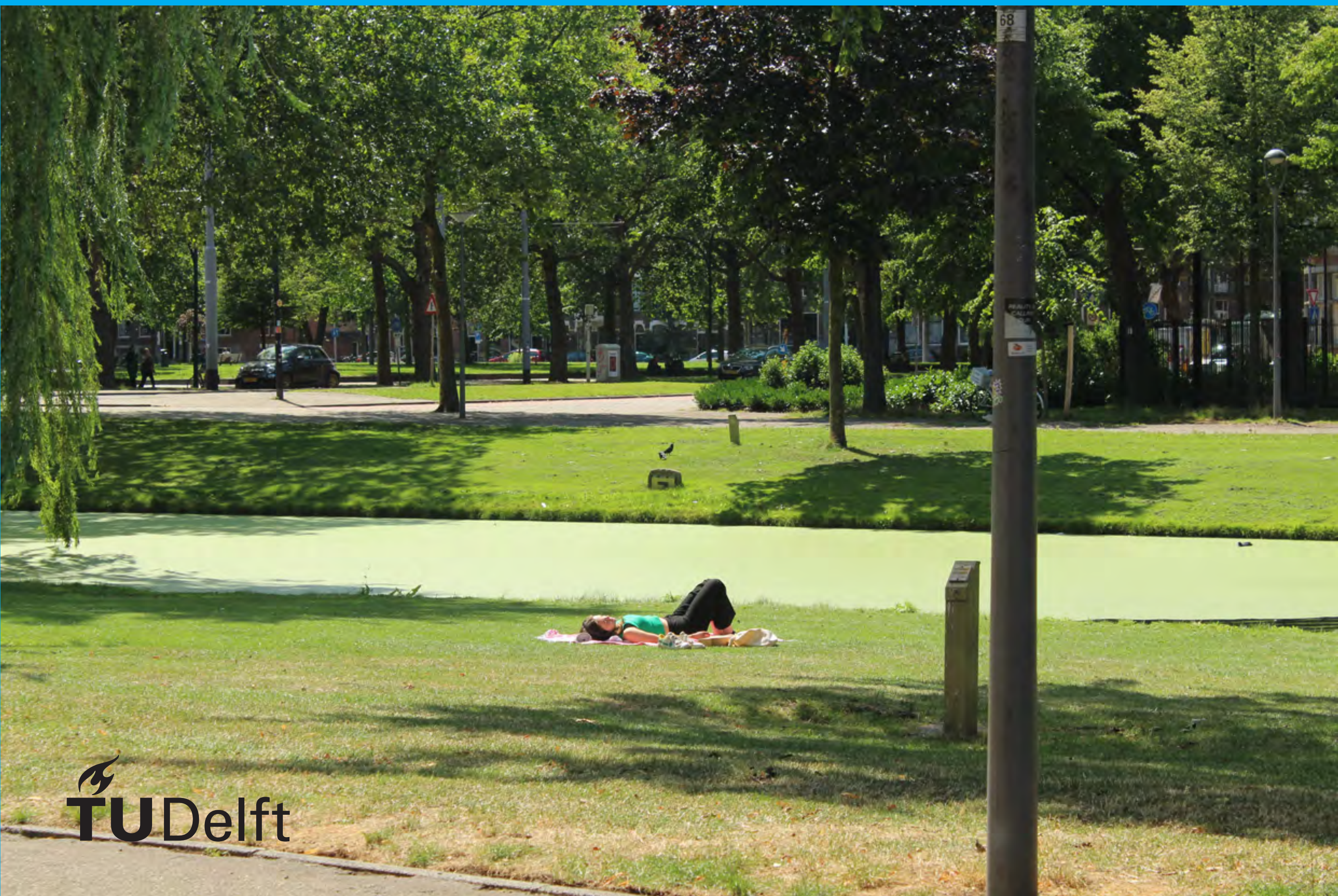
From thermal comfort to heat mitigation action

A reproducable QGIS plugin for calculating the physiological equivalent temperature in Dutch cities

Marieke van Esch

MSc Thesis Geomatics 2024



TUDelft

From thermal comfort to heat mitigation action

A reproducable QGIS plugin for calculating the physiological equivalent temperature in Dutch cities

by

Marieke van Esch

to obtain the degree of Master of Science of Geomatics and Urbanism
at the Delft University of Technology,
to be defended publicly on Wednesday April 17, 2024 at 12:45 PM.

Student number:	4545508	
Project duration:	February, 2023 – April 17, 2024	
Thesis committee:	Ir. E. Verbree,	TU Delft, 1st supervisor
	Dr.Ir. S. van der Spek,	TU Delft, 2nd supervisor
	Dr.Ir. S. Koopmans,	Wageningen University, co-reader
	Dr.Ir. S. Kahmedi,	Building Technology Department External Comitee

Abstract

In the summer of 2023 heatwaves became quite prominent in the south of Europe. The Netherlands Meteorological Institute predicts that heat waves will increase from 26 to a maximum of 47 days by 2050, affecting also the Netherlands in the future. The main research question was how to propose a strategy for a liveable environment by designing public spaces while mitigating heat stress for vulnerable target groups in the context of Bospolder Tussendijken in Rotterdam, the Netherlands. The research included a literature review, expert consultations, scenario planning, modeling of the urban environment and mapping techniques. The research on Bospolder Tussendijken aimed to assess its liveability in terms of climate, social conditions and policies. The climate part of the research focused on creating a reproducible heat stress assessment tool to identify high-risk areas in public spaces. Factors such as solar radiation, evaporation and wind affect heat stress in the city, and designers could influence these factors based on their level of intervention at the built environment scale. Social conditions are divided into spatial mobility and social mobility. The spatial mobility of fast traffic affected the thermal experience of public space and social mobility, especially walking. Finally, the application of the reproducible PET tool helped to identify the temporal vulnerability to heat stress. In addition, the accessibility of public spaces for vulnerable groups on a summer day was assessed despite the range restriction caused by heat stress, and this information was used to inform design strategies and evaluate the final design. The design guidelines focused on mitigating heat stress, improving walkability as spatial mobility and enhancing social mobility spaces for vulnerable groups. The research emphasized the importance of identifying heat stress in public spaces and the need for urgent action to maintain the quality of life in the future. The spatio-temporal heat stress tool introduced in this study brought a new dynamic layer to urban planning and could suggest maximum technical improvements to improve the public space network. The research also proposed a way to calculate the cumulative cost of experiencing the thermal accessibility of an area, which could open up discussions for health organizations to investigate the thermal endurance acceptability of different target groups. Ultimately, the research concluded that urban planning should priorities the network of interventions to be durable and readable for citizens to function in the urban environment, whilst not being the option to maximize heat mitigation.

Keywords: Physiological Equivalent Temperature, Thermal Accessibility, Liveability, heat mitigation

Acknowledgments

I would like to thank my supervisors Edward Verbree, Stefan van der Spek and Marjolein van Esch. Also the external supervisors Sytse Koopmans and Gert-Jan Steeneveld from Wageningen University. I would also like to thank external parties such as Niels van der Vaart for providing the links to Rotterdam. Martijn Meijers for his time in showing me how to use the API's of KNMI. Merel Scheltema as urban designer of the municipality of Rotterdam for sharing her knowledge about the Rotterdam heat plan and Andre de Wit as mobility expert of the municipality of Rotterdam. Laurens Versluis for sharing his knowledge about Witteveen and Bos. Most of all, I would like to thank my family for supporting me throughout the whole graduation year, also the graduate group I studied with. I would also like to thank Diego Sieglevulda for his guidance during the first quarter of the Master's orientation process.

Preface

The past summer has shown signs of changing climate variability. In Spain, people are already feeling the effects of heat at the beginning of spring, according to The Guardian [Guardian, 2023]. Due to carbon emissions over the past decades, the heat will continue to linger in the atmosphere. The government has warned people to take precautions due to drought and temperatures 7-11 degrees Celsius above the average for this time of year. They have also highlighted behavioral thermoregulation strategies to cope with the heat [Millyard et al., 2020]. If emissions continue at the current rate, heat events are likely to occur more frequently in the future, affecting not only the southern part of Europe but also other regions. It is important to take action in the built environment to address climate change, which requires a new approach to how we design our surroundings. Speculative design is necessary to sketch future scenarios with different stakeholders by creating scenarios and testing them to develop comprehensive designs [Dunne and Raby, 2013]. Climate modeling requires consideration of the complexity of meteorological and physical factors. The synergy of the social aspect of public space usage is a key driver for adapting to climate adaptation in the built environment. This report is part of the joint degree between the studies Geomatics and Urbanism, in which Geomatics form strategies for urban development. The title of the Geomatics report is: “From thermal comfort to heat mitigation action: Informed Strategies for Mitigating PET Heat Stress in Public Spaces for Vulnerable Groups – A Rotterdam Case Study”.

Contents

List of Figures

List of Tables

1
Introduction

This chapter introduces the topic of graduation research. Furthermore, the former research will be introduced and the research gap will be acknowledged. The proposal for this research is formulated by the research aim. The approach will summarize the main research questions. The approach summarizes the methodology which will set out the sub-research questions related to this topic. At the end of this chapter, the structure of the report is elaborated.

1.1. Health at risk

A heat wave is defined as a period of at least 5 consecutive summer days with a maximum temperature of 25.0 °C or higher, of which at least three days have a maximum temperature of 30.0 °C or higher, as measured at the meteorological weather station in De Bilt, the Netherlands. This phenomenon is expected to become more common as our emissions contribute to climate change. This is further explained in scenarios with high and low emissions (see Figure 64), which predict an increase in the number of summer days with temperatures above 25 degrees Celsius.

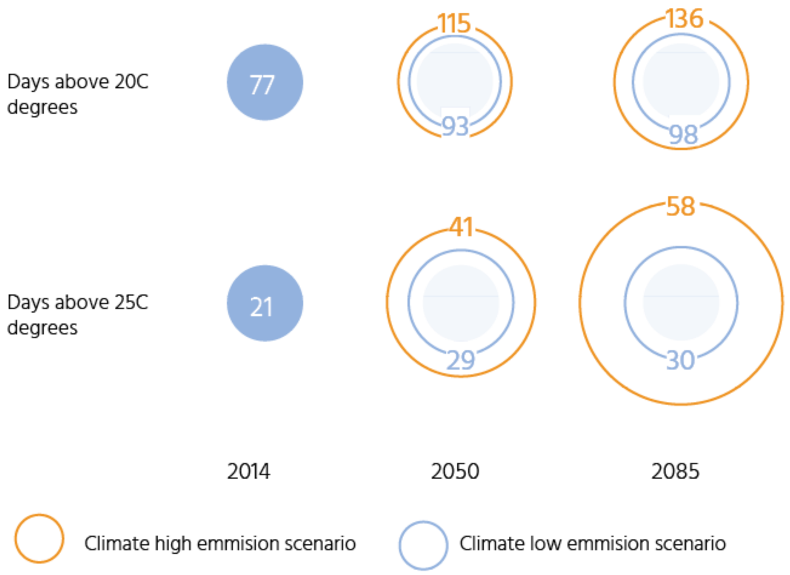


Figure 1.1: KNMI climate scenario's and predictability of amount of warm days and summer days adapted from [KNMI, 0000] [CAS, 2020]

People lack an adaptable response of the human body to a day of 25 Celsius degrees or above and this is an indicator of the mortality rates of people. This puts the health of citizens at risk. Physiological factors like heart rate will take some days to adapt to a warmer environment. Another aspect is that people can dress more for colder situations in contrast to hotter days [Lenzholzer, 2018] . These combinations lead to higher mortality rates with heat extreme like the occurrence of a summer day of 25 degrees [Daanen, 2023]. This is a serious issue now and in the future.

1.2. Heat mitigation research and action in the Netherlands

The Delta Plan on Spatial Adaptation [of Infrastructure and Waterboard, 2018] requires all municipal governments in the Netherlands to conduct a climate stress test addressing flood risk, heat stress, and drought. In 2019, Wageningen University created a report and code for RIVM, and in 2020, Witteveen en Bos released a PET-heat map for the Netherlands in cooperation with Wageningen University and Climate Adaptive Services (CAS) [CAS, 2020]. Although this web viewer is publicly available, it does not allow designers to assess spatial and temporal effects and make design decisions in specific locations. The "Hot Issues" conference at HVA in 2020 highlighted that municipalities are all in the process of reproducing this code themselves [HVA, 2020].

Furthermore, the National Heat Plan has been active since 2015 under the supervision of RIVM, with multiple stakeholders involved in heat mitigation matters in the Netherlands. These stakeholders are divided into state, private, and civil society parties, with a distinction made between primarily involved stakeholders and a wider audience of stakeholders. There are several collaboration formations identified using a power and interest matrix. The first formation involves health-considered parties, the second involves financial parties, and the third is particularly interested in the liveability component of society, including immediate residents, academia, urban planners, and municipalities. Collaborative sharing of knowledge and action based on the power-interest is crucial for taking care of heat stress mitigation.

Several sources are mapped out below and positioned on the "know, want, and taking action" framework Fig1.4 of the Delta Plan. It is evident that knowledge and action on this subject are fragmented and can be consolidated from the perspective of urban environment modelers towards the application of action-based urban design practitioners.

Figure 1.2

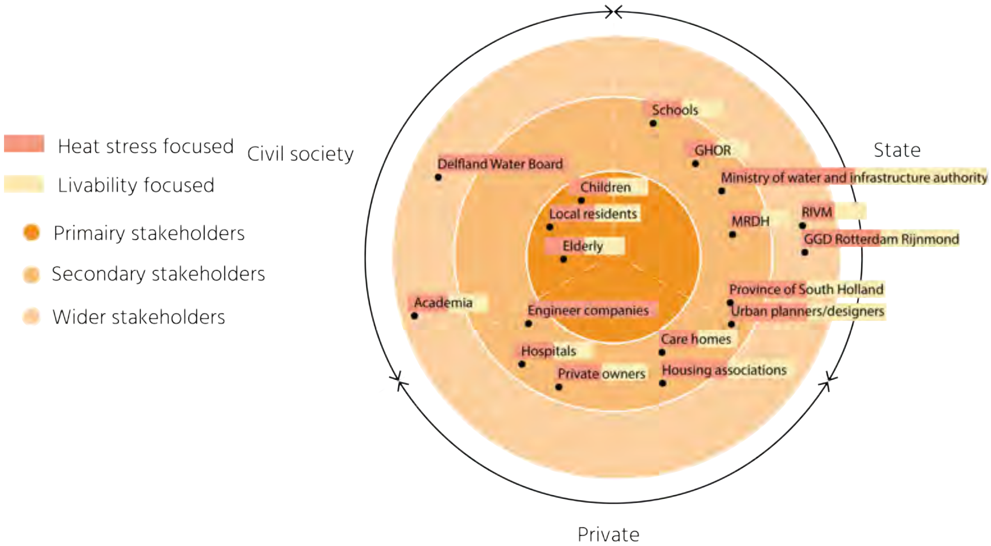


Figure 1.2: Stakeholder diagram. Adapted from [Hofman, 2022]

Figure 1.3

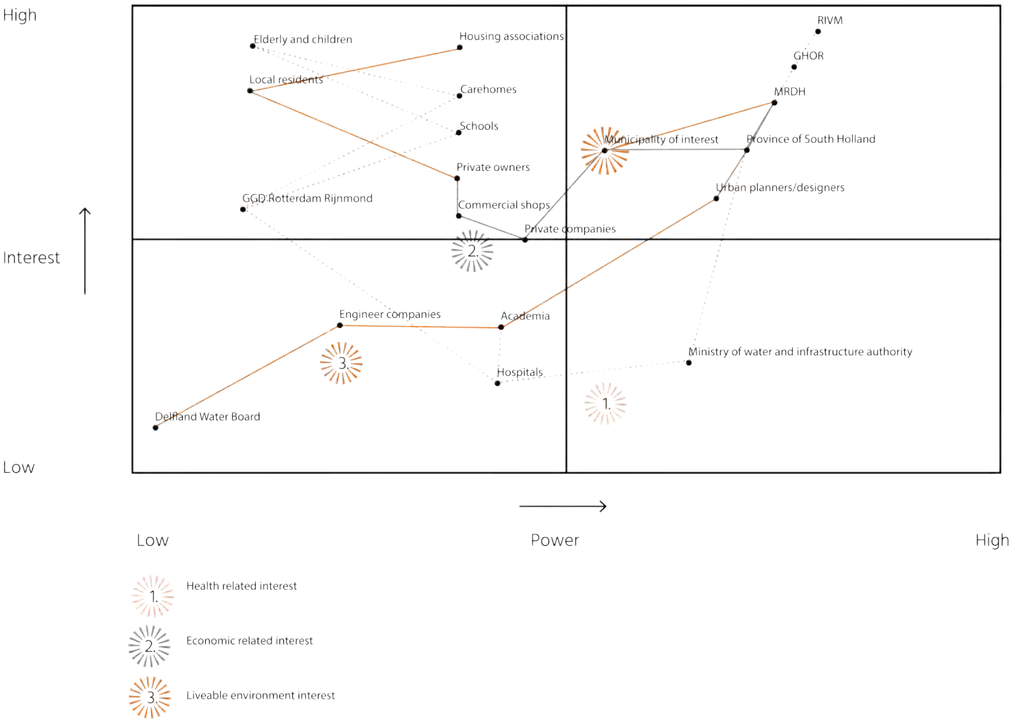


Figure 1.3: Stakeholder power interest matrix. Adapted from [Hofman, 2022] .

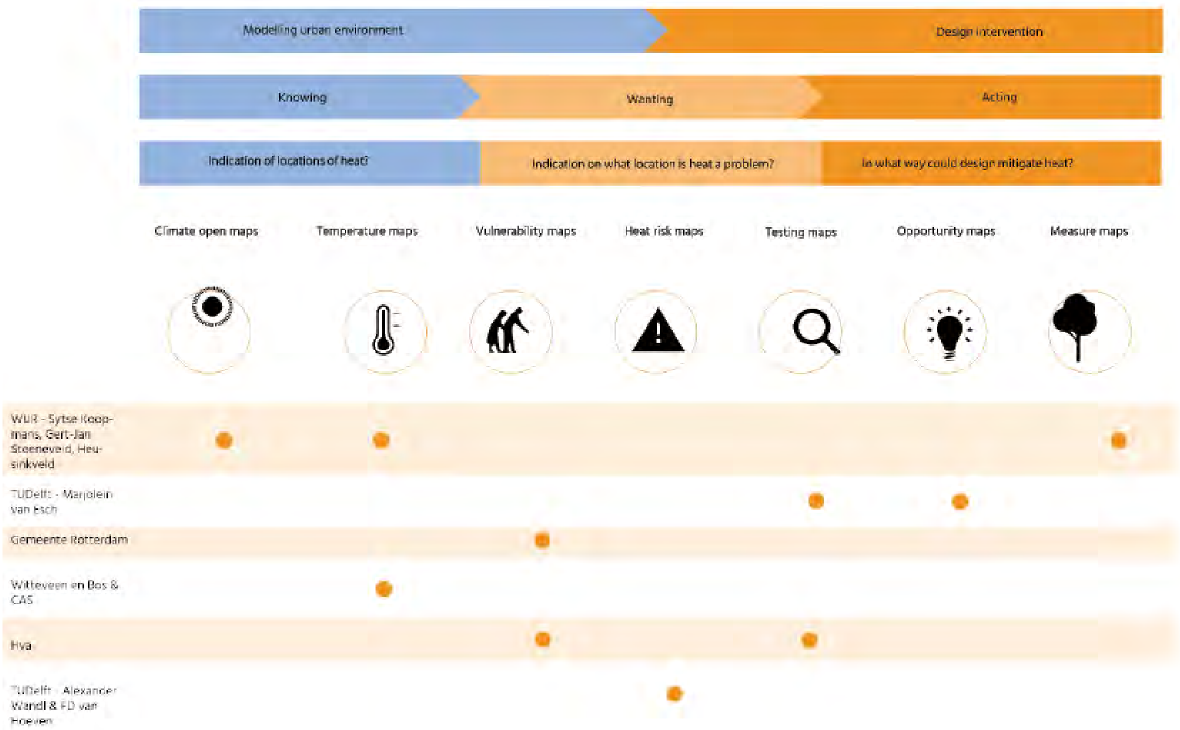


Figure 1.4: Placement of this research within the field of knowledge and action. Put in the framework of Deltaprogramme [Programme, 2018]

1.3. Research gap

Based on the orientation phase, which involved talking to various parties, two main research gaps have been identified. One is the lack of an interactive, open-access tool that helps discover knowledge for an action-based approach. The other gap is the absence of a developed strategy on how to target the most important public spaces for transformation.

1. Lack of one open platform with knowledge for multiple parties/stakeholders

   The PET published there is designed to represent the average conditions from 10:00 UTC to 16:00 UTC on the first of July. However, it doesn't take into account the spatial-temporal variations throughout the day, nor does it offer a baseline for typical daily conditions in cities. As a result, it's not possible to test any interventions based on this data. To address this, the research opts to model the PET using the calculation model developed by Koopmans et al. [2020], in line with the reproducability guidelines advocated by the Agile conference [Framework, 2022]. We will need to provide a more detailed explanation of the PET calculation method using Python for the next steps in the process.

2. Strategy approach missing for intervening in public space

   Currently, several municipalities are addressing this issue in their own way. There are no established guidelines for how municipalities should approach this problem, and their strategies vary widely. During the symposium at the University of Applied Sciences "Hot issues" organized by [Hogeschool van Amsterdam, 2023], the differences became evident. However, there is no standardized approach to the strategic implementation of interventions in public space design to make cities more heat-resistant.

1.4. Research aim

The first research aim of this part of the graduation project is to combine an interactive open tool for addressing the spatial-temporal behavior of heat stress in urban environments. A second research aim is a strategy for creating a design to mitigate heat stress with the application case study in the neighborhood of Bospolder Tussendijken in Rotterdam North.

1.5. Academic Value of the Research

The academic value of [Koopmans et al., 2020] can be enhanced by opening up and restructuring the code. This will enable the generation, verification, and comparison of intermediate results, facilitating the integration of research from other disciplines based on a shared knowledge base. As well as spreading awareness through the expansion of educational opportunities. The academic positioning of the strategy development and methodology development alongside the work of [van Esch, 2015] and ongoing developments in the Dutch government places this research as an interesting integration of vulnerable groups which need a more climate-safe environment.

1.6. Social Relevance of the Research

The research introduces an accessible tool that can help a wide range of people understand the impact of heat in their local area. This tool can encourage more efficient communication and inspire collaborative efforts involving various parties to create strategies for mitigating heat stress. The significance of this lies in devising a plan to revamp public spaces, ultimately enhancing the quality of life for residents.

1.7. Research questions

Main research question: "How can a strategy be developed for mitigating heat stress through Physiological Equivalent Temperature model while ensuring a livable environment for vulnerable groups in Bospolder Tussendijken, Rotterdam, the Netherlands?"

The objective was twofold: to create an interactive tool indicating PET heat stress in urban areas of the Netherlands and to design a strategy specifically tailored to Bospolder Tussendijken. This part of the joint thesis focused on reproducable tool to indicate the PET in Dutch cities.

The main question will be answered using this research question:
"To what extent could a reproducible tool help with identifying spatial-temporality of heat stress through PET in urban environments and test design interventions?"

