of machine learning methods, geostatistics, or data assimilation to enhance the accuracy of interpolated data. Furthermore, the development of models that capture seasonal trends in environmental data and enable scenario analysis will be invaluable for long-term urban planning and resilience against climate change.

Standardization and Data Accessibility: Continuing the research in the domain of standardization, the integration of emerging OGC API standards with existing geospatial applications is an area of great importance. Research should focus on bridging the gap between the second generation of OGC standards and established standards, making data exchange more seamless and efficient. Moreover, the exploration of methods for directly publishing sensor data in standard formats (such as OGC API Coverages or WCS) will eliminate the need for intermediate steps in the data publication process. Additionally, it is essential to

6. Conclusions, limitations and future work

develop user-centric tools and interfaces that make it easy for both developers and non-technical users to access standardized sensor data, ultimately enhancing the usability of urban digital twins.

Data Visualization and Analysis Tools: Further advancements in data visualization and analysis tools can be explored. This includes expanding the capabilities of 3D data visualization tools like CesiumJS to include advanced data analysis features, enabling users to perform in-depth data analytics directly within the 3D urban models. Integration with popular Geographic Information Systems (GIS) software should also be investigated, facilitating data analysis, decision-making, and urban planning processes. As an extra step, the raw observation data could be visualized in the platform in a dashboard configuration to provide more insight into each sensor data, individually.

Enhancing Infrastructure and Operational Efficiency Additionally, it would be highly recommended to establish and run the pipeline and the required dependencies on a server to ensure the reception of real-time data. Another limitation of running all the processes of this research locally is the fact that the computer needs to be activated in order to deploy and run node-red. As a result, when the pipeline is not active, the real-time data acquisition fails, and the dataset is not consistent. Addressing these infrastructure and operational challenges will be crucial for ensuring the reliability and continuity of real-time data acquisition and processing in urban digital twin systems.

A. Reproducibility self-assessment

A.1. Marks for each of the criteria

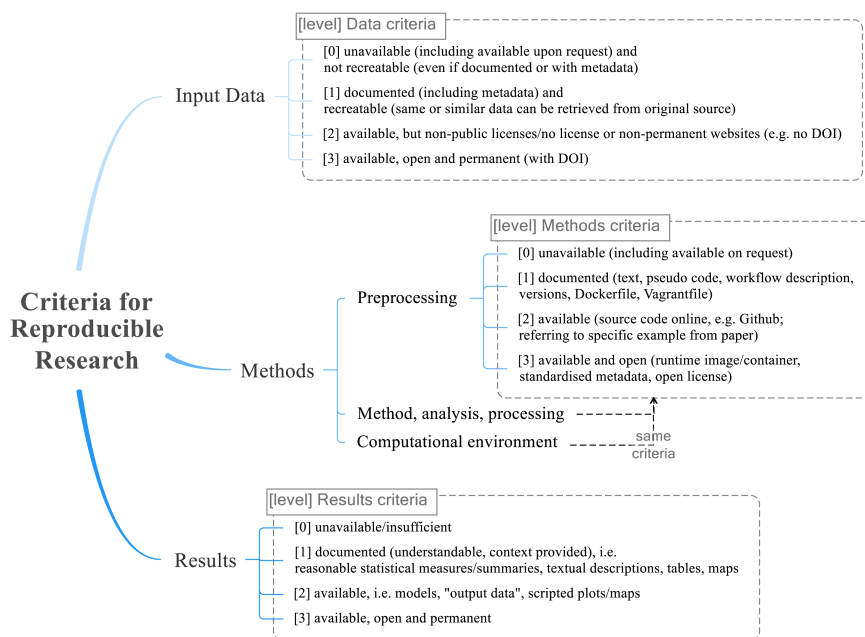


Figure A.1.: Reproducibility criteria to be assessed.

Category	Grade
Input Data	3
Preprocessing	3
Methods	2
Computational Environment	1
Results	0

Table A.1.: Evaluation of reproducibility criteria

A.2. Self-reflection

Input Data: The sensor data that was used in the thesis are available through an open-access Netatmo API. Because of the nature of crowd-source data and the sensors' fluctuating

A. Reproducibility self-assessment

availability and connectivity, the data input could potentially differ from the input data of the thesis.