1. What is the position of PET next to other thermal comfort models?
2. Which software is available for open use for modeling heat stress?
3. In what way could the reproducability of [Koopmans et al., 2020] be improved?
4. What is the sensitivity of the wind computation and how could this model be applied to other locations?
5. How can the PET be applied on in Rotterdam for urban design interventions?

1.8. Structure of the report

The structure of this report will include an analysis of the availability of modeling heat stress. This will be discussed in Chapter 2: Thermal comfort models. Next, it accesses the available software in Chapter 3: Thermal comfort models. The physical model of [Koopmans et al., 2020] and the reproducability will be assessed in Chapter 4: Physiological equivalent temperature model. This reveals the improvement of the code. Chapter 5: PET simulator showcases the reproducible procedure of the QGIS plug-in developed by the author. Eventually, in Chapter 6: PET model verification, there will be validation of the model and the potential opportunity to use it for other use cases. Chapter 7: PET application, shows the application of the Rotterdam case study and the application of the thermal comfort model to investigate heat stress, thermal accessibility of several public spaces and testing design interventions. Also chapter Chapter 8 PETs evaluation, looks back on the reproducability of the plugin for other third-party applications. Chapter 9Discussions and limitations, will dive into the discussion and limitation of the research. Chapter 10 addresses the conclusions. Lastly, Chapter 11 proposes the future research. This research is part of the joint graduation research with application to the Rotterdam case study. See Figure 1.5.

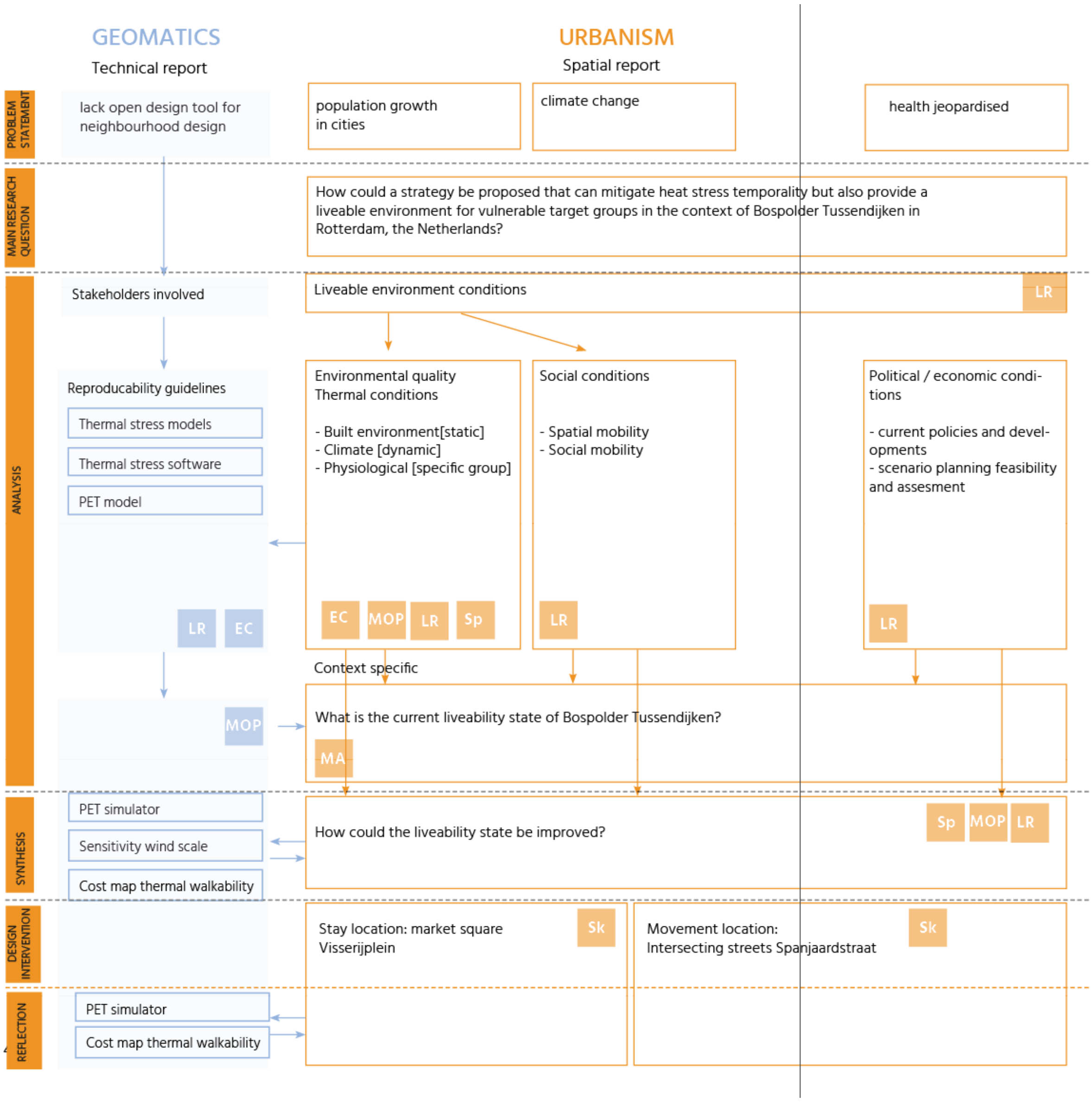


Figure 1.5: Flowchart proposed in the Urbanism part.

2

Thermal comfort models

2.1. Positioning heat stress models

This section positions the heat stress models available related to the researched Physiological Equivalent Temperature model used by [Koopmans et al., 2020].

For thermoregulation for the heat storage model the energy heat balance model is developed. It holds an equilibrium for people to function [Havenith, 1999].

$$\delta s = M + R + C_{\mathrm{v}} + C_{\mathrm{d}} - E \tag{2.1}$$

Metabolic rate (M) is the rate at which the body generates heat internally. Typically, the average metabolic rate at rest is 70 W, while during extensive exercise it can rise to 700 W. Net radiation (R) is the balance between the radiation absorbed and emitted by the body. Mean radiant temperature (MRT) characterizes the radiation field. Convection C_v is the transfer of heat by the movement of air and is enhanced by wind. Conduction C_d refers to the transfer of heat between materials in direct contact. Heat loss occurs through evaporation of sweat and respiration, where exhaled air tends to be warmer and more humid than inhaled air (E). The thermal balance depends on the weather conditions. Higher net radiation tends to increase heat storage, while heat loss can occur through sweating or exposure to wind [Matzarakis and Amelung, 2008] and [Höppe, 1999]. Several thermal indices have been developed to quantify thermal comfort.

Mean Radiant Temperature

The Mean Radiant Temperature (Tmrt) is an effective indicator of thermal stress experienced by the human body due to the radiant heat emitted by its surrounding environment. Conceptually, Tmrt is the uniform temperature where the radiant heat transfer from the human body equals the non-uniform enclosure.

Predicted Mean Vote

The Predicted Mean Vote (PMV) is a widely used thermal index for assessing indoor thermal comfort. It originates from research by Fanger (1970) [Fanger, 1970] and is based on the idea that comfort is achieved when there is thermal equilibrium without physiological stress. The PMV is based on a steady-state heat balance model and is evaluated by individuals in a controlled indoor environment. They rate their experience on a seven-point scale ranging from -3 (cold) to 3 (hot), with 0 representing neutrality.

Munich Energy model

The steady-state model includes the sweat rate as a function of mean skin temperature and core temperature [Mayer and Hoppe, 1987b]. Heat fluxes are determined by the energy balance equation, from the body core to the skin, and from the skin through clothing. Additionally, the individual's age and sex are factored in when calculating both metabolic rate and sweat rate. This model closely aligns with thermophysiology and is highly personalized for each individual.

PET	Thermal perception	grade of physiological stress
< 4 °C	very cold	extreme cold stress
4 - 8 °C	cold	strong cold stress
8 - 13 °C	cool	moderate cold stress
13 - 18 °C	slightly cool	slight cold stress
18 - 23 °C	comfortable	no thermal stress
23 - 29 °C	slightly warm	slight heat stress
29 - 35 °C	warm	moderate heat stress
35 - 41 °C	hot	strong heat stress
>41 °C	very hot	extreme heat stress

Table 2.1: Physiological Equivalent Temperature classification

Physiological Equivalent Temperature

The MEMI was a starting point for the Physiological Equivalent Temperature (PET) developed by [Mayer and Hoppe, 1987a]. It compares complex outdoor conditions to a typical steady-state indoor setting (MRT = Ta, v=0.1m/s, VP= 12hPa or RH=50% at Ta=20C) with the age of a 35 year old male. [Höppe, 1999]. The real outdoor climate is matched with a fictive indoor environment where the same level of temperature discomfort is experienced. Physiological Equivalent Temperature (PET) is linked with the bio climate of the place. It is calculated by determining the temperature at which the energy balance for indoor conditions is the same as the mean skin temperature and sweat rate for outdoor conditions. This makes it easier for people to assess the thermal comfort of a place, as compared to interpreting mean skin temperature values. PET values around 21°C are considered comfortable, while higher values indicate a higher chance of heat stress, and lower values indicate a too cool environment for comfort see Table2.1 [Mayer and Hoppe, 1987b] [Höppe, 1999] [Fiala et al., 2012]. This is an widely used measure around urban planners, and persons not familiar with thermo-physiology. This semantic representation of spatial temporal influences of built environment as static factors, physiological factors as static factors and climate factors as dynamic factors give a better understanding for other disciplines to deal with the effects of heat stress on the public health.

Wet bulb globe temperature

The Wet Bulb Globe Temperature (WBGT) is a measure used to assess heat stress. It combines the readings from three instruments: the Natural Wet Bulb (NWB), Globe Temperature (GT), and Dry Bulb (DB) thermometers. It was developed during World War II in the military and indicates the amount of exercise a person can handle before experiencing heat stroke. Nowadays, it is a common measure for employees working outside [RIVM, 2023]. This links metabolic actions to temperature and requires specific materials to obtain accurate measurements. Shortcomings include the underestimation of humidity and air movement, which can lead to an unclear understanding of stress in environments with limited evaporation. This inadequacy exacerbates the existing inconsistencies in effective temperature measurements for two main reasons [Budd, 2008].

UTCI

The Universal Thermal Climate Index (UTCI) is an internationally standardized thermal index developed by the World Meteorological Organization (WMO). It assesses thermal comfort or stress in both outdoor and indoor environments by considering various environmental factors such as air temperature, humidity, wind speed, and radiation from the sun and surrounding surfaces. UTCI estimates the equivalent air temperature at which the human body would experience thermal stress as it would under the prevailing environmental conditions. It's widely used for assessing heat stress and thermal comfort in research, policy, and practice due to its comprehensive and standardized approach that can be applied across different geographic locations and climates [Blazejczyk et al., 2013].

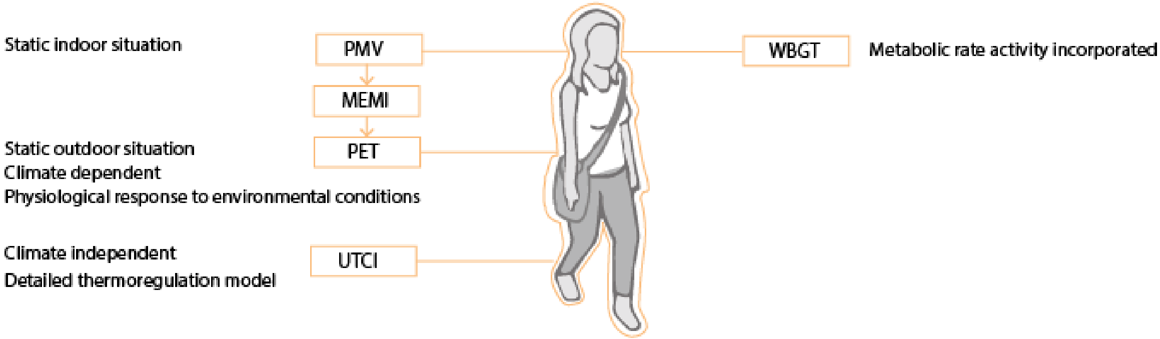


Figure 2.1: Overview thermal models

2.2. Conclusions

Several models have evolved from the well-known Physiological Equivalent Temperature (PET) model, ranging from thermostatically PMV and MEMI to a more universally comprehensible PET model across disciplines. These models consider three key influences: dynamic climate data, static built environment data, and standardized physiological performances. Given the standardization of the PET model in the Netherlands, it remains the appropriate choice for modeling the thermal comfort of citizens in the country. PET serves as a comparison between complex outdoor conditions and a typical steady-state indoor environment, aligning indoor energy balance with outdoor mean skin temperature and sweat rate for simplified thermal comfort assessment. However, PET is a static model for indoor thermal environments, whereas UTCI and WBGT incorporate factors such as clothing and metabolic rate, providing more comprehensive overview.

3

Thermal comfort software

3.1. Requirements

Software requirements

In the previous chapter, various models are discussed for identifying heat stress in urban environments. The standard measure for the Netherlands is the PET. This chapter examines the available software programs for this purpose. The selection criteria for the software depend on urban climate factors and the accessibility of data to users, in line with Agile reproducability guidelines [**?**].

Reproducability requirements

In the context of knowing, wanting and acting as outlined in the Deltaplan (2018), it is crucial to ensure that the software is reproducible for a wider audience of users.

With the Agile (2020) reproducability Guidelines document, Figure-E.9 refers to the reproducability assessment of the different stages of reproducing georeferenced material. The aim of this report is that every step towards higher reproducability counts. Authors should also be aware of the benefits, such as contributing to a community. The three steps of geo-handling are thus distinguished. First, the input data are assessed, for example, if the data are open available and well documented. Secondly, the methods are described, i.e. the software tools for pre-processing the data, methods for analysis and processing, and finally the computing environment and visualisation of the material. Finally, the results will be evaluated.

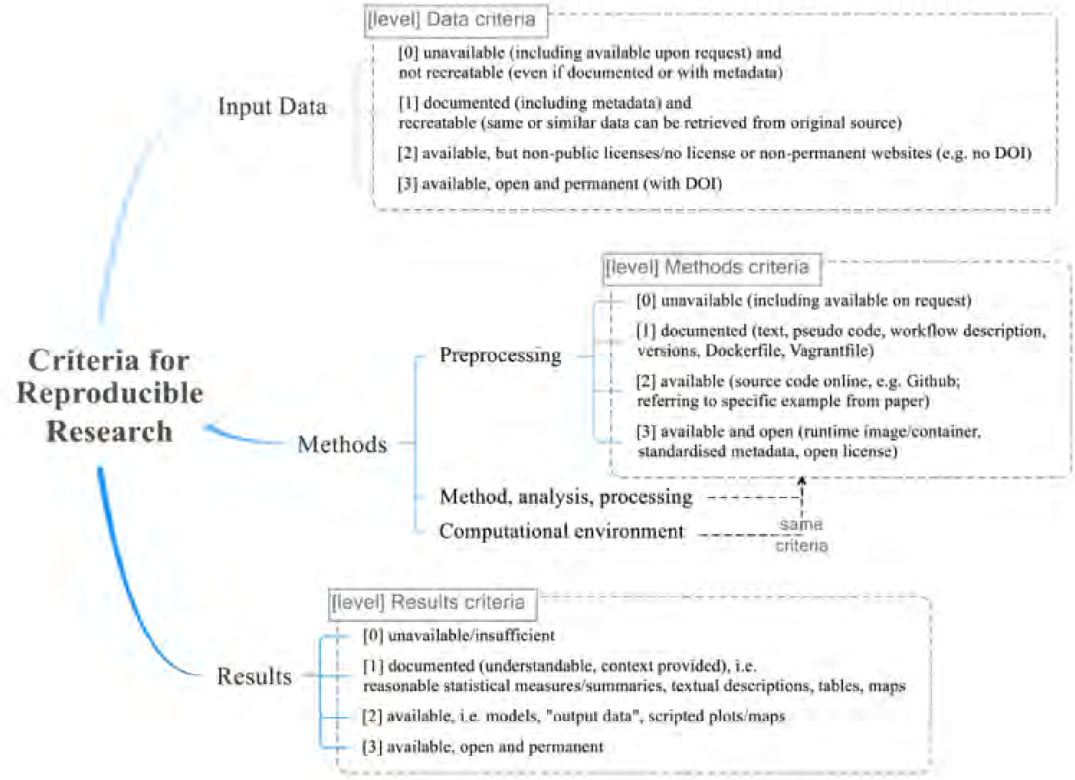


Figure 3.1: Reproducability guidelines

AGILE Association of Geographic Information Laboratories in Europe

Website: https://osf.io/phmce/
Version: December 2020
DOI: 10.17605/OSF.IO/CB7Z8

REPRODUCIBILITY CHECKLIST

For all **datasets** included/produced in the paper, check if data:

- ❏ Is provided in a non-proprietary format
- ❏ Is documented for third parties to reuse
- ❏ Is accessible in a public repository and has an open data licence

For all **software tools/libraries/packages** and **computational workflows** included/produced, check if:

- ❏ Reproduction steps are explained in a README (plain text file), flowchart, or script
- ❏ Computational environments (including hardware) are documented or provided
- ❏ Versions of relevant software components (libraries, packages) are provided
- ❏ All parameters and expected execution times for the computational workflow are provided
- ❏ Software developed by the authors is available in a public repository and has an open licence
- ❏ There is a clear connection between **tables, figures, maps, and statistical values** and the data and code that they are based on, e.g., using file names or documentation in the README

In the **Data and Software Availability section**, check if you include:

- ❏ Data and software statements (see examples below)
- ❏ The reasons, if any, for not being able to share (parts of) data or code

For all **data and software** check that:

- ❏ All datasets and code (used or mentioned) are assigned DOIs
- ❏ Datasets and code are cited throughout the paper

After acceptance in the **camera-ready paper** check that:

- ❏ If data has been shared privately or anonymously for peer review, they are updated with all metadata and accessible via a DOI and referenced from the paper
- ❏ If a reproducibility review report will be published for your paper, a DOI URL in the Data and Software Availability section is included using the following template:
  A reproducibility report for this paper is available confirming that [considerable parts of the computational workflow / all results / Figures 1 and 4] could be independently reproduced, see https://doi.org/link_to_report.

You will find more checklists, some of them much more extensive than this one, online. If you like this style of ensuring your research is reproducible, take a look at the checklist for the *Geoscience Paper of the Future*[1] for one related to GIScience, the comprehensive *ASCE checklist*[2] with an engineering perspective, or look for checklists suited to your methods, e.g., for machine learning[3] and sharing code for machine learning[4]. The FAIR[5] (Findable, Accessible, Interoperable and Reusable) principles for scientific data management and stewardship provide further extensive guidance on improving the way we share digital assets of research.

[1] https://doi.org/10.1002/2015EA000136
[2] https://doi.org/10.1061/(ASCE)WR.1943-5452.0001215
[3] https://ai.facebook.com/blog/how-the-ai-community-can-get-serious-about-reproducibility/
[4] https://medium.com/paperswithcode/ml-code-completeness-checklist-e9127b168501
[5] https://www.go-fair.org/fair-principles/ and https://doi.org/10.5281/zenodo.2248199

Association of Geographic Information Laboratories in Europe | https://agile-online.org/ | 2

Figure 3.2: Reproducability checklist according to [Framework, 2022]

Urban designers climate factors

For urban designers it is important to know what is changeable of the built environment which influences the climatic dynamic factors in cities. It's important to consider factors such as radiation, air temperature, and wind computation. The physical built environment, including surface materials, water, and vegetation, can significantly impact the dynamic values in cities. Additionally, it's crucial to examine the micro climate, which can vary based on size. The software should accurately handle fluctuations in the presence of landscaping elements. The influence of shadow and vegetation patterns is limited to the immediate surroundings of trees or buildings. Therefore, it is important to capture these mitigating fluctuations [van Esch, 2015]. The software needs to be able to scale from a small to a large scale, considering various scopes such as neighborhoods or entire cities. Next to this, the running time of the simulations should be taken care of. When it comes to wind modeling, it's important to distinguish between methods. One method focuses on the abstraction of wind flow in one direction with the representation of an averaging method of building height resistance translated from roughness layer to the ground [Macdonald et al., 1998] (Figure 3.3). However, a more advanced model, such as the computational fluid dynamics model, does take into account the real behavior of wind [Mirzaei, 2021] (Figure 3.4). In order for the wind computation tool to be effective, it should be easy to operate and provide results quickly. The time taken to run the tool is a crucial factor that affects the accuracy of the results, and needs to be considered at all levels of computation to ensure robustness.

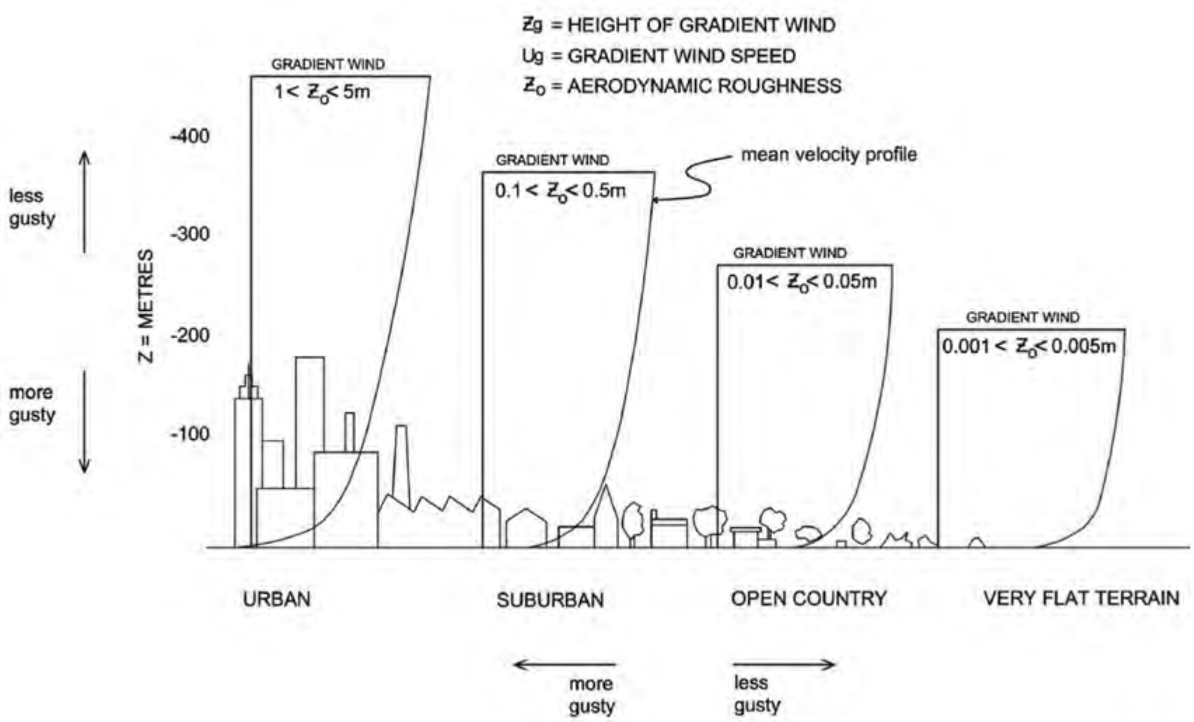


Figure 3.3: Wind modeling roughness layer retrieved [Cochran and Derickson, 2005] figure 4

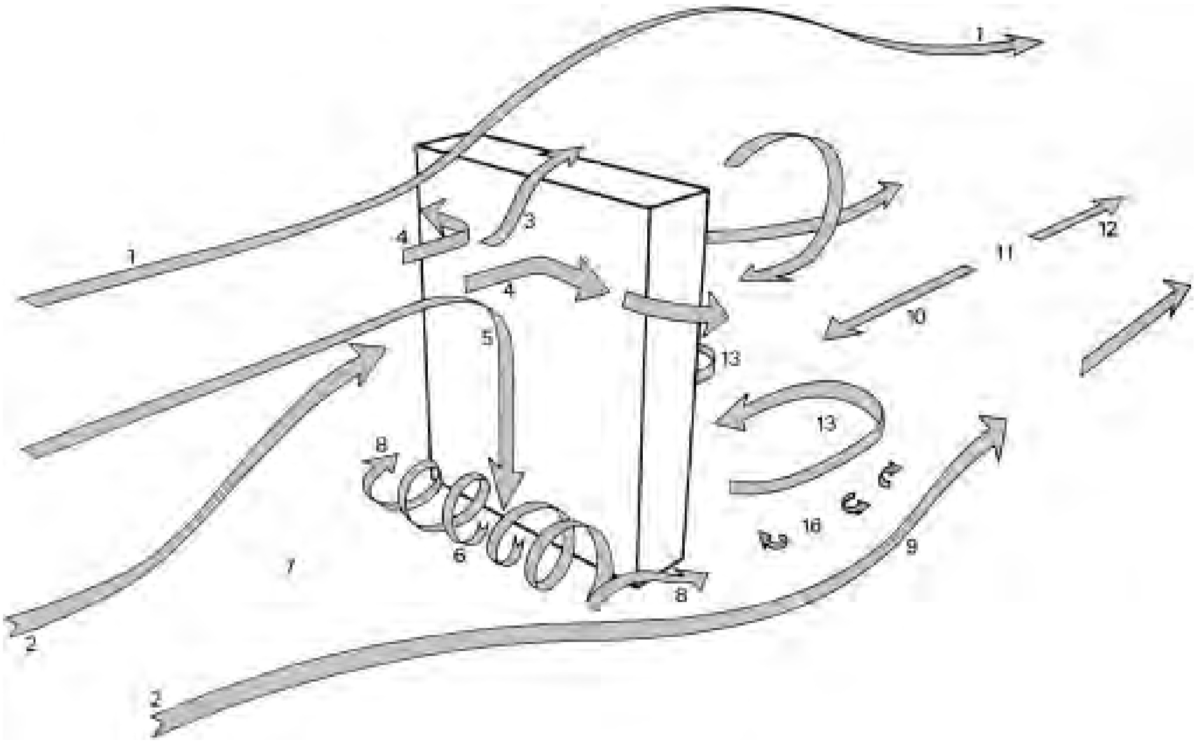


Figure 3.4: Schematic representation of 3d wind flow pattern around a high-rise building retrieved from Urban Physics: Effect of the micro-climate on comfort, health and energy demand by [Moonen et al., 2012]

3.2. Thermal comfort software models

Several software options are available for modeling the urban microclimate, including ENVIMET, PET national map, Urban Microclimate, UMEP and CRC tool. Each of these software options has unique features that distinguish them from each other."

ENVIMET

ENVIMET is one of the highly accurate climate modeling software and is used by heat experts [met GMBH, n.d.]. The wind modeling uses computational fluid dynamics. However, this software is not open source. The software could only be retrieved with a fee subscription. Also due to its high precision this modeling software the runtime is relatively large and is therefore suitable for calculating small urban areas. A simplification of this modeling software is also suitable for indicating the micro climate on urban level.

Urban Multi-scale Environmental Predictor

The Urban Multi-scale Environmental Predictor (UMEP) is a climate service tool, designed for researchers and service providers (e.g. architects, climatologists, energy, health and urban planners) presented as a plugin for QGIS. It works with different methods like pre-processor, processor and after result. All dependencies have to be performed sequentially in order to make it working. The modifications of inbetween results are only suitable for the plugin in order to let it work Lindberg et al. [2018]. The wind modeling is done by a the Macdonald et al. [1998] method. With the proper knowledge of the plugin it is usable for neighborhood scale and city scale.

Urban micro climate

Urban micro climate is a widely used climate analysis software among architects ([MIT, 0000]). It is integrated as a plugin in the Rhino environment, with plugins called Ladybug that read various climate data. The primary output is the dry bulb temperature, which does not reflect the PET. However, it is adaptable software environment for designers to make urban environmental differences and test the results. In the input CAD file, buildings, courtyards, public squares, roads, and trees are represented in poly lines or surfaces. This should be regurlary updated and could potentially overestimate the performance of mitigating measures like evaporative surfaces. These input data are a representation of the real world and need to be generated first by manually drawing or retrieving from the BAG. Despite this, many urbanists use QGIS to perform geo-spatial analyses.

PET national map

It is developed for the weather input of the Netherlands, and therefore suitable for Dutch test cases. There is provided documentation of [Koopmans et al., 2020] on the code. It has an 1-m accuracy which makes it suitable for modeling fluctuations of shading and evaporative surfaces. The input data is obtained from publicly available sources and generated for each location in the Netherlands in a seamless manner. The wind modeling uses the MacDonald method [Macdonald et al., 1998]. It is suitable for urban micro climate modeling to identify critical areas. The code itself is not publicly available but the steps are documented in [Koopmans et al., 2020].

CRC tool

CRC tool does not indicate the areas which are endangered by heat stress but only showcases potential mitigation elements and measured in costs [Deltares, 2020]. It is a privately developed tool and not transferable to other interfaces to reproduce the outcome yourself.

3.3. Conclusion

The software requirements were assessed if it was a reproducible manner of retrieving the information with the connection between knowing, wanting and acting see Table3.2. Therefore it is necessary to indicate the critic areas and also being able to intervene in the public space. Next to that it should be reproducible for a broader audience. Therefore the AGILE requirements of reproducability are important which are divided in input, methods and results. Also the requirements of the influencing factors of the urban environment which can be changed by the urban designer should be integrated in the software. Small fluctuations of evaporative surfaces or shadow are important to model. For the usability for multiple users the scalability of the area is important as well as the runtime of the software. As seen in the inquiry there are different software models with their own purpose and audience. The PET map developed for the Netherlands does have the potential to be scaled to other locations in the Netherlands [Koopmans et al., 2020]. It has a scalability potential for multiple research areas and it can handle the fluctuations of evaporative and shadow patterns. It does use an abstraction of the wind method to speed up the computation process. In the next chapter the PET calculations will be addressed and the reproducability will be assessed.

Table 3.1: Comparisonsoftwaremodels

	Urba nmicro climate	**PETkaart**	**ENVIMET**	**CRC(ClimateResilient City)tool**	**UMEPtool**
Open source	YES	YES	NO,against fee	NO	YES
Adaptabledata	YES	NO	YES	YES	**NO**
Publisher	MIT	Wageningen university, Witteveen en Bos	ENVIMET GMBH Essen Germany	Deltares	Fredrik Lindberg, TingSun,Sue Grimmond, Yihao Tang, Nils Wallenberg
Users	Architects	Public accessible as viewer	Commercial	Public accessible and advanced version against fee.Commercial	Researchers and service providers(e.g.architects, climatologists, energy, health and urbanplanners)
Website	https://urbanmicroclimate .scripts.mit.edu/umc.php	https://www.klimaateffect atlas.nl/nl/	https://www.envi-met.com/	https://www.deltares.nl/en /software/climate-resilient-city-tool/	https://umep-docs.readthedocs. io/en/latest/
level	3D	2.5D	2.5D	2.5D	1and2D
Software	Grasshopper and ladybug	Viewable online or can be retrieved by klimaat effectatlas	ENVIMET	CRC tool	QGIS
Input	3D geometry, Weather data	Weather data KNMI local, spatial data of built environment	Weather data national	Weather data	Built environment height and canopy trees
Output	Dry bulb temperature, energy consumption	PET	Mean Radiant Temperature(MRT), Physiological Equivalent Temperature (PET) and Universal Thermal Climate Index (UTCI)	Heat reduction, Cost analysis	Shadow, wind, skyviewfactor, UHI, Thermal outdoor comfort
Scope area	Micro level	Micro, Meso, Macro level	Micro level	Macro, Meso and Micro level	Micro,Meso,Mesolevel
Purpose	Indicate areas and design	Indicating high experienced temperatures in areas	Indicate areas and design	Indicate areas and design	Indicate urban heat island and how to mitigate heat
Takes environment into account	Built environment	Built environment,evaporation water and greenery	Evaporation water and greenery and green roofs and green facades	Built environment, evaporation water and greenery	Buildings and vegetation
Runtime 1km x 1km	0-10min	-	100+min	0-10min	0-10min

Table 3.2: Comparison software table

4

Physiological Equivalent Temperature (PET) model

4.1. Physical model

The method for the Physiological Equivalent Temperature (PET) calculation is described in Koopmans et al. [2020]. The formulas used, along with the corresponding variables and units of measure, are provided in the flowchart of figure 4.1. This chapter provides an overview of the formulas.

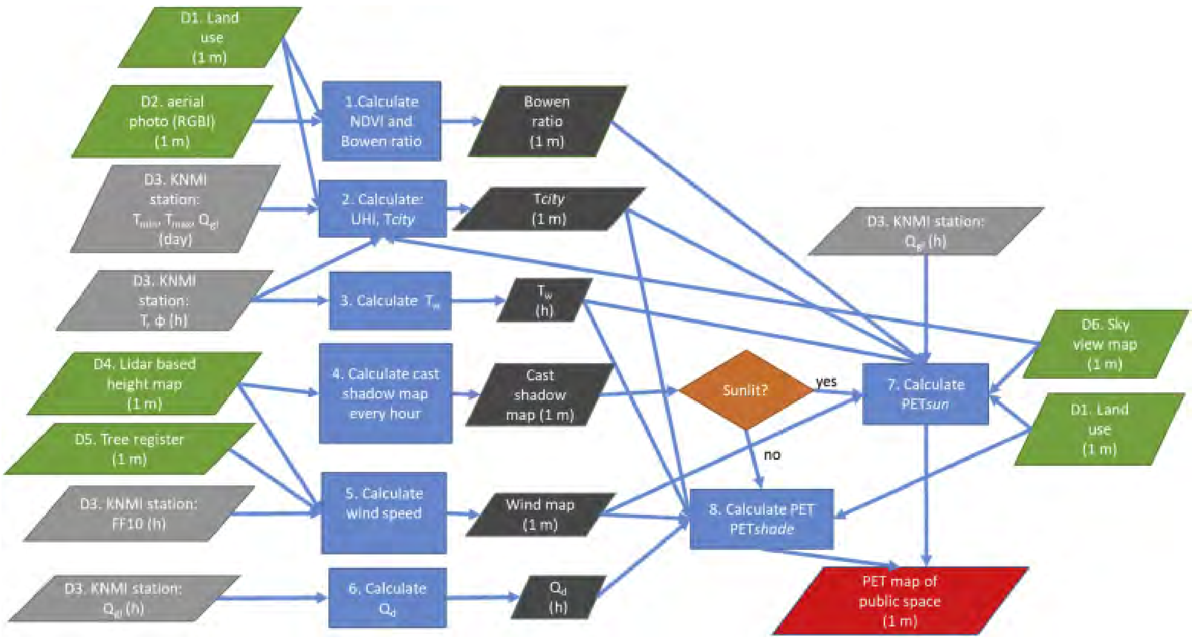


Figure 4.1: Simplified flowchart as published in [Koopmans et al., 2020]

PET (°C) is calculated for a sun, a shade or a night situation. The parameter depends on the air temperature T_a (°C), measured at a height of two meters above the land surface, the wet bulb temperature T_w ($°C$), the global solar radiation Q_s (Wm^{-2}), the diffusive radiation Q_d (Wm^{-2}) and the latent heat flux. PET_{sun} is expressed by

$$\begin{aligned} PET_{\mathrm{sun}} = &-13.26 + 1.25 T_a + 0.011 Q_s - 3.37 \ln(u_{1.2}) + 0.078 T_w + 0.005 Q_s \ln(u_{1.2}) 5.56 sin(\phi) \\ &- 0.0103 Q_s \ln(u_{1.2}) \sin(\phi) + 0.0546 B_b + 1.94 S_{vf} \end{aligned} \tag{4.1}$$

where σ ($5.67 \cdot 10^{-8}$ Wm^{-2}K^{-1}) is the Stefan Boltzmann constant, S_{vf} (-) denotes the sky-view factor and ϕ (degrees) denotes the solar elevation angle. The latent heat flux follows from the Bowen ratio B_b that relates this flux to the sensible heat flux. The latent heat flux follows from evaporation of water from the land surface. Evaporation is affected by the wind speed, which is measured at a height of 1.2 m $u_{1.2}$ (ms^{-1}). PET_{night} and PET_{shade} are given by

$$PET_{\text{night,shade}} = -12.14 + 1.25T_a - 1.47\ln(u_{1.2} + 0.060T_w + 0.015S_{vf}Q_d + 0.0060(1 - S_{vf})\sigma(T_a + 273.14)^4 \tag{4.2}$$

Air temperature and wet bulb temperature

The urban heat island coefficient $UHI_{\max}$ that is used for calculating the air temperature on a 2-m level. The coefficient follows from

$$UHI_{\max} = (2 - S_{vf} - F_{veg})\sqrt[4]{\frac{S\downarrow \cdot (T_{\max} - T_{\min})^3}{U}} \tag{4.3}$$

This equation consists of a physical part and a meteorological part. The first part describes the physical part with the sky-view factor S_{vf} and the vegetation fraction F_{veg} within a certain source area. Water bodies are treated as buildings overnight and as grass during the day. Both parameters are averaged over a source area of 500 x 1100 m with a resolution of 25 meters. The orientation of this source area depends on the wind directionHeusinkveld et al. [2014]. The second part consists of a meteorological term $S\downarrow$ (Kms^{-1}) that represents the mean downward shortwave radiation and the average wind speed during the day U (ms^{-1}). The temporal conductivity $T_{\max}$ - $T_{\min}$ (°C) is measured between 8:00 UTC and 7:00 UTC the next day. Air temperature at given hour $T_a(h)$ (°C) follows from

$$T_a(h) = T_{\text{refstation}} + UHI_{\max} \cdot d_{\text{cycle}}(h) \tag{4.4}$$

where $T_{\text{refstation}}$ (°C) denotes the atmospheric temperature measured at a KNMI weather station at a height of 1.2 m, d_{cycle} corrects $UHI_{\max}$. The diurnal correction factor varies between -0.02 and 1 as can be seen in Appendix **??**. The table in this appendix was derived from Oke [1982].

The wet bulb temperature $T_w(h)$ (°C) follows from the air temperature and the solar elevation angle as

$$T_w(h) = T_a(h)\ \text{atan}\ (0.151977(\phi + 8.313659)^{0.5}) + \text{atan}\ (T_a(h) + \phi) - \text{atan}\ (\phi - 1.676331) + 0.00391838\ \phi^{1.5}\ \text{atan}(0.023101\ \phi) - 4.686035 \tag{4.5}$$

Wind velocity

For the wind calculation the MacDonald method is used [Macdonald et al., 1998]. The calculation provides a spacial frontal area density factor λ that can be written as

$$\lambda_{\text{tot}} = 0.6\lambda_{\text{buildings}} + 0.3\lambda_{\text{trees}} + 0.015 \tag{4.6}$$

The factor resembles the resistance of buildings and trees on the wind. The resistance depends on the height of buildings and trees in front of a spatial location and on the variation in height. Heights are considered over a source area A_s (m^2) of 280 x 140 m area with a scaled resolution of 35 meters. Frontal areas determine the perpendicular surfaces towards the wind direction. If there are a lot of buildings in this direction then the frontal area density will be high which will lead to less wind. The frontal area density factor scales the wind speed that is measured by KNMI weather stations at a height of 10 meters above land surface u_{10} (m/s). For the PET calculation a wind speed at a height of 1.2 meters above land surface $u_{1.2}$ (m/s) has to be obtained.

With a sufficient frontal surface area $\left(0.6\lambda_{\text{buildings}} + 0.3\lambda_{\text{trees}}\right) > 25/A_s$ the wind speed at this level follows from

$$u_{1.2} = u_H \exp\left[9.6\lambda\left(\frac{1.2}{H} - 1\right)\right] \tag{4.7}$$

where λ expresses either $\lambda_{\text{buildings}}$ or λ_{trees}. The wind speed at roof in case of a building at height H (m) is written as u_H (m/s)

$$u_H = \frac{-u^*}{B}\ln\left(\frac{A + Bz_w}{A + B_H}\right) + u_{zw} \tag{4.8}$$

the parameters A and B are presented by Table-4.1 and z_w (m) denotes the top of the roughness layer. The friction velocity u^* follows from

$$u^* = 0.4\ \frac{u_{60}}{\ln\left(\frac{60-d}{z_0}\right)} \tag{4.9}$$

where z_0 is the surface roughness length, u_{zw} is expressed as

$$u_{zw} = u_{60} \frac{\ln\left(\frac{z_w - d}{z_0}\right)}{\ln\left(\frac{60-d}{z_0}\right)} \tag{4.10}$$

where the wind at a height of 60 meter follows from the wind speed measured by a weather station $u_{60} = 1.3084 u_{10}$. If the frontal surface area is insufficient according to $\left(0.6\lambda_{\text{buildings}} + 0.3\lambda_{\text{trees}}\right) < 25/A_s$ then the wind speed directly follows from

$$u_{1.2} = 0.6350\, u_{10} \tag{4.11}$$

λ_{tot}	d/H	z_w/H	z_0/H	A/H	B
0.05 (<0.08)	0.07	2.0	0.048	-0.35	0.56
0.11 (0.08 till 0.135)	0.26	2.5	0.071	-0.35	0.50
0.16 (0.135 till 0.18)	0.32	2.7	0.084	-0.34	0.48
0.20 (0.18 till 0.265)	0.42	1.5	0.08	-0.56	0.66
0.33 (=> 0.265)	0.57	1.2	0.077	-0.85	0.92

Table 4.1: The A and B interpolation matrix

Diffusive radiation

The diffusive radiation follows from the measured solar radiotion is calculated as

$$Q_d = \begin{cases} Q_s & \tau_a \leq 0.3 \\ (1.6 - 2\tau_a)\, Q_s & 0.3 \leq \tau_a \leq 0.7 \\ 0.2 Q_s & \tau_a > 0.7 \end{cases} \tag{4.12}$$

The atmospheric transmitivity τ_a is given by

$$\tau_a = \frac{Q_s}{1367 \sin(\phi)} \tag{4.13}$$

Latent heat flux

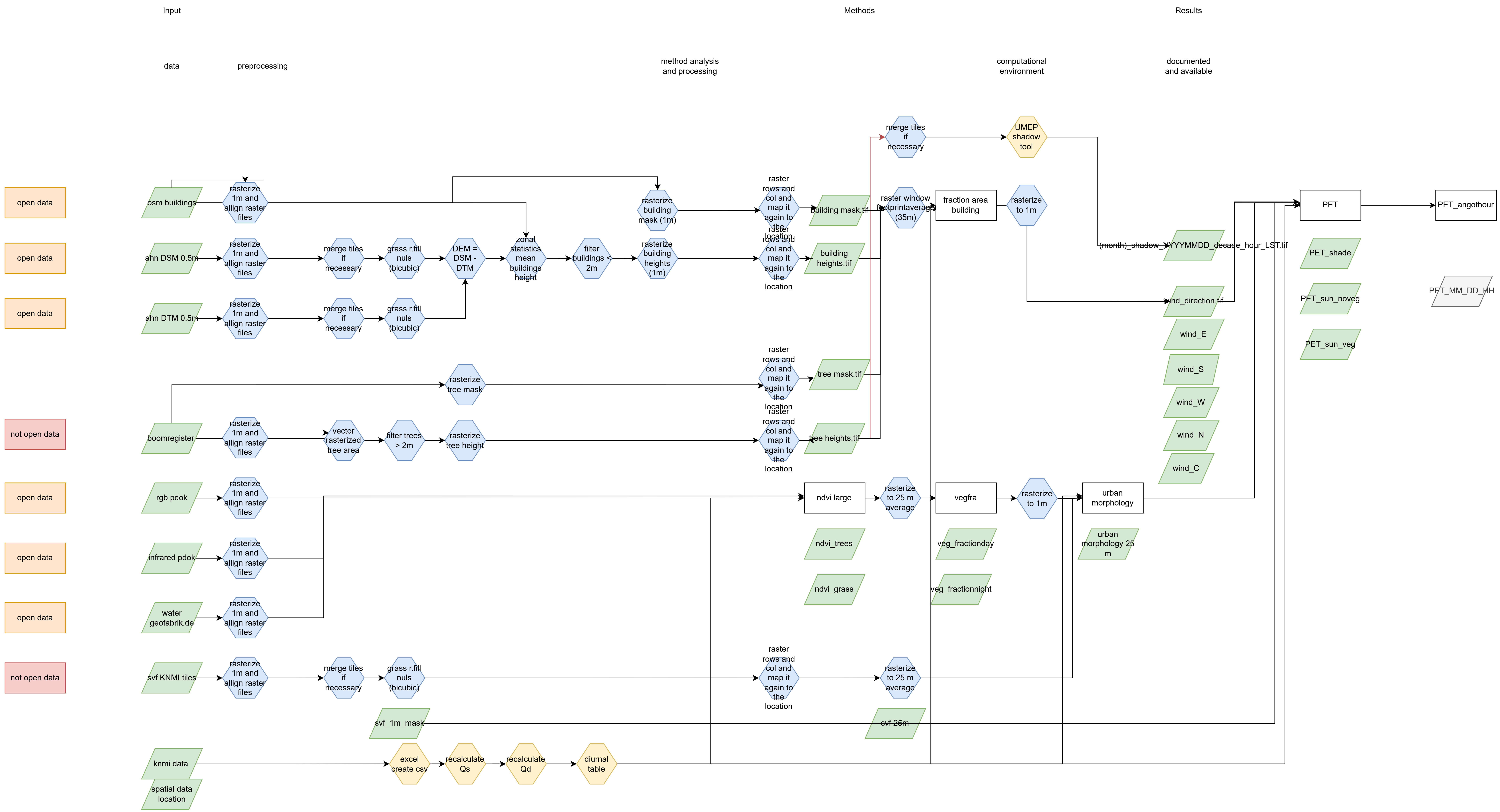
In order to retrieve the evaporative surfaces the Normalized Difference Vegetation Index ($NDVI$) is introduced. This index evaluates the red band of the RGB image and the red band of the infrared image. The index provides ranges that represents the health and evaporative functioning of the greenery in the urban environment. For information about the values see Appendix G.

$$NDVI = \frac{NIR - R}{NIR + R} \tag{4.14}$$

If the $NDVI$ exceeds 0.16 then vegetation is assumed to evaporate well and the Bowen ratio is set to 0.4. For impervious urban surfaces the ratio is set to 3.0 [Oke, 2002].