Methods: The implementation of the research is based on open-source applications and tools. The server (Frost-Server, pygeoapi, node-red) instances that need to be deployed are also open access along with the data cleaning algorithm, as well as the python library that is used for the data interpolation and the CesiumJS that is used for the visualization. As far as the deployment of the whole architecture, directions for the different components can be found in Github. Cesium ion is the only aspect of the project that is not open-source software but it can be used as a free tier, by individual users and for low-traffic application.

Results The results of the thesis are spatio-temporal and visualized through a web-application according to specific time frames. Sequential, the complete and detailed documentation or description of the results is not feasible in a report. In Github there is a full documentation on how to deploy the system but there is no actively running instance of the project and therefore not for the results either.

Bibliography

- [1] Benjamin, K., Luo, Z., and Wang, X. (2021). Crowdsourcing urban air temperature data for estimating urban heat island and building heating/cooling load in london. *Energies*, 14:5208.
- [2] Bhattacharya, D. and Painho, M. (2016). DESIGN FOR CONNECTING SPATIAL DATA INFRASTRUCTURES WITH SENSOR WEB (SENSDI). *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-4:33–39.
- [3] Bibri, S., Alahi, A., Sharifi, A., and Krogstie, J. (2023). environmentally sustainable smart cities and their converging ai, iot, and big data technologies and solutions: an integrated approach to an extensive literature review. *Energy Informatics*, 6.
- [4] Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3D City Models: State of the Art review. *ISPRS international journal of geo-information*, 4(4):2842–2889.
- [5] Bloomfield, H., Brayshaw, D., Troccoli, A., Goodess, C., De Felice, M., Dubus, L., Bett, P., and Saint-Drenan, Y.-M. (2021). Quantifying the sensitivity of european power systems to energy scenarios and climate change projections. *Renewable Energy*, 164:1062–1075.
- [6] Boje, C., Guerriero, A., and Rezgui, Y. (2020). Towards a semantic construction digital twin: directions for future research. *Automation in Construction*, 114:103179.
- [7] Caprari, G. (2022). Digital Twin for urban planning in the Green Deal Era: a state of the art and future perspectives. *Sustainability*, 14(10):6263.
- [Catalogue Service - OGC] Catalogue Service - OGC. Catalogue Service - Open Geospatial Consortium.
- [9] Catherine, S. and Sue, G. (2006). Applied climatology: urban climate. *Progress in Physical Geography: Earth and Environment*, 30(2):270–279.
- [10] Centraal Bureau voor de Statistiek (June 29, 2023). Households; persons by gender, age and region, january 1.
- [11] Cesium - 3D Tiles (2020).
- [12] Cesium Supporting State-Sized Digital Twins (2022). Cesium supporting state-sized digital twins.
- [13] Cesium. Tiler Data Types and Formats (n.d.).
- [14] Chapman, L., Bell, C., and Bell, S. (2017). Can the crowdsourcing data paradigm take atmospheric science to a new level? a case study of the urban heat island of london quantified using netatmo weather stations. *International Journal of Climatology*, 37(9):3597–3605.