4.2. Reproducability paper code guidelines Koopmans et al. [2020]

First, the script provided by Sytse Koopmans from Wageningen will be assessed for reproducability according to the Framework [2022] as named in Figure E.9 and Figure 3.2. Next, the new code for assessing PET on a city scale will be executed. An analysis was conducted based on the code provided by the author (see Figure 4.2).



Input
Methods
Results
data
preprocessing
method analysis and processing
computational environment
documented and available
open data
open data
open data
not open data
open data
open data
open data
not open data
osm buildings
ahn DSM 0.5m
ahn DTM 0.5m
boomregister
rgb pdok
infrared pdok
water geofabrik.de
svf KNMI tiles
knmi data
spatial data location
rasterize 1m and allign raster files
merge tiles if necessary
grass r.fill nuls (bicubic)
DEM = DSM - DTM
zonal statistics mean buildings height
filter buildings < 2m
rasterize building mask (1m)
rasterize building heights (1m)
rasterize tree mask
vector rasterized tree area
filter trees > 2m
rasterize tree height
raster rows and col and map it again to the location
building mask.tif
building heights.tif
tree mask.tif
tree heights.tif
merge tiles if necessary
UMEP shadow tool
raster window footprintaverage (35m)
fraction area building
rasterize to 1m
(month)_shadow_YYYYMMDD_decade_hour_LST.tif
wind_direction.tif
wind_E
wind_S
wind_W
wind_N
wind_C
PET
PET_angothour
PET_shade
PET_sun_noveg
PET_sun_veg
PET_MM_DD_HH
ndvi large
rasterize to 25 m average
vegfra
rasterize to 1m
urban morphology
ndvi_trees
ndvi_grass
veg_fractionday
veg_fractionnight
urban morphology 25 m
svf_1m_mask
svf 25m
excel create csv
recalculate Qs
recalculate Qd
diurnal table

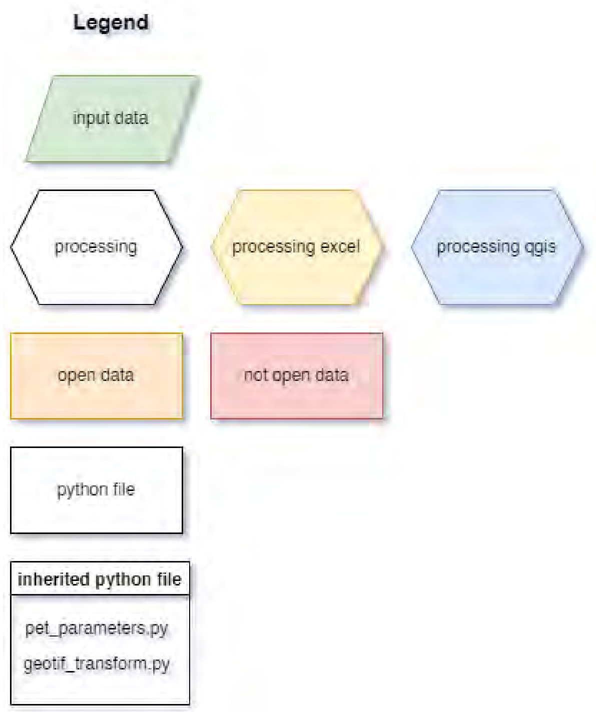


Figure 4.3: Legend flowchart

Input data

The input data is focused on the datasets required to run the method in order to conduct the results. The input data is categorised in whenever they are in non-proprietary format, if third party reuse is possible, if the guidelines are referenced to the data. The datasets provided are in non-proprietary formats and include Geotiff, text, and vector datasets in Geopackage format. The spatial data consists of raster Tiff and vector datasets, while the climate data is in text format. The text file is derived from [KNMI, 0000] and contains hourly data. It includes atmospheric temperature (TT), wind speed (FF), wind direction (DD), global solar radiation (Q), relative humidity (RH), and minimum and maximum temperatures (Tmin and Tmax) between 8:00 UTC and 9:00 UTC of the following hour. It also includes the average daily wind speed (U). The file has been modified to calculate Qdif, generate Sunalt, activate the Day/Night switch, and display the diurnal factor on an hourly basis, making it not immediately repeatable for other users. The vector data, including building envelopes, trees, and water, are derived from [Geofabrik, 2020] and [NEO and Geodan, 2024], saved as geopackages, and eventually rasterized as Tiffs in QGIS. The spatial dataset is in Tiff format. Geospatial information, which could be seen as static parameters, are added later for each dataset, such as RGB, Infrared, Sky view factor, and rasterized vector datasets. These could have been generated immediately by saving the files as Geotiff and handling them with the metadata properties. Due to the repetition in mentioning these static parameters and by changing that in each file, inconsistencies can appear which cause incompatabilities. These static parameters next to the dynamic parameters of the climate should be centralized on a place where each separate python file could make use of.

The datasets used for third-party purposes are referenced in Figure 10 and can be obtained from various sources such as PDOK [Kadaster, 2023], geofabrik.de [Geofabrik, 2020], KNMI, and AHN. It's important to note that the tree registry data from WUR, NEO, and Geodan is not accessible to the general public. The research outlines two methods for obtaining tree registry data or determining tree crown height using the position and height of trees, one of which involves using AHN and NDVI. The paper utilizes accurate tree registry data.

The paper discusses the authors referenced in the data. All the links are accessible, but the datasets must be downloaded separately from various web links. The bomen register and Sky View Factor are initially not available. The bomen register contains high-quality data and requires a subscription. For other locations, a workaround is needed to make the data publicly accessible. As for the Sky view factor, a script must be written to derive the datasets of the Sky view factor and the sky view factor mask from [KNMI, 2023].

For each Tiff image the georeferencing were done separately.

```
    latarray=np.zeros(shape=(h,w))
    lonarray=np.zeros(shape=(h,w))
    ymin=171322
    ymax=177291
    xmin=439813
    xmax=445583
    latmin=xmin+(xmax-xmin)/(2*h)
    latmax=xmax-(xmax-xmin)/(2*h)
    lonmin=ymin+(ymax-ymin)/(2*w)
    lonmax=ymax-(ymax-ymin)/(2*w)
    ##cells=32*48
    ##create lat and lons
    for i in enumerate(lonarray[0]):
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
    #print('lonarray',lonarray)
    for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)
```

Methods

The method section is subdivided into pre-processing, method, analysis and processing, and computational environment. The method to develop the procedure is open data licence. The software code is, however, only retrieved by the developers themselves. This could be made open and available via GitHub or a plugin of QGIS. This was due to the lack of amount of money to create reproducible software for third-party use.

The pre-processing reproduction steps are documented in Appendix A of [Koopmans et al., 2020] via the DOI that is provided https://doi.org/10.1016/j.buildenv.2020.106984. In the paper is a clear connection between tables, figures, maps and statistical values and the documentation is available in a README file. However, third-party users are hard to regenerate. Climate data is modified in Excel using a CSV format, and the climate parameters are referenced multiple times in separate Python computation files. It also involves generating missing climate parameters such as Qs in the correct units of measure, as well as Qdif, salt, and diurnal factors. Refactoring the data by centralizing the parameters as dependencies is useful to make it more operable. For method, analysis, and processing, it is necessary to dive into the software tools/libraries/packages and computational workflow. The reproduction steps are visible in the flowchart in [Koopmans et al., 2020] but are a bit oversimplified. Figure 4.2 with legend Figure K.2 shows the elaborated steps necessary to reproduce the same in-between results and results.

For the method, the approach of calculating the PET is intended to calculate the wind by the MacDonald method validated for the Dutch context (to be more specific in the Wageningen Herwijnen context). Each Python code has a README file that explains the use, but not the precise intermediate output results. For the analysis part, the same in-between output should be generated and reproduced through other parties. Since the Python files were not directly connected, all the Python file outputs were Tiff-based and of the format of CSV output per cell a value. Also, the Fveg and Svf were manually averaged over 25m outputs in QGIS. This led to non-linearity in the generation of the intermediate values, since modifications of the output files of Python were modified in QGIS. This causes untraceable intermediate step output files of in-between procedures. To upgrade to ideal, a software package is required with structured metadata, tests, and an automated workflow if applicable add a link to the running instance of the software. To upgrade to Ideal: minimum, versioned code repository to upload to GitHub and an open license of the software is required. The processing involves using Python software for computational steps, along with importing libraries such as PIL, Pandas, and Numpy. There are 7 separate Python files: ndvi_calculator, svf_footprint, vegfra_footprint, fraction_area_buildings_treeregr, PET_angothour, and PET_calculate. Unfortunately, these Python files are not interconnected, leading to disjointed results. The ndvi_calculator is used to calculate areas that qualify as evaporative surfaces and contain a Bowen ratio. svf_footprint and vegfra_footprint depend on wind direction to average the values on a 25m resolution. fraction_area_buildings_treeregr is for calculating wind, while PET_angothour projects climate scenarios for 2050 with high- and low emissions. PET_calculate combines output files of intermediate steps and climate dynamic data to calculate PET in sunny and shady locations.

For the computational environments, Python was used. Pre-processing values do include actions in software like Excel, QGIS basic environment and the UMEP QGIS plugin for generating the shadow patterns of buildings and trees. Also as already mentioned the in-between results were modified in QGIS. To properly install UMEP, you need a compatible version of Python in both QGIS and PyCharm. To minimize the amount of errors the Excel manual procedure could be included in a Python file, which can generate the desired output per day. The visualization environment is QGIS. This is a graphical environment used by urban designers.

Results

The results of the code have been verified for the Wageningen area, and the names of the services for download are provided. The software has been assessed through interaction with the publishers. One of the requirements is a camera-ready paper. Peer review is conducted in cooperation with Gert-Jan Steeneveld and Bart Heusinkveld but is not incorporated in the code. If a reproducability review report is published, a DOI will be included in a template. The report should ensure that all the steps in the workflow are reproducible. On request the output is available.

Assessment reproducability

This scientific research institute of Wageningen University has included reproducability measures for its verified research on PET in Wageningen. After this chapter a conclusion assessment has been evaluated as seen in table 4.2, with 0 as minimum to reproducability measures and 3 as maximum of reproducability. For the processing part, reproducability can be increased. Many pre-processing steps are needed to handle the good input data for the code to work. For the method, analysis and processing the method is well documented in [Koopmans et al., 2020], but due to lack of money this is not funded to make it open software for third parties to use. For the processing ndvi_calculator, svf_footprint, vegfra_footprint, fraction_area_buildings_treeregr, PET_angothour, PET_calculate are used, see appendix H. Also there are modifications in QGIS instead of linearity in Python file handling, this hinders the calculation workflow to work fluently. Next, the factorisation of static parameters such as location and weather values is highly valued instead of filling them into each file. Also, the Python code is not openly available in a GIT repository for others to see. The computing environment is QGIS, Python, the UMEP plugin in QGIS and Excel. Python is used for the calculations and Excel for the weather data parameters. QGIS is used only as a visual environment. The results are documented in Appendix A in [Koopmans et al., 2020], but the results are only available and documented on request. If you wish to reproduce the results yourself, the final results should be calibrated against the results of [Koopmans et al., 2020] to verify the results.

Input data		2
Methods	pre-processing	1
	method, analysis, processing	1
	computational environment	1
	visualisation	2
Results		1

Table 4.2: Legend flowchart. Values are ranging from 0 minimum towards 3 maximum reproducability

5

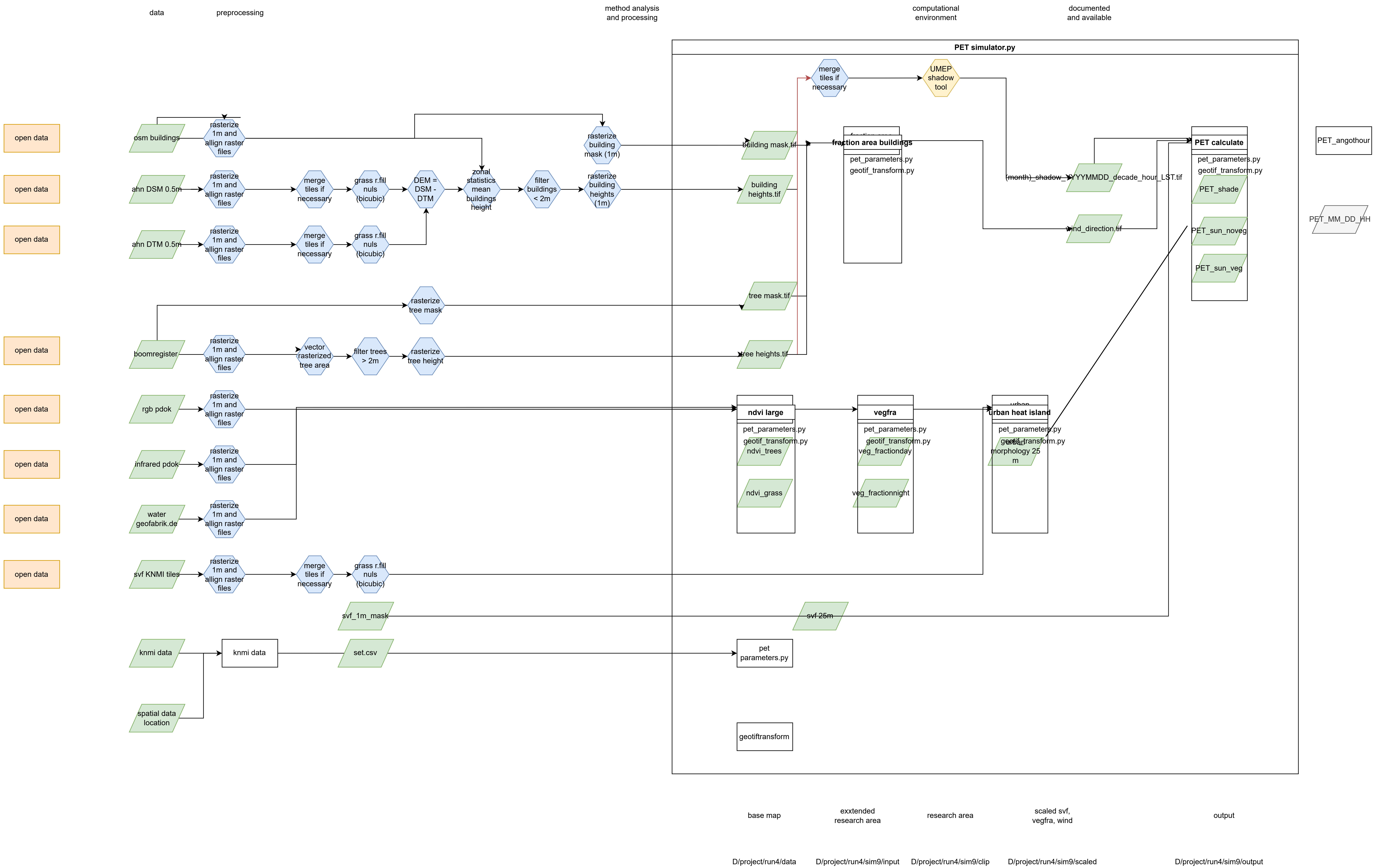
PETs simulator

5.1. Computational workflow

For the improvement of the code of Sytse Koopmans the decision is made to improve the reproducability for each step of the steps from input datasets, methods and results. The flowchart for the advanced refactored PET calculator is shown in Figure 5.1. The refactored Python code is displayed in Appendix B. In Appendix H the original code is displayed as a reference. Appendix C showcases the User Manual, and Appendix **??** presents step-by-step the extended research input files of Wageningen.

Input datasets

For the input files, an upgrade is made through using open-source accessible input. For the calibration of the code for the Wageningen test case the boomregister is used [NEO and Geodan, 2024]. For the Rotterdam test case [diensten Rotterdam, 2023] which is an open source of the municipality of Rotterdam. In the future, a detection method of tree classification could be used. Also, the modifications of the climate data by [KNMI, 0000] can now be generated by the Python script pysolar1.py which are taken as input for pet_parameters.py and also the retrieval of the Sky view factor geotiff maps are retrieved through the API link through the code get_svf.py. For the processing part, the decision was made to make an integral user interface to link the Python files with each other via one driver Python file. With Qt Designer, the link is made to create one graphical user interface in QGIS since this is the platform urban planners use the most for working at multiple scales [Lawhead, 2018]. Therefore this report created a QGIS plugin called PET simulator which can be downloaded via GIT, more explanation is in Appendix C. Therefore the computation kernel, in the code the Python file is called pet_simulator, is integrated into the QGIS plugin for third-party users to use. The link between Python and QGIS is made by the graphical user interface supporter Qt designer. Furthermore, there is a refactoring of the parameters which are used in each Python file which functions as classes. These files are geotiff_transform.py for the georeferencing towards arrays and vice versa. Additionally, pet parameters are introduced to standardize the input parameters like static factors, such as the research area coordinates and the cell size and block size of the wind computation, as well as climate dynamic factors retrieved from the [KNMI, 0000]. This makes it more understandable for other software developers. Through the decision to integrate the computational workflow with the integration of the visual representation environment of QGIS, the workflow process is more understandable and can be modified for other test cases in the Netherlands as well.



data
preprocessing
method analysis and processing
computational environment
documented and available
PET simulator.py
open data
osm buildings
ahn DSM 0.5m
ahn DTM 0.5m
boomregister
rgb pdok
infrared pdok
water geofabrik.de
svf KNMI tiles
knmi data
spatial data location
rasterize 1m and allign raster files
merge tiles if necessary
grass r.fill nuls (bicubic)
DEM = DSM - DTM
zonal statistics mean buildings height
filter buildings < 2m
rasterize building heights (1m)
rasterize building mask (1m)
rasterize tree mask
vector rasterized tree area
filter trees > 2m
rasterize tree height
svf_1m_mask
set.csv
building mask.tif
building heights.tif
tree mask.tif
tree heights.tif
UMEP shadow tool
fraction area buildings
pet_parameters.py
geotif_transform.py
(month)_shadow_YYYYMMDD_decade_hour_LST.tif
wind_direction.tif
PET calculate
PET_shade
PET_sun_noveg
PET_sun_veg
PET_angothour
PET_MM_DD_HH
ndvi large
ndvi_trees
ndvi_grass
vegfra
veg_fractionday
veg_fractionnight
urban heat island
urban morphology 25 m
svf 25m
pet parameters.py
geotiftransform
base map
exxtended research area
research area
scaled svf, vegfra, wind
output
D/project/run4/data
D/project/run4/sim9/input
D/project/run4/sim9/clip
D/project/run4/sim9/scaled
D/project/run4/sim9/output

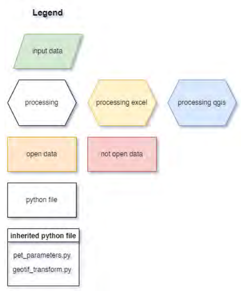


Figure 5.2: Legend flowchart refactored

Processing

To calculate the urban morphology heat attribution, we need to compute svf averaging fraction, and vegetation fraction averaging which are depended on the wind direction. Self-evident, this is also required for the wind computation. This requires handling the necessary input files for extended research outcomes. We will create clips of the basis maps for the research area needed to compute for each wind direction, which are called extended research areas. Detailed procedures for the 1000x1000m research area of Wageningen will be explained in this chapter.

Figure 5.3: Research area 1000x1000 m white (output), extended research area 1500x2100 m black (input), and base map RGB 4000x4000 m (data).

In the program refactoring, the parameterized block size for modeling the vegetation fraction, sky view fraction, and wind computation is taken into account. Instead of the variable averaging of approximately 25m and 35m, the window frames were adjusted to a standard block size of 25m. The wind averaging window is shown in Figure 5.4 , and the sky view factor averaging window is shown in Figure 5.5.

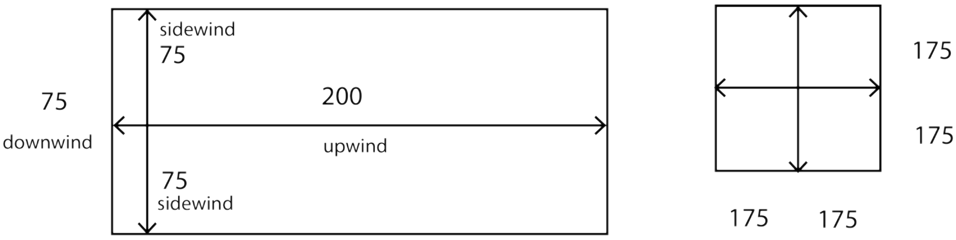


Figure 5.4: Wind averaging footprint from roughness layer to 1.2m wind speed factors field.

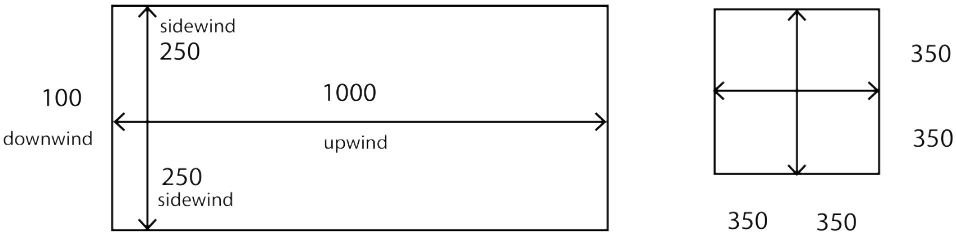
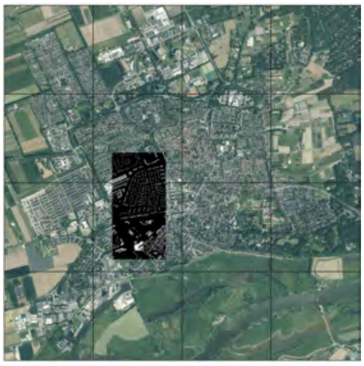


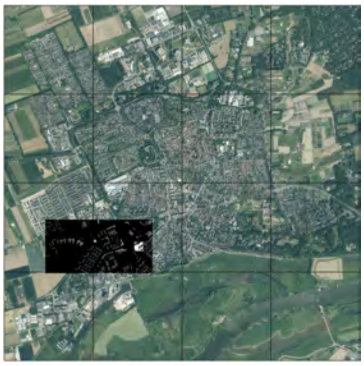
Figure 5.5: Vegetation fraction and sky view factor averaging footprint for determining the UHI max depending on the wind direction.

(a)

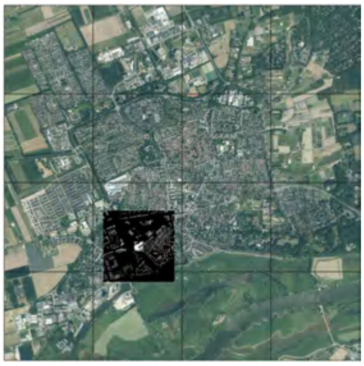


(b) block size 1, N

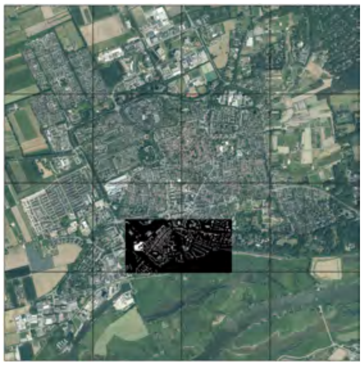
(c)



(d) block size 1, W



(e) block size 1, no wind



(f) block size 1, E

(g)

(h) block size 1, S

(i)

Figure 5.6: Wind direction for the research area of 100x100m.