Bibliography

- [15] Climate of the Netherlands- KNMI (n.d.). Cesium supporting state-sized digital twins.
- [16] Consortium, O. G. (2023). 3D IoT platform for smart Cities Pilot - Open Geospatial Consortium.
- [17] Dembski, F., Wössner, U., Letzgus, M., Ruddat, M., and Yamu, C. (2020). Urban digital twins for smart cities and citizens: the case study of herrenberg, germany. *Sustainability*, 12:2307.
- [18] Digital Twin Victoria Program (n.d.). Digital twin victoria program.
- [19] Fenner, D., Bechtel, B., Demuzere, M., Kittner, J., and Meier, F. (2021). Crowdqc+—a quality-control for crowdsourced air-temperature observations enabling world-wide urban climate applications. *Frontiers in Environmental Science*, 9.
- [20] Ferré-Bigorra, J., Casals, M., and Gangolells, M. (2022). The adoption of urban digital twins. *Cities*, 131:103905.
- [21] FROST - FRAunhofer Opensource SensorThings Server. (2016).
- [22] GeoServer (n.d.).
- [23] Grassmann, T., Napoly, A., Meier, F., and Fenner, D. (2018). Quality control for crowd-sourced data from CWS. *Frontiers in Earth Science*.
- [24] Grothe, M., Vanden Broecke, J., Carton, L., Volten, H., and Kieboom, R. (2016). Smart Emission - Building a Spatial Data Infrastructure for an Environmental Citizen Sensor Network. *e*, pages 1–7.
- [25] Guccione, P., Appice, A., Ciampi, A., and Malerba, D. (2012). Trend cluster based kriging interpolation in sensor data networks. In Atzmueller, M., Chin, A., Helic, D., and Hotho, A., editors, *Modeling and Mining Ubiquitous Social Media*, pages 118–137, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [26] Hobona, G., Simmons, S., Pau, J. M., and Jacovella-St-Louis, J. (2023). OGC API Standards for the next generation of web mapping. *Abstracts of the ICA*, 6:1–2.
- [27] Hoyer, S. and Hamman, J. (2017). Xarray: n-d labeled arrays and datasets in python. *Journal of Open Research Software*, 5:10.
- [28] Hristov, P., Petrova-Antonova, D., Ilieva, S., and Rizov, R. (2022). Enabling city digital twins through urban living labs. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B1-2022:151–156.
- [29] Härmäläinen, M. (2021). Urban development with dynamic digital twins in helsinki city. *IET Smart Cities*, 3:201–210.
- [30] Initiative, O. (2021). Openapi specification v3.1.0.
- [31] Kong, L., Mu, X., Hu, G., and Zhang, Z. (2022). The application of resilience theory in urban development: a literature review. *Environmental Science and Pollution Research*, 29:49651–49671.
- [32] Kulawiak, M., Kulawiak, M., and Łubniewski, Z. (2019). Integration, processing and dissemination of lidar data in a 3d web-gis. *ISPRS International Journal of Geo-Information*, 8:144.

- [33] Ledoux, H., Otori, K., Kumar, K., Dukai, B., Labetski, A., and Vitalis, S. (2019). cityjson: a compact and easy-to-use encoding of the citygml data model. *Open Geospatial data Software and Standards*, 4.
- [34] Liang, S., Khalafbeigi, T., and van der Schaaf, H. (2021). *OGC SensorThings API Part 1: Sensing Version 1.1*.
- [35] Linden, L. v. d., Hogan, P., Maronga, B., Hagemann, R., and Bechtel, B. (2023). Crowdsourcing air temperature data for the evaluation of the urban microscale model palm—a case study in central europe.
- [36] Meier, F., Fenner, D., Grassmann, T., and Otto, M. (2017). Crowdsourcing air temperature from citizen weather stations for urban climate research. *Urban Climate*, 19:170–191.
- [37] Meta, I., Serra-Burriel, F., Carrasco-Jiménez, J. C., Cucchiatti, F. M., Diví-Cuesta, C., Calatrava, C. G., García, D., Graells-Garrido, E., Navarro, G., Làzaro, Q., Reyes, P., Navarro-Mateu, D., Julian, A. G., and Martínez, I. E. (2021). The camp nou stadium as a testbed for city physiology: a modular framework for urban digital twins. *Complexity*, 2021:1–15.
- [38] Moyne, J., Qamsane, Y., Balta, E. C., Kovalenko, I., Faris, J., Barton, K., and Tilbury, D. M. (2020). A requirements driven digital twin framework: specification and opportunities. *IEEE Access*, 8:107781–107801.
- [39] Nakamura, R. and Mahrt, L. (2005). Air temperature measurement errors in naturally ventilated radiation shields. *Journal of Atmospheric and Oceanic Technology*, 22:1046–1058.
- [40] Netatmo Weather Data API Documentation (n.d.). Netatmo weather data api documentation.
- [41] Newman, P., Beatley, T., and Boyer, H. (2009). Resilient cities: responding to peak oil and climate change. *Australian Planner*, 46:59–59.
- [42] Niu, H. and Silva, E. A. (2020). Crowdsourced data mining for urban activity: review of data sources, applications, and methods. *Journal of Urban Planning and Development*, 146.
- [43] NRF Virtual Singapore (2021). Nrf virtual singapore.
- [44] OGC - 3D Tiles Specification 1.0 (2019). 3d tiles specification 1.0.
- [OGC API Common] OGC API Common. Standards - Open Geospatial Consortium.
- [OGC API Coverages] OGC API Coverages. Standards - Open Geospatial Consortium.
- [OGC API Features] OGC API Features. Standards - Open Geospatial Consortium.
- [OGC API Maps] OGC API Maps. Standards - Open Geospatial Consortium.
- [OGC API Processes] OGC API Processes. Standards - Open Geospatial Consortium.
- [OGC API Tiles] OGC API Tiles. Standards - Open Geospatial Consortium.
- [51] Potgieter, J., Nazarian, N., Lipson, M., Hart, M., Ulpiani, G., Morrison, W., and Benjamin, K. (2021). Combining High-Resolution land use data with crowdsourced air temperature to investigate Intra-Urban microclimate. *Frontiers in Environmental Science*, 9.