In pet_simulator this is made possible through

Listing 5.1: clip to extended research area window code snippet

```
    # clip to extended research window
    outputfile = f'{self.spatial.directory_out}input\\{self.spatial.label}_{name}.tif'
    bounds = (self.spatial.xmin-xleft, self.spatial.ymin-ydown, self.spatial.xmax+xright, self.spatial.ymax+yup)
    gdal.Warp(outputfile, intiff, outputBounds=bounds)
```

Listing 5.2: visualisation tif for in the report code snippet

```
    self.TifToJPG(self.spatial.directory_out, 'input', f'{self.spatial.label}_{name}', large=True)
```

Listing 5.3: writing the array to text file code snippet

```
            if self.dlg.checkBox.checkState():
    ArrayWriteG(f'{self.testin}', f'{self.spatial.label}_{name}', f'{outputfile
        }')
```

Listing 5.4: adding layer to QGIS project

```
    raster_layer = QgsRasterLayer(outputfile, f'{name}', 'gdal')  # input from
        file
    if not raster_layer.isValid():
    print('Error: Invalid raster layer.')
    else:
    QgsProject.instance().addMapLayer(raster_layer)
```

Wind calculation

python code: fraction_area_buildings_treeregr
input: buildings_mask, buildings_height, trees_mask, trees_height
output: wind_2d

Original input data building map vector 1(m) `https://www.geofabrik.de/` open data lidar height raster 1(m) https://www.ahn.nl/ahn-4 open data tree map vector 1 (m) `https://diensten.rotterdam.nl/arcgis/rest/services/SB_Infra` or bgt download `https://app.pdok.nl/lv/bgt/download-viewer/` open data Input data for code buildings_mask Figure D.4, buildings_height Figure D.5, trees_mask Figure D.7, trees_height Figure D.6 Output fraction_area_buildings_treeregr wind_2d Figure 5.7 on blocksize scale The building mask scaled area

Listing 5.5: The building mask scaled area code snippet

```
    building_area = np.mean(mask_building_fine[istart: iend + 1, jstart: jend +
         1])
    if building_area > 1e-2:
    building_height[i,j] = np.mean(building_height_fine[istart: iend + 1,
        jstart: jend + 1]) / building_area
    mask_building[i, j] = 1.0
    tree_area = np.mean(mask_tree_fine[istart: iend + 1, jstart: jend + 1])
    if tree_area > 1e-2:
    tree_height[i, j] = np.mean(tree_height_fine[istart: iend + 1, jstart: jend
         + 1]) / tree_area
    mask_tree[i, j] = 1
```

Building weight scaled with wind

Listing 5.6: Building weight scaled with wind code snippet

```
    if wind_on:
    if WE: # east-west or west-east wind
    for m in range(istart, iend + 1, 1):
    for n in range(jstart, jjend, 1):
    building_weight[i, j] += abs(building_height_fine[m, n + 1] -
        building_height_fine[m, n]) * 0.5
    tree_weight[i, j] += abs(tree_height_fine[m, n + 1] - tree_height_fine[m, n
        ]) * 0.5
    else: # north-south or south-north wind
    for n in range(jstart, jend + 1, 1):
    for m in range(istart, iiend, 1):
    building_weight[i, j] += abs(building_height_fine[m + 1, n] -
        building_height_fine[m, n]) * 0.5
    tree_weight[i, j] += abs(tree_height_fine[m + 1, n] - tree_height_fine[m, n
        ]) * 0.5
```

Building weight scaled without wind

Listing 5.7: Building weight scaled without wind code snippet

```
else: # no wind
for m in range(istart, iend + 1, 1):
for n in range(jstart, jjend, 1):
building_weight[i, j] += abs(building_height_fine[m, n + 1] - building_height_fine[m, n]) * 0.5
tree_weight[i, j] += abs(tree_height_fine[m, n + 1] - tree_height_fine[m, n]) * 0.5

for n in range(jstart, jend + 1, 1):
for m in range(istart, iiend, 1):
building_weight[i, j] += abs(building_height_fine[m + 1, n] - building_height_fine[m, n]) * 0.5
tree_weight[i, j] += abs(tree_height_fine[m + 1, n] - tree_height_fine[m, n]) * 0.5
```

Building front computation

Listing 5.8: Building front computation code snippet

```
# calculate building and tree fronts for a cell using its window (1 no blockage, 0 fully blocked)
tree_front = 0
building_front = 0

for m in range(istart, iend + 1, 1):
for n in range(jstart, jend + 1, 1):
building_front += building_weight[m, n] * buildingfactor
tree_front += tree_weight[m, n] * treefactor
```

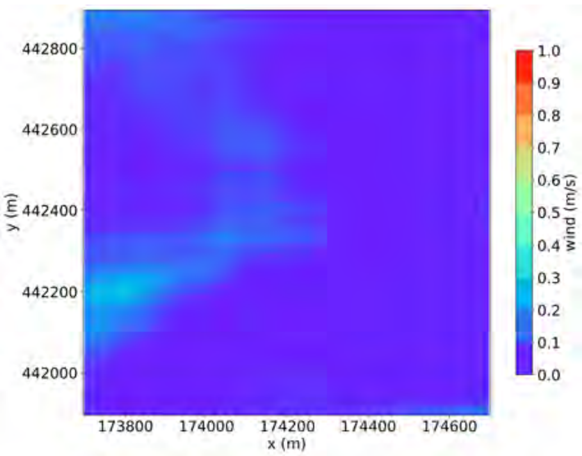


Figure 5.7: Output files on research area.

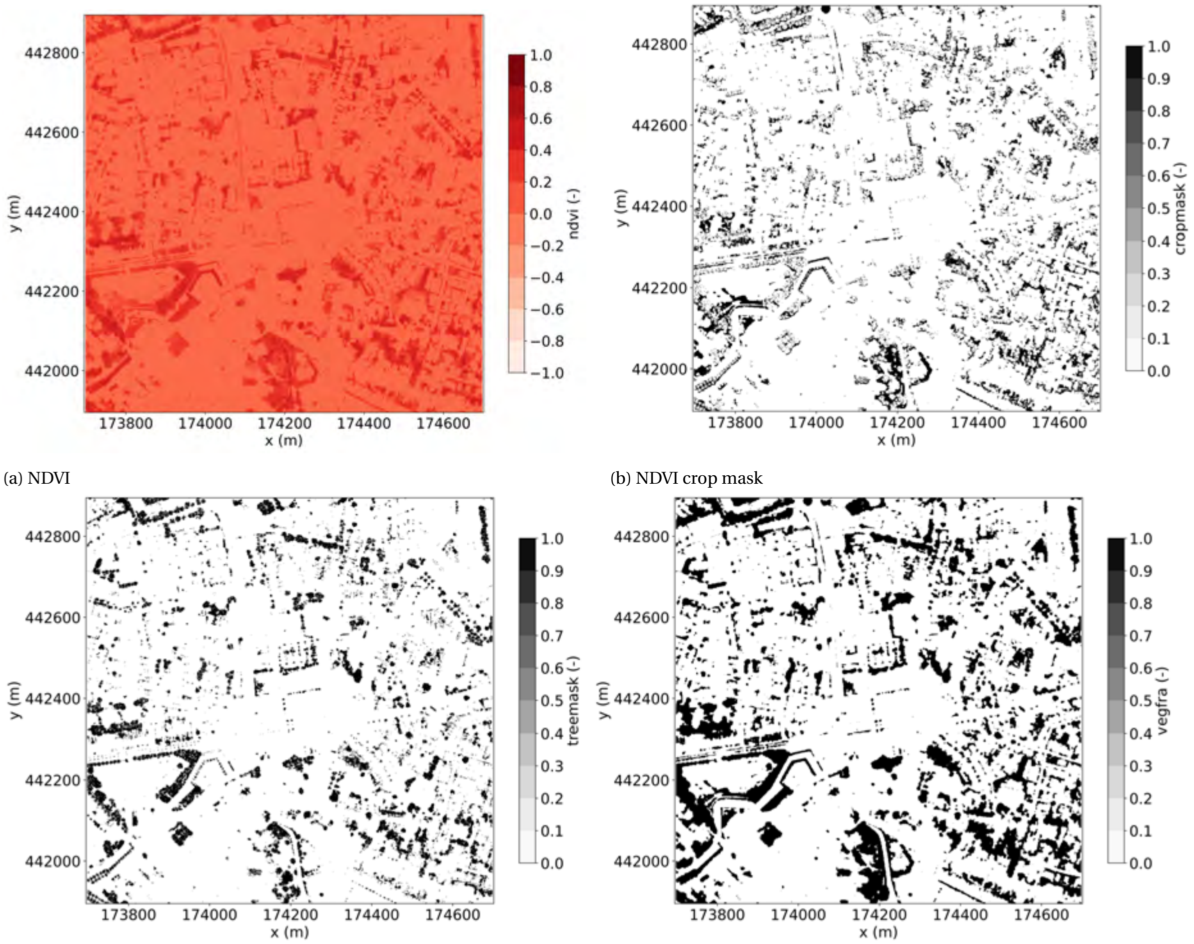
Ndvi large calculation

python code: ndvi_infra_large
input: rgb, infr, water_mask, tree_mask
output: ndvi, vegfra, ndvi_crop_mask, ndvi_tree_mask

Original input data aerial photo (RGB) raster 1(m) 0.25 `https://www.pdok.nl/` open data, near infrared (NIR) raster 1(m) 0.25 `https://www.pdok.nl/` open data water map vector 1(m) `https://www.geofabrik.de/` Input data code input rgb Figure D.12, infr Figure D.11, water_mask Figure D.10, tree_mask Figure D.7 output NDVI Figure 7.21a, NDVI crop mask Figure 7.21b NDVI tree mask Figure 7.21c, vegetation fraction Figure 7.21d

```
lufo_rgb, meta = GeotifToArray(rgb, 3)
```

```
    lufo_infr, meta = GeotifToArray(infr, 3)
    r = lufo_rgb[:, :, 0].astype(int)
    g = lufo_rgb[:, :, 1].astype(int)
    b = lufo_rgb[:, :, 2].astype(int)
    infr = lufo_infr[:, :, 0].astype(int)
    ndvi_infr = (infr - r) / (infr + r)
    ndvi_infr[ndvi_infr < 0] = 0
    arr = ndvi_infr
```



(a) NDVI

(b) NDVI crop mask

(c) NDVI tree mask

(d) vegetation fraction

Figure 5.8: Intermediate output files on research area.

Fveg vegetation footprint calculation

python code: vegetation_footprint
input: vegfra
output: vegfra_2d

Input data from ndvi_infra_large Figure 7.21d Output data vegfra_2d Figure 5.9 with blocksize resolution in this case 25m

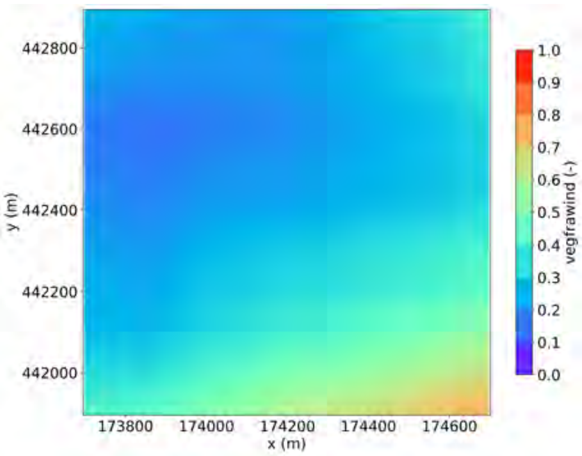


Figure 5.9: Scaled vegetation fraction wind.

Sky view factor footprint calculation

python code: skyviewfactor_footprint
input: skyview_factor
output: skyview_2d

Original input data Sky-view factor map raster 1m `https://api.dataplatform.knmi.nl/open-data/v1` open data with API provided with the code get_skyview.py Input data code skyviewfactor_footprint 1m resolution Figure D.8 Output data skyview_2d with blocksize resolution in this case 25m Figure 5.10

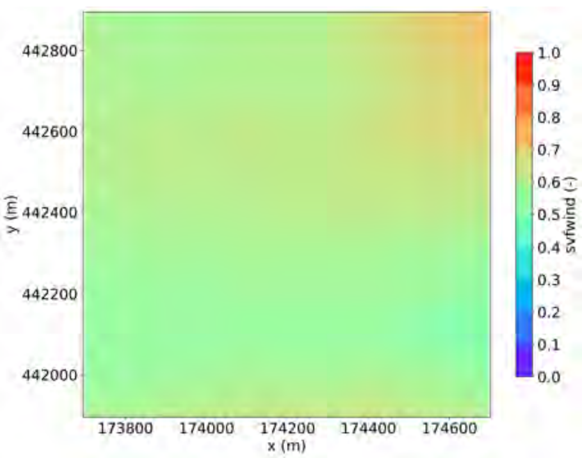


Figure 5.10: Scaled sky view fraction wind.

Urban heat Island Max calculation

python code: urban_heat
input: vegfra_wind, svf_wind
output: urban_heat

Input data vegfra_wind with 25m resolution from vegetation_footprint Figure **??**, svf_wind with 25m resolution from skyviewfactor_footprint Figure **??** Output data Urban heat morphology geospatial contribution Figure 5.11

Listing 5.9: urban heat morphology code snippet

```
    uhi *= 2
```

```
    uhi = uhi - vegfra - svf
    factor = (S * (Tmax - Tmin) ** 3 / U) ** (1 / 4)
    uhi *= factor
    im3 = ArrayToGeotif(uhi, meta)
```

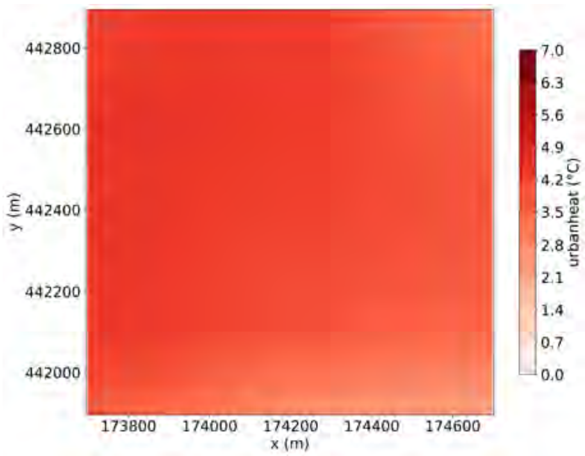


Figure 5.11: Urban heat.

PET calculation

python code: pet_calculate
input: shadow, urban_heat, wind_2d, svf, svf_mask, ndvi_crop_mask, ndvi_tree_mask
output: pets

Input data Shadow Figure D.13, urban_heat Figure 5.11, wind_2d Figure 5.7, svf Figure D.8, svf_mask Figure D.9, ndvi_crop_mask Figure 7.21b, ndvi_tree_mask Figure 7.21c. Output Hourly Physiological Equivalent Temperature Figure 5.12 The calculation of the PET could be performed on the day with sun and without sun areas, as well as on places were there is vegetation present. As well as in the night situation.

Listing 5.10: wet bulb temperature code snippet

```
    Ta = uhi[:] * diurnal + TT
    Tw = TT * np.arctan(0.15198 * (RH + 8.3137) ** 0.5) + np.arctan(TT + RH) -
        np.arctan(
    RH - 1.676) + 0.0039184 * RH ** 1.5 * np.arctan(0.023101 * RH) - 4.686
```

Listing 5.11: scaling factor multiplied with wind speed 60m height code snippet

```
    wind = ((wind - 0.125) * 0.5829 + 0.125) * FF
    wind[wind < 0.5] = 0.5
```

Listing 5.12: PET calculation day situation code snippet

```
# day
if Q > 0:
sun_temp, meta = GeotifToArray(im1, 1)
sun = sun_temp * (1 - trees_2m[:])
PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw + 0.015
     * svf[:] * Qdif +
0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1 - sun[:]) * svf_mask
    [:]
PETveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.sin(
sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
sunalt / 360 * 2 * np.pi) + 0.546 * Bveg + 1.94 * svf[:]) * mask_vegfra[:] *
    sun[:] * svf_mask[:]
```

```
PETnoveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.sin(
sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
sunalt / 360 * 2 * np.pi) + 0.546 * Bnoveg + 1.94 * svf[:]) * (1 - mask_vegfra
    [:]) * sun[:] * svf_mask[:]
PET = PETshade + PETveg + PETnoveg
```

Listing 5.13: PET calculation night situation code snippet

```
        # night
    else:
    PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw +
        0.015 * svf[:] * Qdif
    + 0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1 - sun[:]) *
        svf_mask[:]

    PET = PETshade
```

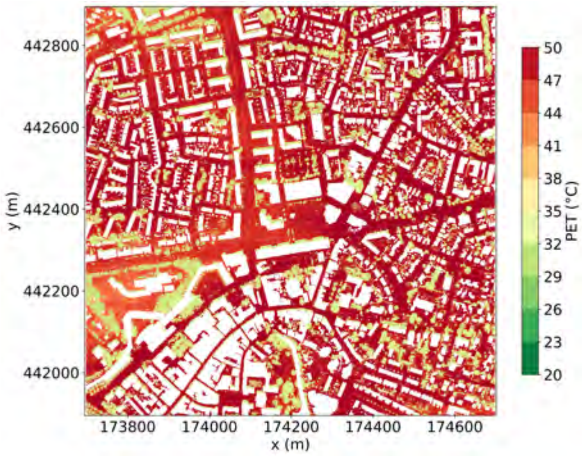


Figure 5.12: PET.

5.2. PET simulator

Pet simulator eventually combines all the python files together as shown in Appendix B. It combines geotif.creator places the retrieved arrays as georeferenced tifs in EPSG:28992 - Amersfoort / RD New coordinates of x and y. pet_parameters which combines the static data of the location and the dynamic weather data of the generated weather.py csv files see subsection dynamic parameters and static parameters. With addGtiffLayer the link between QGIS and python is made to immediately publish all the in-between results to the visualisation software of QGIS.

Listing 5.14: Example of invocation of PET calculate in PET simulator.py

```
    from .algorithm.pet_calculate import PET_calculate
    flag = []

    # import geotiff
    flag.append(time.perf_counter())
    name = f'Shadow_{self.weather.year}{self.weather.month:02d}{self.weather.
        day:02d}_{self.weather.hour:02d}{self.weather.min:02d}_LST'
    name = "Shadow_20150701_1400_LST"

    im1 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.
        label}_{name}.tif') # small
```

```
    im2 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.label}_urban_heat.tif') # small
    im3 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.label}_wind.tif') # small
    im4 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_svf.tif')  # small
    im5 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_svf_mask.tif')  # small
    im6 = self.clipper(self.spatial.directory_out, 'output', f'{self.spatial.label}_ndvi_crop_mask.tif')  # small
    im7 = self.clipper(self.spatial.directory_out, 'output', f'{self.spatial.label}_ndvi_tree_mask.tif')  # small

    # calculate
    flag.append(time.perf_counter())
    im8 = PET_calculate(self.spatial, self.weather, im1, im2, im3, im4, im5, im6, im7) # small #nonetype

    # add layer and write geotiffs
    flag.append(time.perf_counter())
    name = 'pets'
    self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im8, driver, root)
    im1 = im2 = im3 = im4 = im5 = im6 = im7 = None
    self.dlg.label_17.setText('checked')
    flag.append(time.perf_counter())
    self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.label}_pets')
    flag.append(time.perf_counter())
```

5.3. User interface

Qt designer

The software that is necessary for running the PET simulation is chosen to be Python and QGIS. QGIS is meant as the graphical user interface for users to visualize the (in-between) results. Python is required to do the computations. In order to make the link between Python and QGIS, Qt designer is necessary. Qt designer is developed to create a plugin in QGIS that users can use to create their maps. This will enhance the reproducability of the (in-between) results for several stakeholders in the process. More in depth explanation is stated in the user manual Chapter C. Some libraries that are required are GDAL package to make the georeferenced projections from matrix calculations to the preferred georeference system. For running the script also the plugin UMEP is still used in QGIS in order to create the shadow files from the DSM-DTM for each hour. In the plugin installer it is possible to install UMEP and the UMEP processing, see B. In order to create reproducability the plugin is developed see Figure 5.13. This is created through the Plugin builder tool in QGIS. Each QGIS version is compiled with Qt designer. Qt Designer is designed to create a graphical user-interface that is compatible with python and QGIS. For the plugin three windows are developed: first the static parameters of the built environment location and the reference to the directory of the file locations on the device of the basis maps. Next window will read the csv files for each run and hour simulation. Also, after clicking on this window the input files of the extended research area are generated and visualised in QGIS. The third window will indicate the processing of the several python files for eventually generating the required research area maps with in-between results and end results.

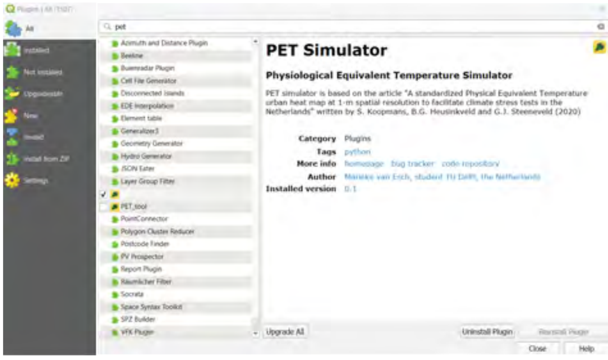


Figure 5.13: Qgis plugin screen PET Simulator plugin.

Parameters for spatial information and weather conditions

In the plugin's parameter section, both dynamic climate data and static data are utilized. Dynamic data can be obtained from the URL `https://www.knmi.nl/nederland-nu/klimatologie/uurgegevens` of a nearby weather location. Wageningen Herwijnen has been selected as the weather location for Wageningen, while the weather station Rotterdam is used for Rotterdam (see Figure 5.14). When looking for a summer day (above 20 °C) or a heatwave day (above 25 °C), an overview of the summer months is needed.

Figure 5.14: Weather stations Netherlands retrieved from.

In the case of a heatwave date, the 1st of July is chosen because it is above 25 degrees. For the validation, it is necessary to have good comparison material. The boxes that need to be checked at the knmi `https://www.knmi.nl/nederland-nu/klimatologie/uurgegevens` are YYYYMMDD, TT, FF, dd, Q, U. YYYYMMDD represents the month, day, hour. TT represents the atmospheric temperature (°C). FF represents the wind speed (in 0.1 m/s) averaged over the last 10 couple of minutes of the past hour. dd represents the wind direction (°) averaged over the last 10 couple of minutes of the past hour 360=North, 90=East, 180=South, 270=West, 0=calm, 990=changeable. Q represents the Global irradiation (in J/cm2)/h. U represents the Relative humidity (%). As mentioned earlier, the Python code weather.py generates the CSV files for the dynamic data used in the script. Table 5.1 is the dynamic weather data necessary for the CSV file.