Bibliography

- [52] Puche, M., Vavassori, A., and Brovelli, M. A. (2023). Insights into the Effect of Urban Morphology and Land Cover on Land Surface and Air Temperatures in the Metropolitan City of Milan (Italy) Using Satellite Imagery and In Situ Measurements. *Remote Sensing*, 15(3):733.
- [53] Pygeoapi (n.d.).
- [54] Rousseeuw, P. J. and Croux, C. (1993). Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424):1273–1283.
- [55] Santhanavanich, T. and Coors, V. (2020). Citythings: an integration of the dynamic sensor data to the 3d city model. *Environment and Planning B: Urban Analytics and City Science*, 48:417–432.
- [56] Scipy Documentation: Scattered Data Interpolation (griddata) (n.d.).
- [57] Semeraro, C., Lezoche, M., Panetto, H., and Dassisti, M. (2021). Digital twin paradigm: a systematic literature review. *Computers in Industry*, 130:103469.
- [58] Shahat, E., Hyun, C. T., and Yeom, C. (2021). City digital twin potentials: a review and research agenda. *Sustainability*, 13:3386.
- [59] Simonis, I. (2019). OGC Standardization: From Early Ideas to Adopted Standards. *IEEE International Geoscience and Remote Sensing Symposium*.
- [60] Singh, S. P., Jain, K., and Mandla, V. R. (2014). Image based 3d city modeling : Comparative study. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5:537–546.
- [61] Spaans, M. and Waterhout, B. (2017). Building up resilience in cities worldwide – rotterdam as participant in the 100 resilient cities programme. *Cities*, 61:109–116.
- [Standards - OGC] Standards - OGC. Standards - Open Geospatial Consortium.
- [Styled Layer Descriptor - OGC] Styled Layer Descriptor - OGC.
- [64] Takahiro, O. and Kunihiro, K. (n.d.). Smart city takamatsu solutions for disaster prevention/management. City of Takamatsu.
- [65] Turchi, S., Bianchi, L., Paganelli, F., Pirri, F., and Giuli, D. (2013). Towards a web of sensors built with linked data and rest. *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*.
- [66] Turney, S. (2023). Pearson Correlation Coefficient (r) — Guide 038; Examples.
- [67] UN-Habitat (2022). World cities report 2022.
- [Web Coverage Service - OGC] Web Coverage Service - OGC. Web Coverage Service - Open Geospatial Consortium.
- [Web Feature Service - OGC] Web Feature Service - OGC. Web Feature Service - Open Geospatial Consortium.
- [Web Map Service - OGC] Web Map Service - OGC. Web Map Service - Open Geospatial Consortium.

- [Web Processing Service - OGC] Web Processing Service - OGC. Web Processing Service - Open Geospatial Consortium.
- [72] Weil, C., Bibri, S. E., Longchamp, R., Golay, F., and Alahi, A. (2023). Urban digital twin challenges: a systematic review and perspectives for sustainable smart cities. *Sustainable Cities and Society*, 99:104862.
- [73] Yin, N. and Cai, M. (2022). urban road landscape design and digital twin simulation modeling analysis. *Discrete Dynamics in Nature and Society*, 2022:1–10.
- [74] Zhao, D., Li, X., Wang, X., Shen, X., and Gao, W. (2022). applying digital twins to research the relationship between urban expansion and vegetation coverage: a case study of natural preserve. *Frontiers in Plant Science*, 13.

Colophon

This document was typeset using \LaTeX , using the KOMA-Script class `scrbook`. The main font is Palatino.