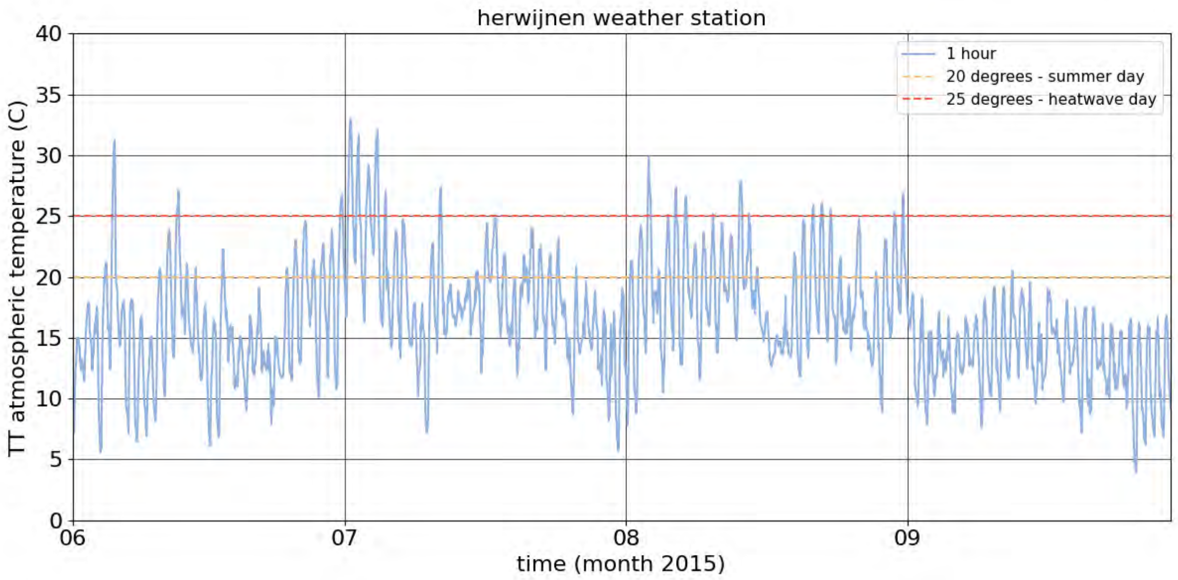


Figure 5.15: T atmospheric temperature for Rotterdam in the months june till september 2023 (Data retrieved from KNMI [0000] post-processed by author

Table 5.1: Table dynamic data Wageningen 1 juli 2015

hour	TT	FF	dd	Q	Qdif	sunalt	RH	wind	WE	winddir	day	diurnal	Tmin	Tmax
9	26.4	5	100	711.1111	142.2222	48	52	TRUE	TRUE	E	day	0.007	21.1	33
10	28	6	100	794.4444	158.8889	55.3	48	TRUE	TRUE	E	day	0.03	21.1	33
11	29.8	6	100	855.5556	171.1111	60.1	44	TRUE	TRUE	E	dayt	0.05	21.1	33
12	31.2	6	130	868.0556	173.6111	60.9	35	TRUE	TRUE	E	day	0.07	21.1	33
13	32.1	5	130	825	165	57.4	37	TRUE	TRUE	E	day	0.11	21.1	33
14	32.8	4	140	743.0556	148.6111	50.8	35	FALSE	FALSE	S	day	0.16	21.1	33
15	32.9	5	120	629.1667	125.8333	42.5	37	TRUE	TRUE	E	day	0.23	21.1	33
16	33	4	130	491.6667	144.1848	33.4	37	TRUE	TRUE	E	day	0.31	21.1	33
17	33.2	4	120	338.8889	132.3261	24.2	39	TRUE	TRUE	E	day	0.42	21.1	33
18	30.9	3	100	130.5556	113.7764	15.2	45	TRUE	TRUE	E	day	0.56	21.1	33

For the static parameters ymax 442895 xmax 174698 ymin 441895 and xmin 173698 are chosen.

Listing 5.15: Link between Plugin and code for static and dynamic parameters

```
        def importdata(self):

    self.spatial.directory_in = self.dlg.lineEdit_3.text()
    self.spatial.directory_out = self.dlg.lineEdit_2.text()
    self.spatial.label = self.dlg.lineEdit_1.text()

    with open(f'{self.spatial.directory_out}set.csv', 'r') as fp:
    lines = fp.readlines()
    lines = [line.strip() for line in lines]
    lines = [line.split(',') for line in lines]
    self.spatial.station = lines[3][1]
    self.spatial.ymax = float(lines[4][1])
    self.spatial.xmax = float(lines[5][1])
    self.spatial.ymin = float(lines[6][1])
    self.spatial.xmin = float(lines[7][1])
    self.spatial.cellsize = float(lines[8][1])
    self.spatial.blocksize = float(lines[9][1])
    self.spatial.Resize()
    self.weather.TT = float(lines[10][1])
    self.weather.FF = float(lines[11][1])
```

```
    self.weather.dd = float(lines[12][1])
    self.weather.wind, self.weather.WE, self.weather.winddir = wind_direction(
        self.weather.dd, self.weather.FF)
    self.weather.Q = float(lines[13][1])
    self.weather.Qdif = float(lines[14][1])
    self.weather.sunalt = float(lines[15][1])
    self.weather.RH = float(lines[16][1])
    self.weather.diurnal = float(lines[21][1])

    self.dlg.lineEdit_7.setText(f'{self.spatial.ymax}')  # north
    self.dlg.lineEdit_6.setText(f'{self.spatial.xmax}')  # east
    self.dlg.lineEdit_5.setText(f'{self.spatial.ymin}')  # south
    self.dlg.lineEdit_4.setText(f'{self.spatial.xmin}')  # west
    self.dlg.lineEdit_17.setText(f'{self.spatial.cellsize}')  # south
    self.dlg.lineEdit_16.setText(f'{self.spatial.blocksize}')  # west
    self.dlg.lineEdit_3.setText(f'{self.spatial.directory_in}')
    self.dlg.lineEdit_2.setText(f'{self.spatial.directory_out}')
    self.dlg.lineEdit_1.setText(f'{self.spatial.label}')
    self.dlg.lineEdit_15.setText(f'{self.spatial.station}')
    self.dlg.lineEdit_8.setText(f'{self.weather.TT}')
    self.dlg.lineEdit_9.setText(f'{self.weather.FF}')
    self.dlg.lineEdit_10.setText(f'{self.weather.dd}')
    self.dlg.lineEdit_12.setText(f'{self.weather.Q}')
    self.dlg.lineEdit_13.setText(f'{self.weather.Qdif}')
    self.dlg.lineEdit_14.setText(f'{self.weather.sunalt}')
    self.dlg.lineEdit_11.setText(f'{self.weather.RH}')
```

Simulation process

Each layer will be put in the QGIS project to link the computational environment of Python computation towards the visualization environment of QGIS.

Results

The results of the PET simulator are compared with the model of Koopmans in the next chapter. However, the end product, the Physiological Equivalent Temperature map, displays heat stress. To communicate the results properly, the principles from [Bertin, 2011] serve three main functions: recording information, communicating information, and processing or simplifying information. The recorded information presents calculated Physiological Equivalent Temperatures. These are the visualization of the calculated maps of PET, which are in a continuous colored way of 18 degrees to 50 °C PET. To communicate the data effectively to third parties, classification of the PET for different levels of thermal perception and physiological stress on human beings is required according to [Höppe, 1999]. This classification is shown in a table, using semantic coloring to express slight cold and no thermal stress with cool tones, and the slight to extreme heat stress with warm to extremely dark colors, reflecting the level of heat stress that people can handle (see Table 5.2).

PET	Thermal perception	Grade of physiological stress	color code
13 - 18 °C	Slightly cool	Slight cold stress	
18 - 23 °C	Comfortable	No thermal stress	
23 - 29 °C	Slightly warm	Slight heat stress	
29 - 35 °C	Warm	Moderate heat stress	
35 - 41 °C	Hot	Strong heat stress	
>41 °C	Very hot	Extreme heat stress	

Table 5.2: Temperature and corresponding thermal perception

6

Physiological Equivalent Temperature verification

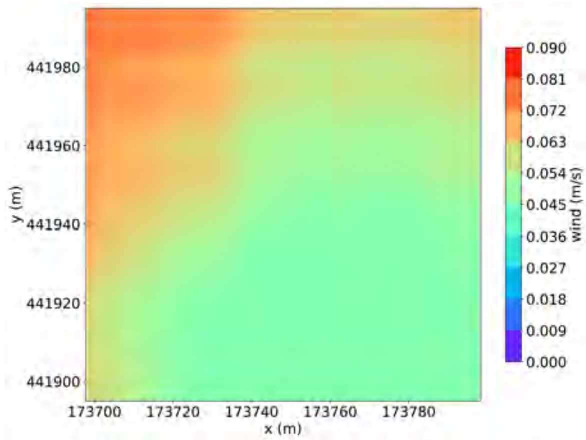
Sensitivity analyses

A wind sensitivity analysis is carried out to understand the performance of the wind calculation for the scalability of the data for use by urban designers, for the test case of urban environments such as cities. Therefore, an analysis of the robustness of the newly introduced varying block sizes of 1m, 5m, and 25m of wind is carried out. The frontal area density factors are also tested to validate the block size change as a granularity option and accuracy validation with the output data from Koopmans. Koopmans used the original 35m block size for his wind calculations. Runtime and scalability are also discussed for use by urban designers.

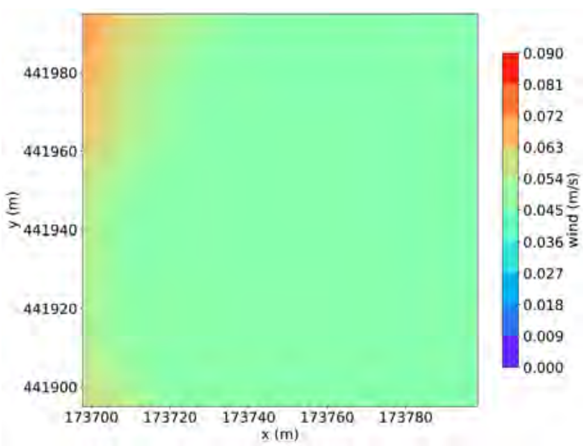
6.1. Wind direction

First, the wind direction will be evaluated. The QGIS plugin can generate different outcomes on the Windfield based on the clip size of the extended areas. A closer look at the wind could be found in the comparison of the 100x100m area as mentioned with the wind direction in the previous chapter Figure 6.1. Figure 6.1b showcases wind coming from the North, Figure 6.1c showcases wind coming from the East, Figure 6.1d showcases wind coming from the South, Figure 6.1e showcases wind coming from the West, Figure 6.1f showcases wind wind is <2.5m/s therefore no wind.

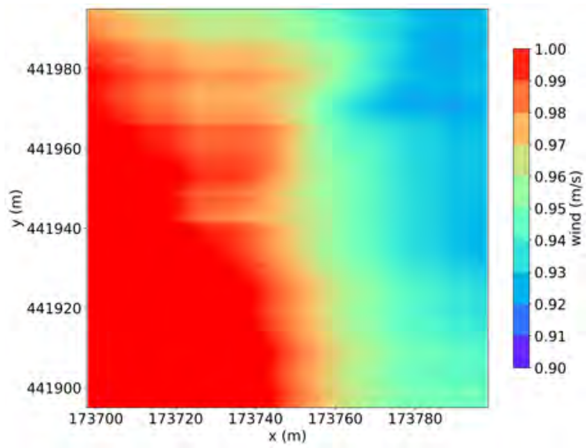
(a) rgb image



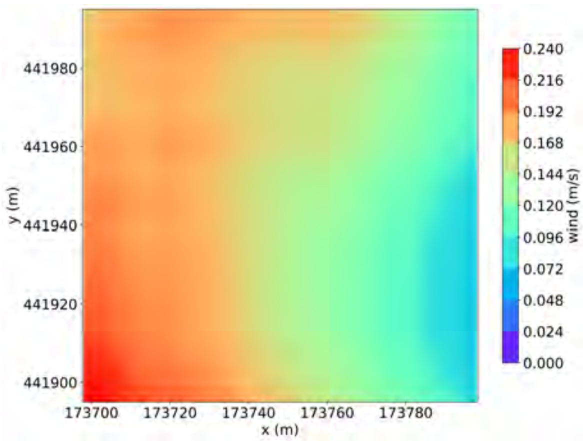
(b) north



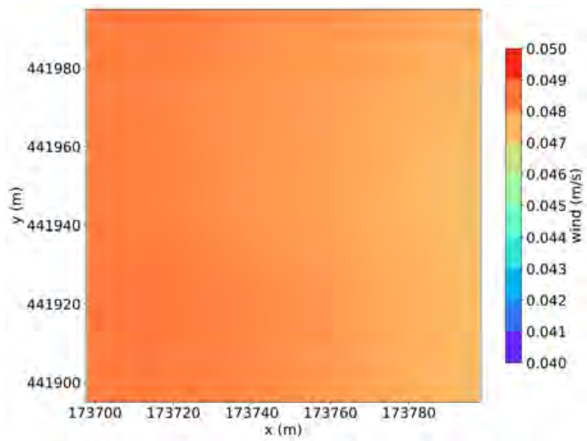
(c) east



(d) south



(e) west



(f) nowind

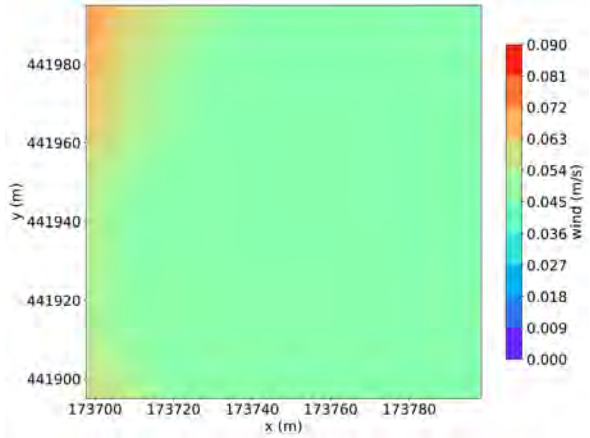
Figure 6.1: Different wind directions files on research area 100x100m

It is evident that the wind scaling factor varies based on the frontal area from different wind directions. North, East, and no wind result in a high frontal area, reducing the impact of wind on PET. In the southern research area, there is less frontal area, leading to higher wind scaling factors. The colors are adjusted to the minimum and maximum values of each field.

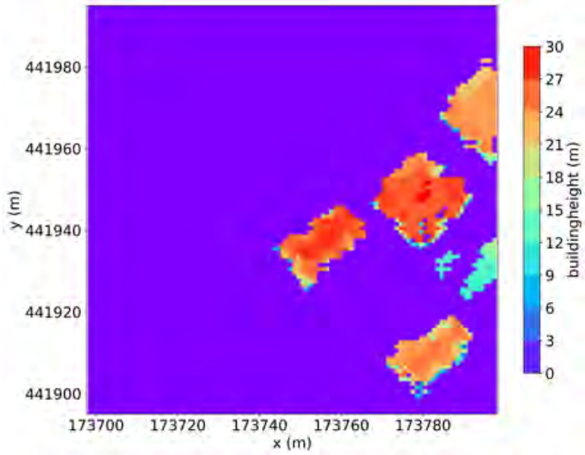
6.2. Block size

For the robustness of the data and accuracy, the built-in function is the block size, which can vary from 1m, 5m, and 25m approximately. For the area of 100x100m, this is the overview. As can be seen in the figure, the scale of the values will be averaged in the same manner. Only the 1m is very accurate based on the whole computation field, which leads to more spikes and fluctuations of the output data, whereas the 5m and 25m are averaged more, thus containing a smoother field.

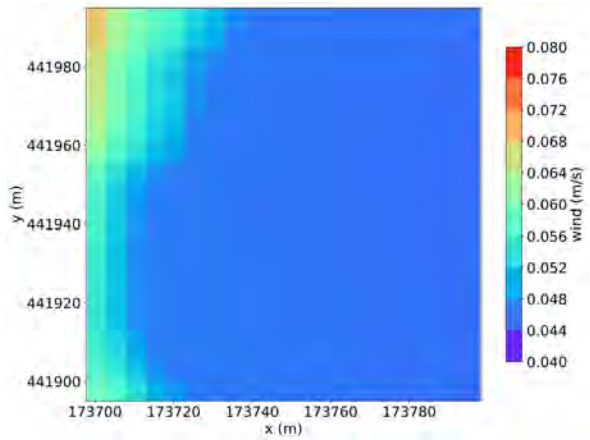
(a) rgb image



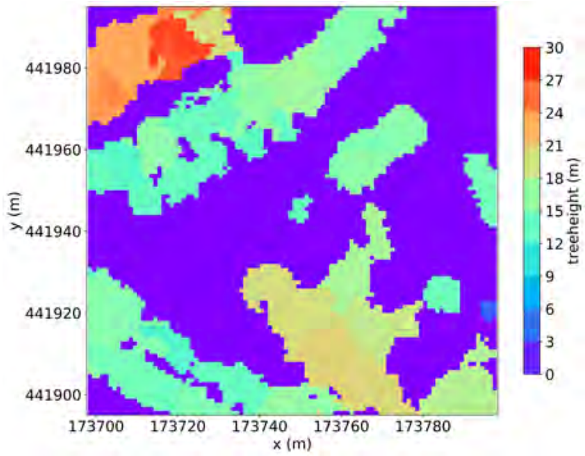
(b) block size 1m



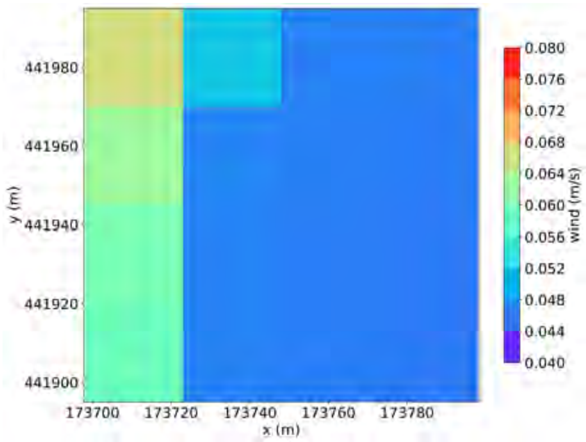
(c) building height



(d) block size 5m



(e) tree height



(f) block size 25m

Figure 6.2: Research area 100 x 100 m, eastern wind.

The robustness of the block size scale is evaluated by computing the mean square error, root mean square error, and the r2 value to assess data similarity.

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{6.1}$$

Comparing the blocksize between 1 and 5 there was a high correlation with the r2 score of 0.641.

Listing 6.1: MSE between blocksize 1 and blocksize 5 100x100 area

```
    R**2 = 0.9863
    MSE = 4.495 *  10**(-7)
    RMSE = 0.00067
```

Listing 6.2: MSE between blocksize 5 and blocksize 25 100x100 area

```
    R^2 = 0.8999
    MSE = 2.0222 *  10**(-5)
    RMSE = 0.004497
```

Listing 6.3: MSE between blocksize 1 and blocksize 25 100x100 area

```
    R^2 = 0.8558
    MSE = 2.119  * 10**(-5)
    RMSE = 0.004
```

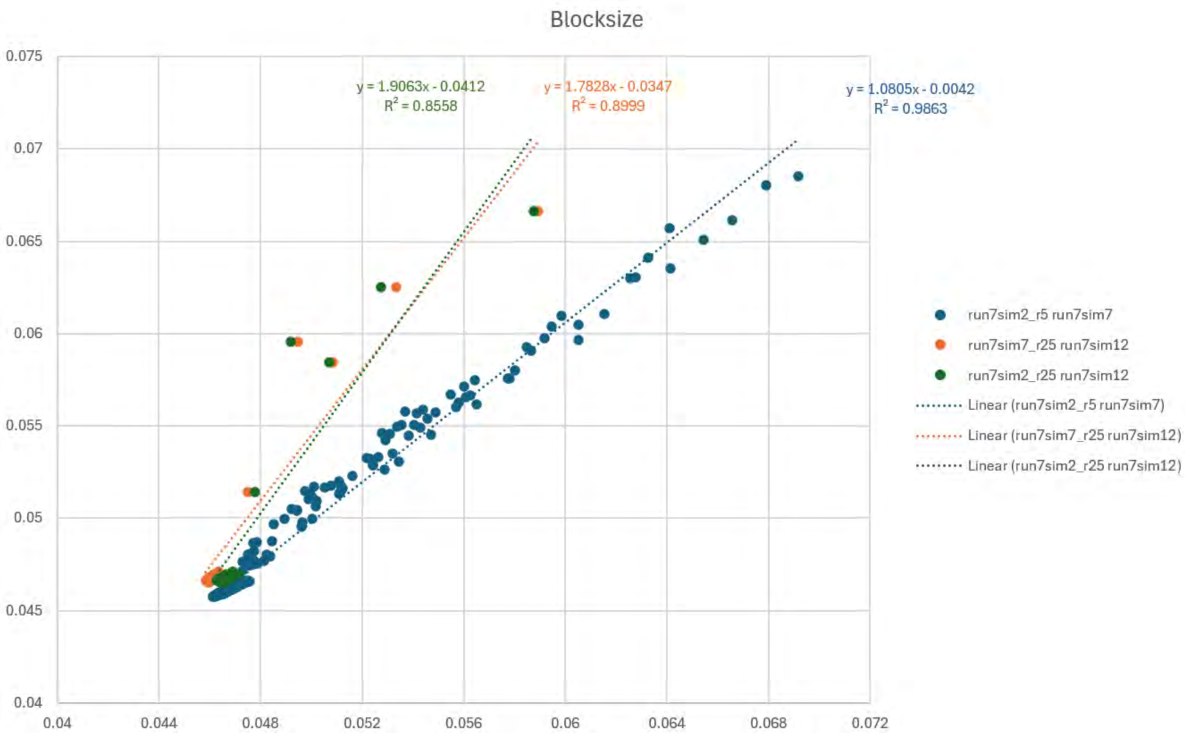


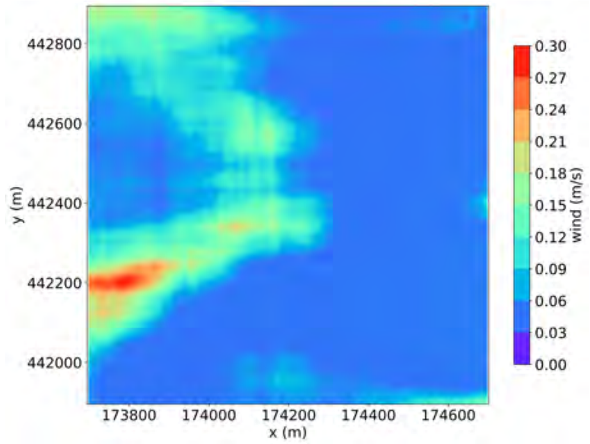
Figure 6.3: Trendline time data block size 5m

By comparing the different resolutions, it was found that the deviations in MSE are of similar small magnitude. Additionally, a high degree of correlation is observed between the different block sizes, approaching nearly 1.

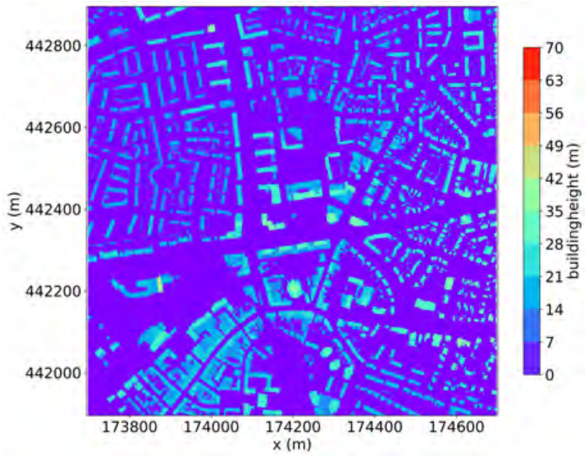
6.3. Block size comparison 1000x1000 research area

For the robustness of the data and the accuracy the built-in function is the block size which can vary from 1m, 5m and 25m approximately. For the area of 1000x1000m this is the overview. As could be seen in the figure winds the scale of the values will be averaged the same. Only the 1m is very accurate based on the whole computation field which leads to higher and lower values, whereas Figure 6.4

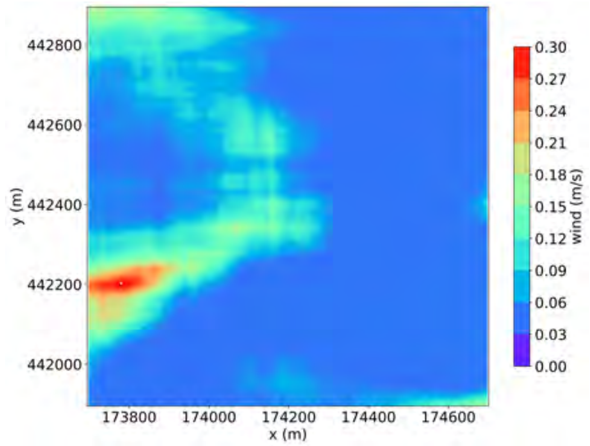
(a) rgb image



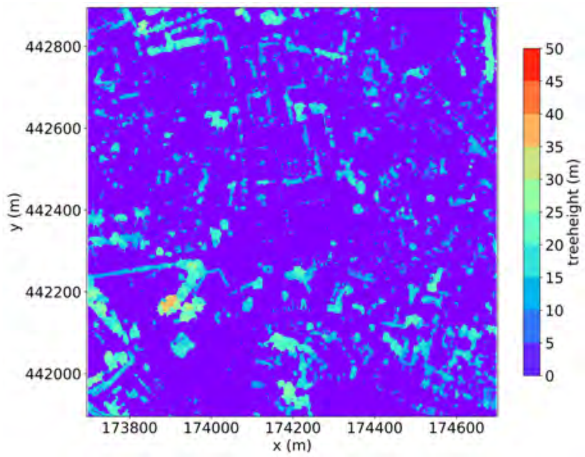
(b) block size 1m



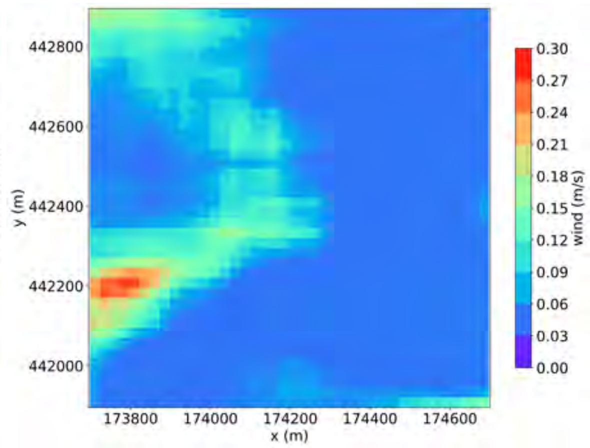
(c) building height



(d) block size 5m



(e) tree height



(f) block size 25m

Figure 6.4: Research area 1000 x 1000 m, eastern wind.

Listing 6.4: MSE between blocksize 1 and blocksize 5 1000x1000 area

```
    R**2 = 0.97
    MSE = 0.000126688
    RMSE = 0.011255589
```

Listing 6.5: MSE between blocksize 5 and blocksize 25 1000x1000 area

```
    R**2 = 0.2978
    MSE = 0.046554846
    RMSE = 0.21576572
```

Listing 6.6: MSE between blocksize 1 and blocksize 25 1000x1000 area

```
    R**2 = 0.2836
    MSE = 0.04469523
    RMSE = 0.211412447
```

After comparing the different resolutions, we found that the deviations in mean squared error (MSE) are of similar small magnitude. Additionally, there is a high degree of correlation between the block sizes of 1m and 5m, approaching nearly 1. However, the comparisons between 5m and 25m, as well as the comparison between 1m and 25m, do not show high correlation and have higher MSE and root mean square error (RMSE) offset.

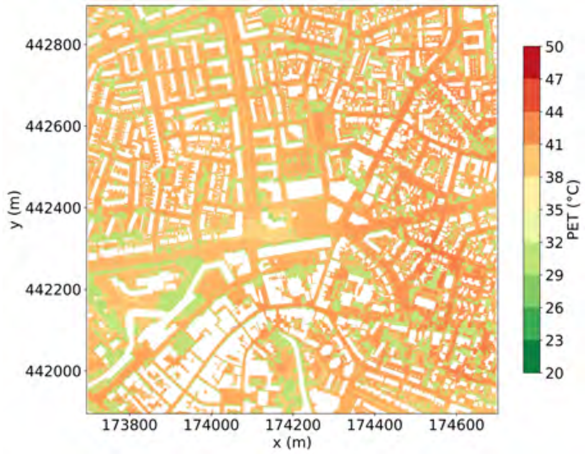
6.4. Frontal area

Wind sensitivity for different wind surface density factors is seen in Table 6.1. The normal frontal density factors (fdf) as mentioned in the formula of the wind are 0.6 respectively for the buildings and 0.3 * 0.9 crown size height for the trees. A comparison is made to validate the outcome of different fdf.

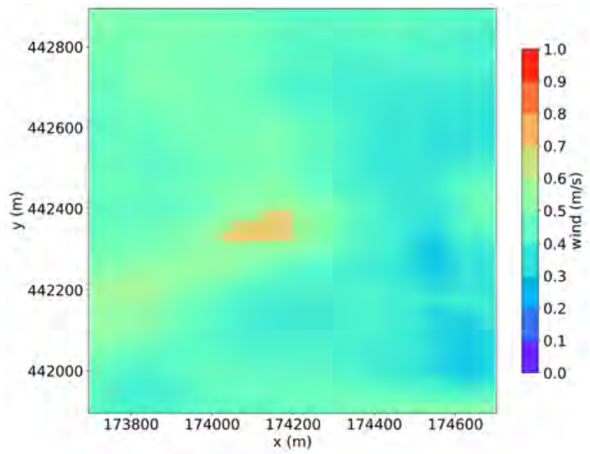
	fdf buildings	**fdf trees**
run8sim16	0.6	0.27
run8sim17	0.1	0.27
run8sim18	0.9	0.27
run8sim19	0.6	0.03
run8sim20	0.6	0.6
run8sim21	0.3	0.27

Table 6.1: Frontal density factors.

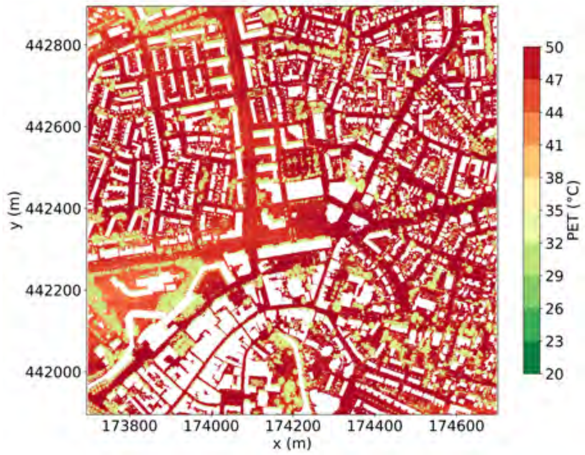
This is showcased by the following figures Figure 6.5 for the building fdf and Figure 6.6 for the varying tree fdf.



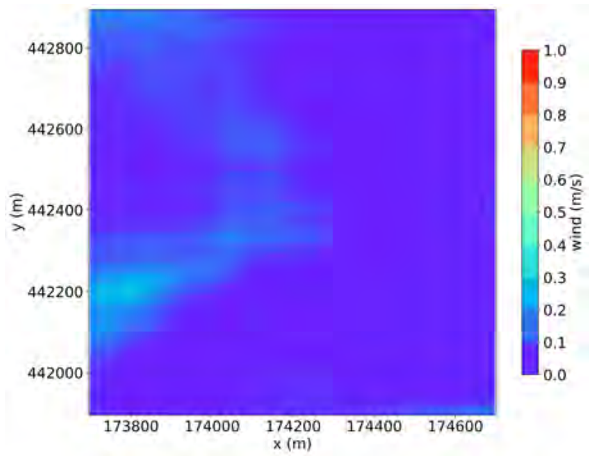
(a) PET fdf building 0.1 fdf tree 0.27



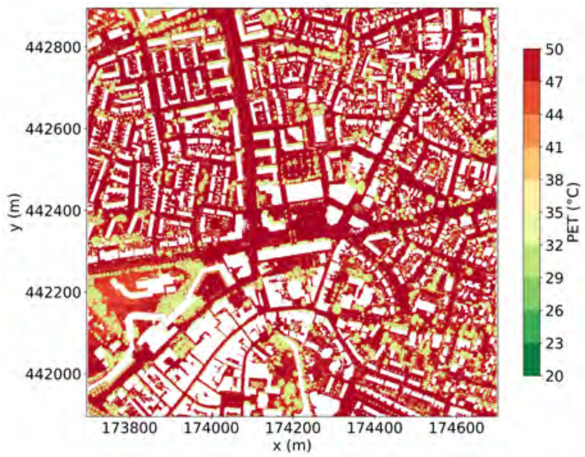
(b) eastern wind fdf building 0.1 fdf tree 0.27



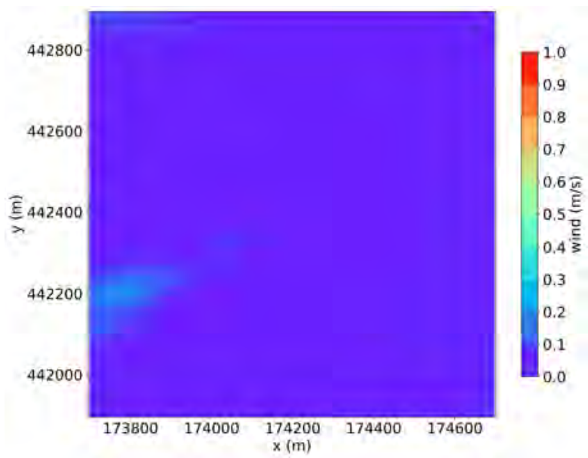
(c) PET fdf building 0.6 fdf tree 0.27



(d) eastern wind fdf building 0.6 fdf tree 0.27

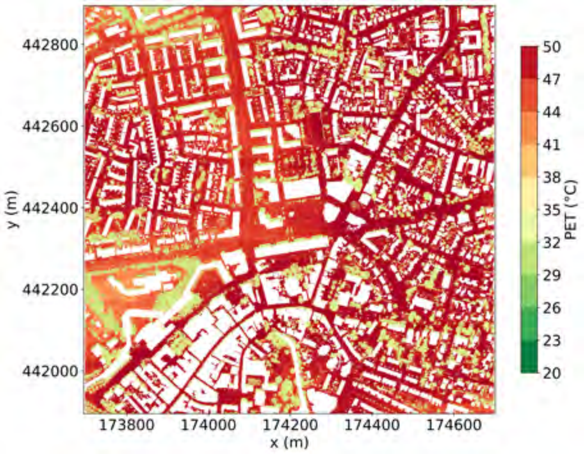


(e) PET fdf building 0.9 fdf tree 0.27

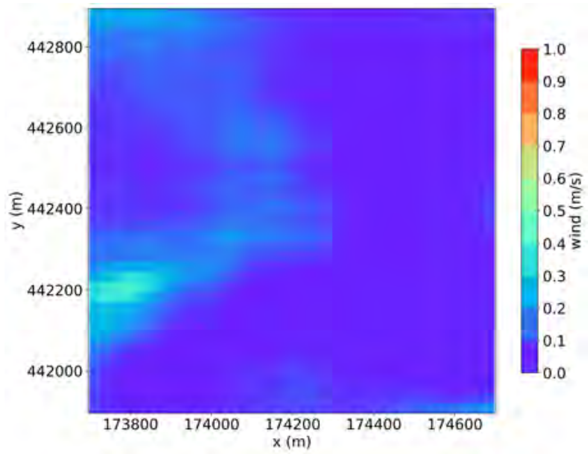


(f) eastern wind fdf building 0.9 fdf tree 0.27

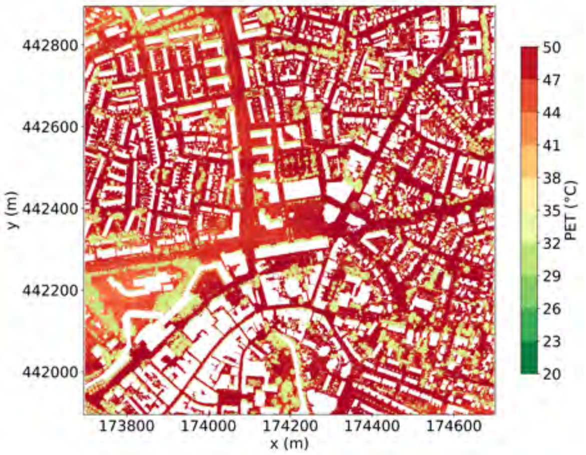
Figure 6.5: Sensitivity analyses frontal density factor buildings.



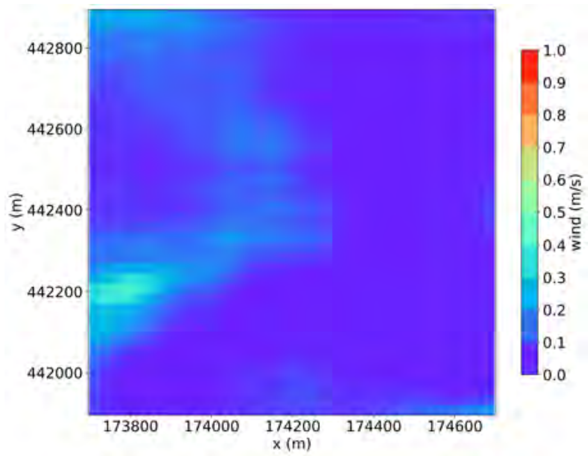
(a) PET fdf building 0.6 fdf tree 0.03



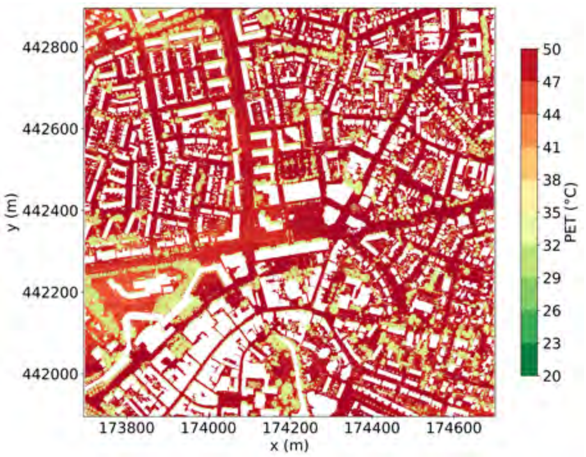
(b) eastern wind fdf building 0.6 fdf tree 0.03



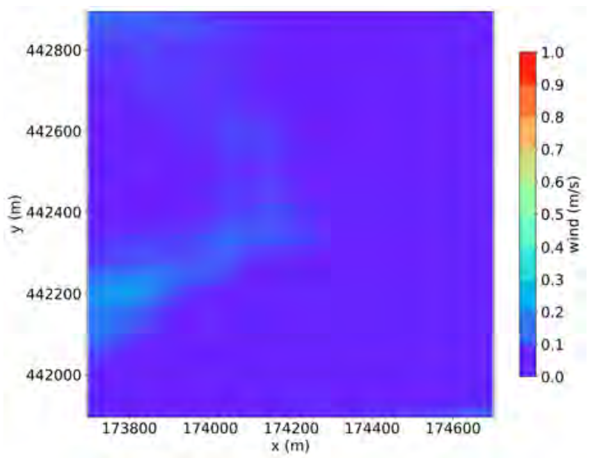
(c) PET fdf building 0.3 fdf tree 0.27



(d) eastern wind fdf building 0.3 fdf tree 0.27



(e) PET fdf building 0.6 fdf tree 0.6



(f) easterns wind fdf building 0.6 fdf tree 0.6

Figure 6.6: Sensitivity analyses frontal density factor trees.

The influence of a lower fdf of the buildings is causing much more roughness in the wind calculation and therefore the PET, in comparison to the fdf for the trees. For the calibration section of this chapter the fdf of buildings is adjusted because of the difference of blocksize width of 25 meters to the original 35 meters approxiametely of the code of Koopmans et al. (2020).

6.5. Scalability

To talk about the usability of the program for urban planners, the computation time according to the research area is important. See Figure 6.7 and Figure 6.8. A lot of computing time is spent on the wind computation. To examine the functionality across various scales, Tables 6.2, 6.3, 6.4 were created with an extrapolation to

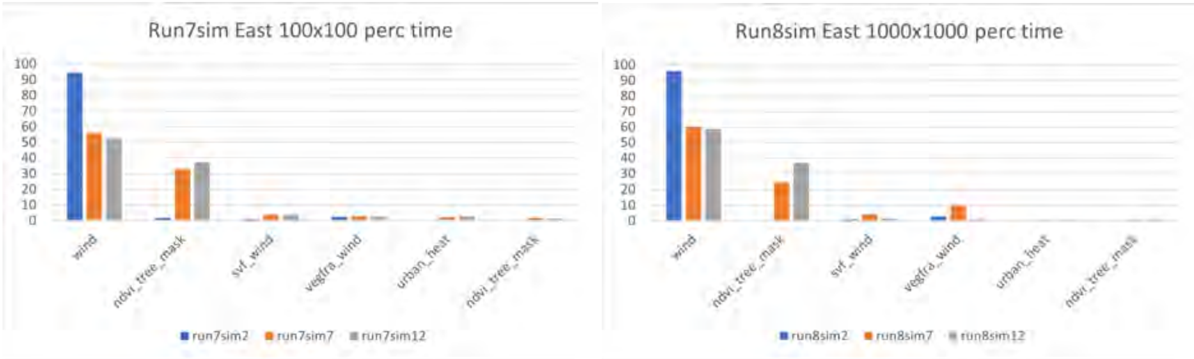


Figure 6.7: Percentage time

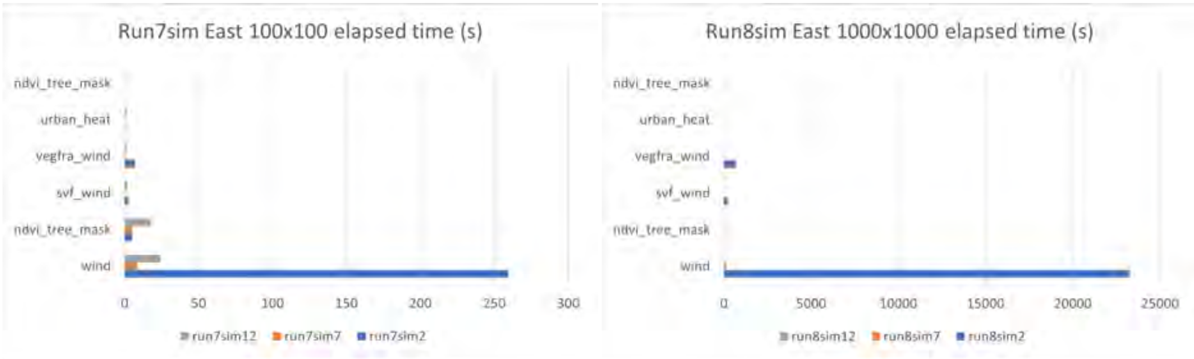


Figure 6.8: Elapsed time (s)

match the size of Rotterdam.

Table 6.2: Blocksize 1 wind computing time extrapolated for South-Holland and the Netherlands

blocksize1	area (m2)	t (s)	t(min)	t(h)	t(day)
	100	259.344	4.3224	0.07204	0.0030017
	200	1074.287	17.90478	0.298413	0.0124339
	500	5846.292	97.4382	1.62397	0.0676654
	1000	23250.96	387.516	6.4586	0.2691083
	2000	47847.62	797.4603	13.29101	0.5537919
Rotterdam	10000	255537.4	4258.957	70.98261	2.9576089

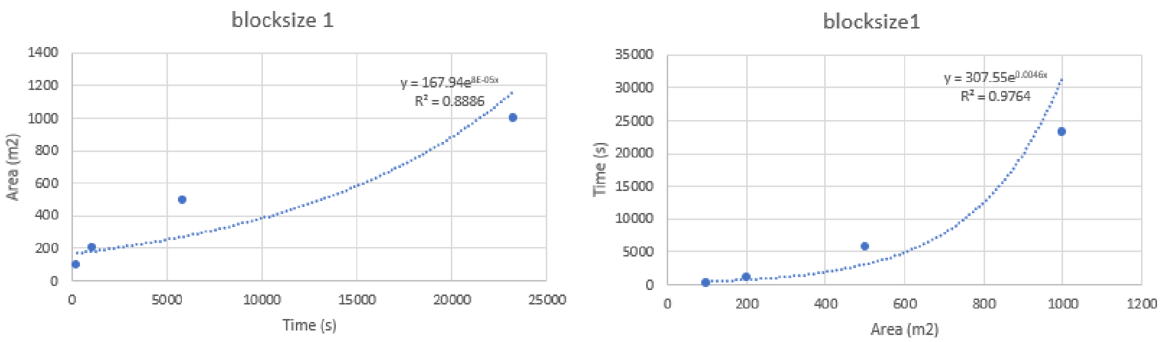
Table 6.3: Blocksize 5 wind computing time extrapolated for South-Holland and the Netherlands

blocksize5	area (m2)	t (s)	t(min)	t(h)	t(day)
	100	8.424	0.1404	0.00234	0.0000975
	200	12.165	0.20275	0.003379	0.0001408
	500	28.615	0.476917	0.007949	0.0003312
	1000	107.562	1.7927	0.029878	0.0012449
	2000	211.9064	3.531774	0.058863	0.0024526
Rotterdam	10000	1103.338	18.38897	0.306483	0.0127701

Table 6.4: Block size 25 wind computing time extrapolated for South-Holland and the Netherlands

block size25	area (m2)	t (s)	t(min)	t(h)	t(day)
	100	24.439	0.407317	0.006789	0.0002829
	200	8.164	0.136067	0.002268	9.449E-05
	500	13.716	0.2286	0.00381	0.0001588
	1000	27.198	0.4533	0.007555	0.0003148
	2000	62.82	1.047	0.01745	0.0007271
Rotterdam	10000	257.718	4.2953	0.071588	0.0029828

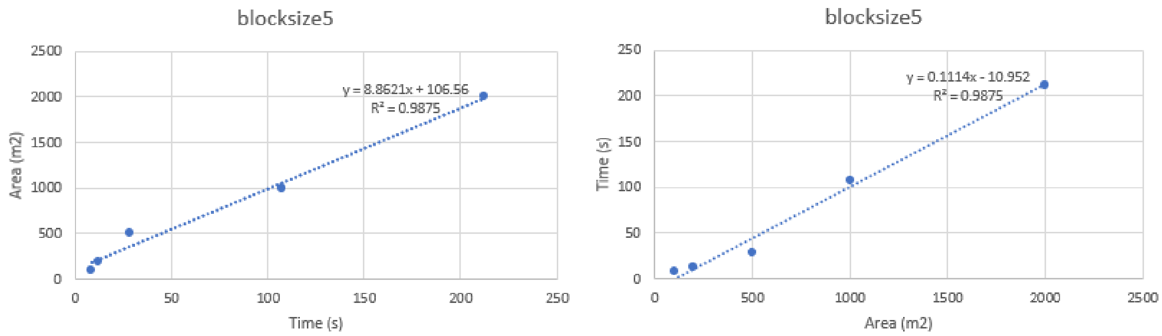
The trend lines for the different block sizes could then be plotted.



(a) rgb changed

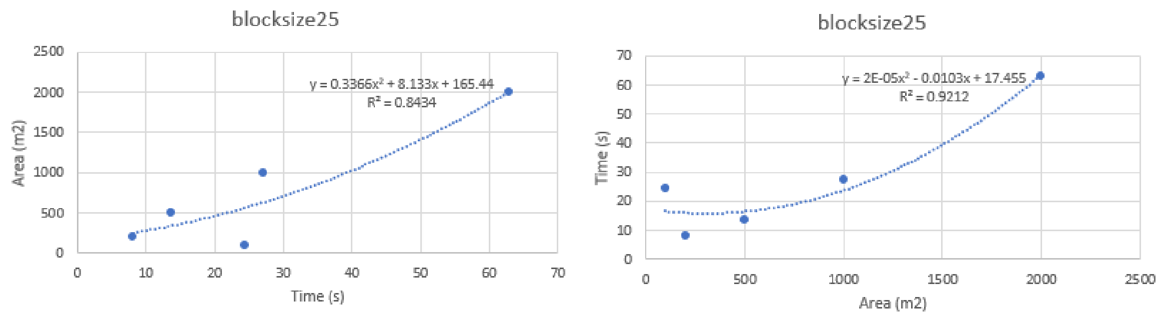
(b) infr changed

Figure 6.9: Trendline time data block size 1m



(a) rgb changed (b) infr changed

Figure 6.10: Trendline time data block size 5m

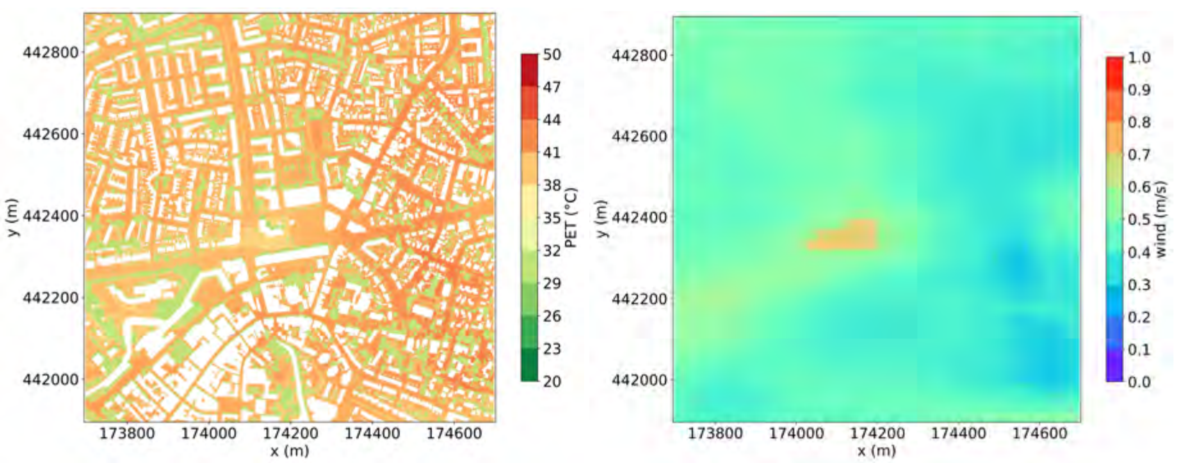


(a) rgb changed (b) infr changed

Figure 6.11: Trendline time data block size 2m

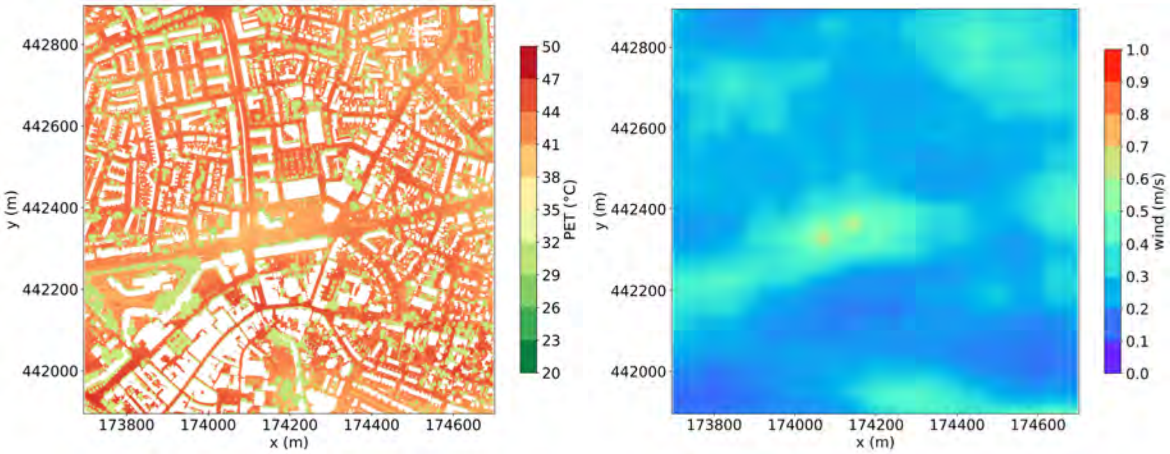
As can be seen from the trend lines, block size 25 is the most useful option for larger-scale calculations as opposed to 1m or 5m. However, at a scale of 100x100m, it will take a little longer to average the data. However, as can be seen from the accuracy of the data, some precise information will be lost. Therefore, for sizes smaller than Rotterdam, the block size of 5m will be more favorable for checking the performance of the public space.

6.6. Calibration of the code



(a) PET fdf building 0.16 fdf tree 0.27 (b) Eastern wind fdf building 0.16 fdf tree 0.27

Figure 6.12: Calibrated frontal density factor trees.



(a) PET fdf building 0.6 fdf tree 0.27

(b) Eastern wind fdf building 0.6 fdf tree 0.27

Figure 6.13: Outcome Sytse Koopmans

The updated model was validated using the findings from Koopmans et al. (2020), and adjustments were made to the five factors based on this validation. Setting the fdf of the buildings to 0.16 and maintaining the fdf of the trees at 0.27 resulted in a high r2 score in the final PET map.

Listing 6.7: MSE between blocksize 25 and blocksize 35 wind outcome sytse 1000x1000 area

```
    R^2 = 0.7803
    MSE = 0.0774
    RMSE = 0.2782
```

Listing 6.8: MSE between blocksize 25 and blocksize 35 PET outcome sytse 1000x1000 area

```
    R^2 = 0.6399
    MSE = 137
    RMSE = 11.7
```

One possible explanation for the discrepancy in the R2 value between the 25m and 35m block sizes, despite a small MSE, is the scaling of wind data values to 0 and 1, respectively. However, there are differences in the R2 value that may be attributed to adjustments in the fdf of the buildings for the 35m block size. Additionally, during the refactoring phase, there may have been a downgrade in the modeling of wind values to either a 35m or 25m resolution, which could be addressed in future improvements. Therefore the positive correlation of the PET is not too strong at the moment. To make it better there should be an evaluation of the other in-between measures as well. The removal or addition of buildings also impacts the generated PET map, contributing to the MSE deviation in the mean square error. Despite this, the R2 value remains consistent with the final result.

The wind sensitivity on blocksize resulted different resolutions with high positive correlation. for the operability for larger scale research areas the wind sensitivity with the blocksize of 25 meter could be easily used to determine a brief overview of the results. Due to the refactoring the fdf factor of the buildings needed to be adjusted to a lower value to be calibrated with the end result. Refinement in the fdf building and other in-between steps in the process are required in order to come to a higher PET resemblance with the code of Koopmans (2020).

7
Physiological Equivalent Temperature application

The flowchart depicting the advanced refactored PET calculator can be found in Figure **??**. The refactored python code is available in Appendix B, while the User Manual is presented in Appendix C. Additionally, Chapter E details the step-by-step process of the extended research input files of Rotterdam North, which was utilized to calculate the PET for Bospolder Tussendijken [van Esch, 2024], see Figure 7.1. Other applications to determine the walkability of the place are described in Chapter K Walkability.

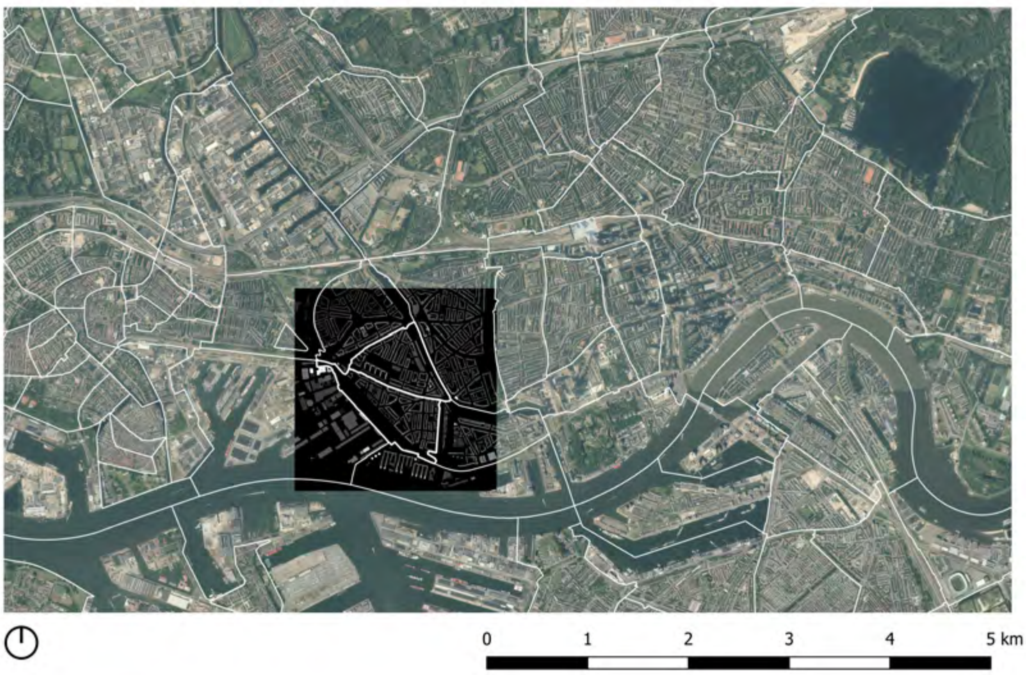


Figure 7.1: Location of Bospolder Tussendijken in Rotterdam

7.1. PET calculation

For the days to be modeled, an overview is made to depicts the days with a temperature above 20 °degrees and a day above 25 °C, see Figure 7.2 .

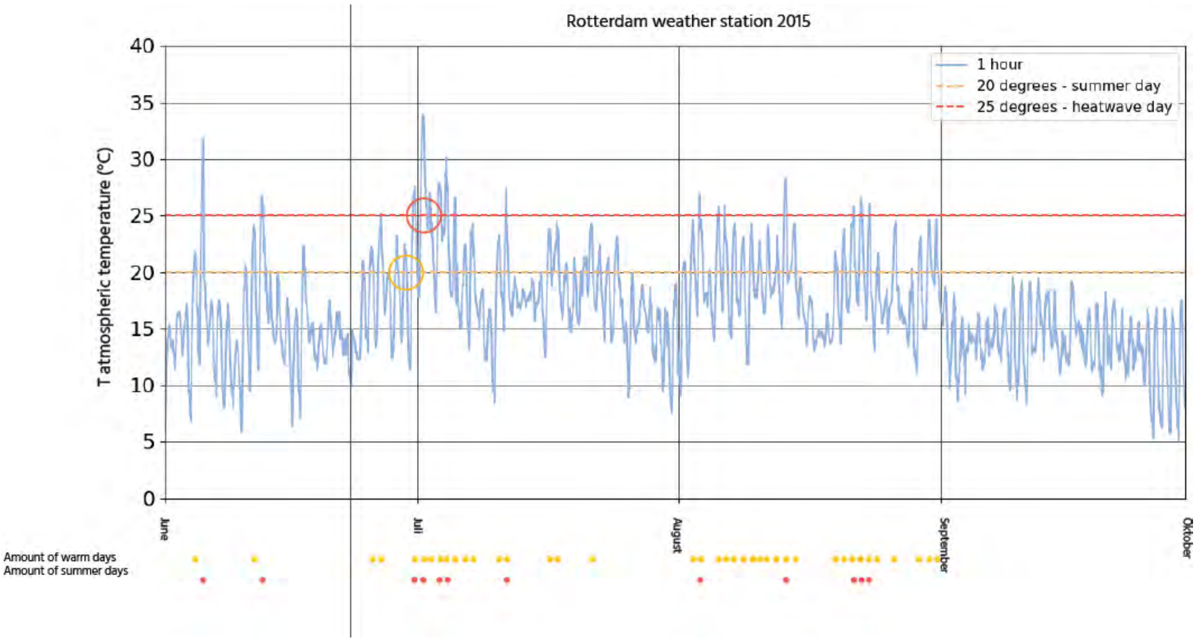


Figure 7.2: Fig. T atmospheric temperature for Rotterdam in the months june till september 2015 (Data retrieved from KNMI [0000] post-processed by author)

The chosen dates are the 1st of July and the 29th of June, see Table 7.1 and Table 7.2.

Table 7.1: Table dynamic data Rotterdam 1 juli 2015

hour	TT	FF	dd	Q	Qdif	sunalt	RH	wind	WE	winddir	day	diurnal	Tmin	Tmax
9	27.2	4	100	699.425	155.9823	48	45	TRUE	TRUE	E	day	0.007	23.7	34
10	29	5	100	808.84	154.012	55.3	43	TRUE	TRUE	E	day	0.03	23.7	34
11	30.3	7	90	865.625	169.524	60.1	39	TRUE	TRUE	E	day	0.05	23.7	34
12	31.8	6	110	865.625	176.726	60.9	32	TRUE	TRUE	E	day	0.07	23.7	34
13	32.5	5	110	821.305	169.524	57.4	29	TRUE	TRUE	E	day	0.11	23.7	34
14	33	5	120	745.13	158.998	50.8	30	TRUE	TRUE	E	day	0.16	23.7	34
15	33.8	5	120	634.33	143.5459	42.5	31	TRUE	TRUE	E	day	0.23	23.7	34
16	34	5	130	501.37	134.1004	33.4	29	TRUE	TRUE	E	day	0.31	23.7	34
17	33.8	5	130	351.79	121.1653	24.2	33	TRUE	TRUE	E	day	0.42	23.7	34
18	32.9	5	110	202.21	95.36945	15.2	36	TRUE	TRUE	E	day	0.56	23.7	34

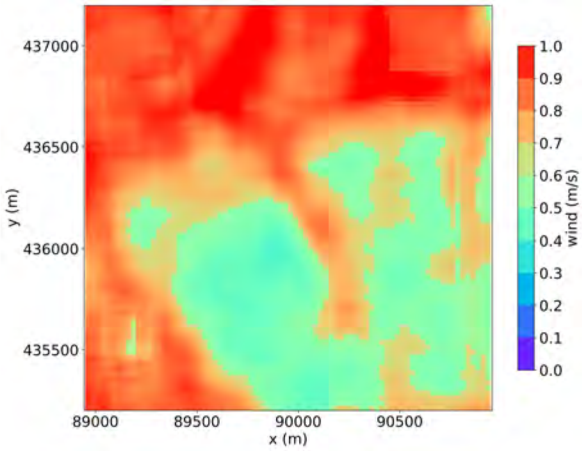
Table 7.2: Table dynamic data Rotterdam 29 june 2015

hour	TT	FF	dd	Q	Qdif	sunalt	RH	wind	WE	winddir	day	diurnal	Tmin	Tmax
9	20.5	4	270	559.54	278.8815	48	65	TRUE	TRUE	W	day	0.007	11.3	23.1
10	21.5	4	250	704.965	243.5441	55.3	57	TRUE	TRUE	W	day	0.03	11.3	23.1
11	22.5	4	270	738.205	261.424	60.1	58	TRUE	TRUE	W	day	0.05	11.3	23.1
12	21.3	4	270	735.435	271.0638	60.9	64	TRUE	TRUE	W	day	0.07	11.3	23.1
13	22	4	290	742.36	230.7026	57.4	64	TRUE	TRUE	W	day	0.11	11.3	23.1
14	21.7	3	270	646.795	245.0592	50.8	58	TRUE	TRUE	W	day	0.16	11.3	23.1
15	22	3	320	533.225	237.4175	42.5	53	TRUE	TRUE	N	day	0.23	11.3	23.1
16	21.2	3	350	368.41	228.7261	33.4	56	TRUE	TRUE	N	day	0.31	11.3	23.1
17	20.4	3	350	271.46	171.3269	24.2	57	TRUE	TRUE	N	day	0.42	11.3	23.1
18	19.9	2	350	210.52	89.52669	15.2	55	FALSE	TRUE	C	day	0.56	11.3	23.1

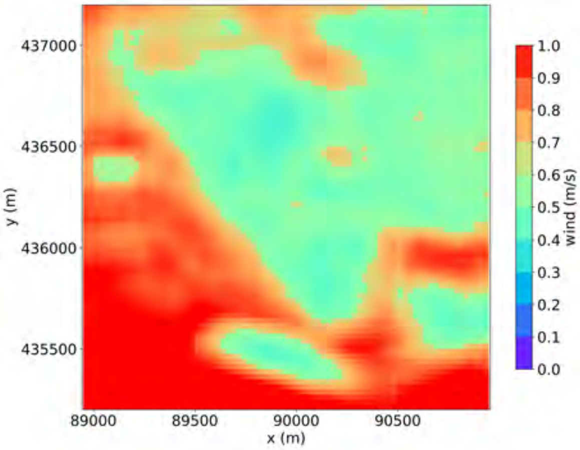
Possible wind directions

The wind field direction possibilities of Rotterdam Bospolder Tussendijken case study, see Figure 7.3.

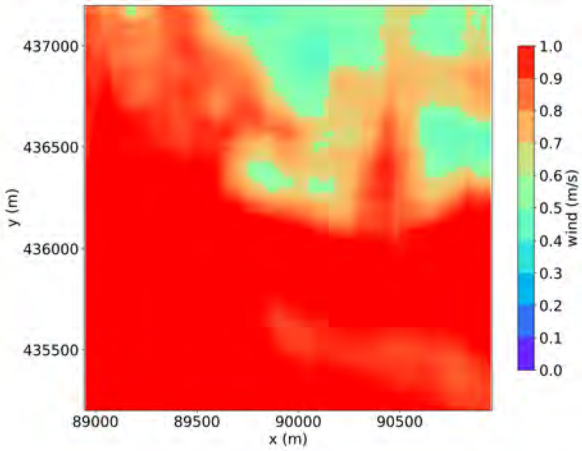
(a) rgb image



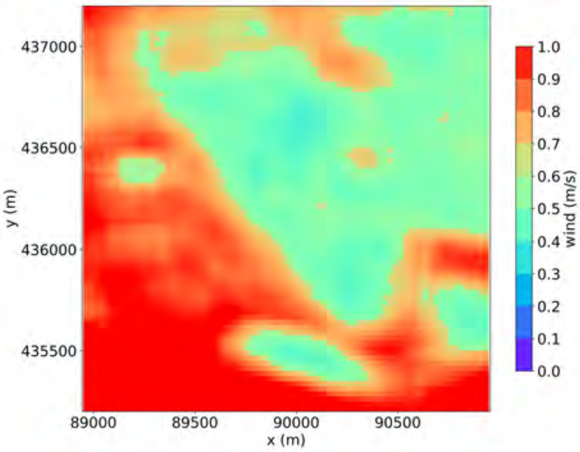
(b) north



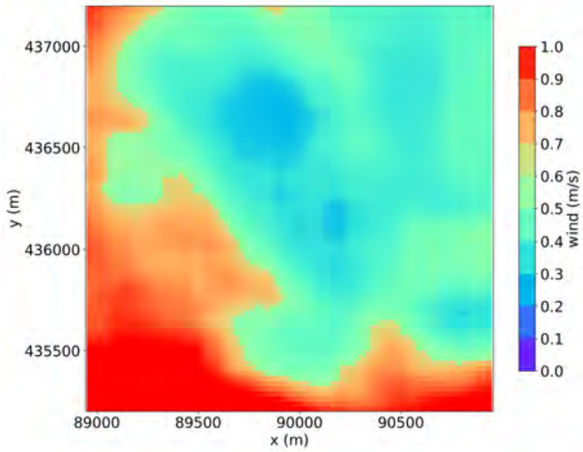
(c) east



(d) south



(e) west

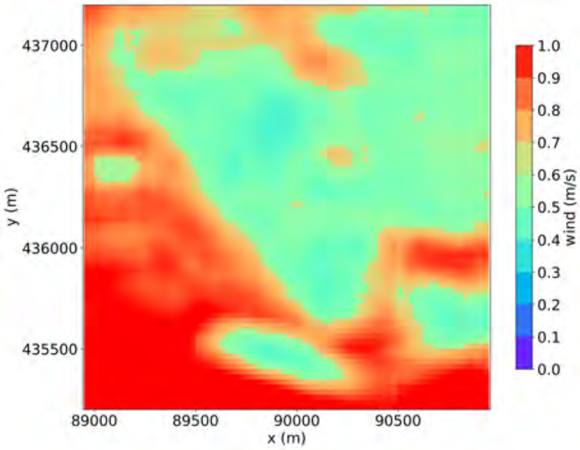


(f) nowind

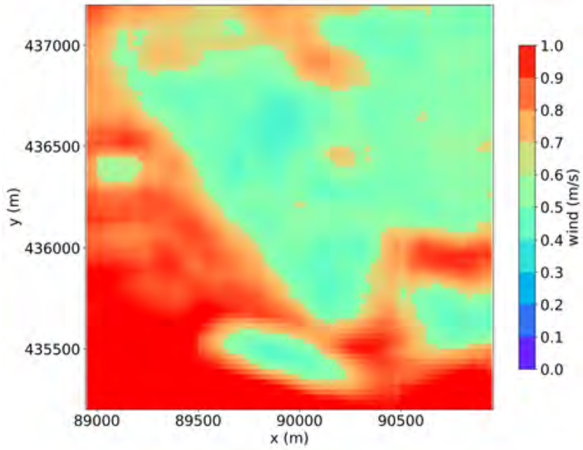
Figure 7.3: Different wind directions files on research area Rotterdam Bospolder Tussendijken

1st of Juli 2015

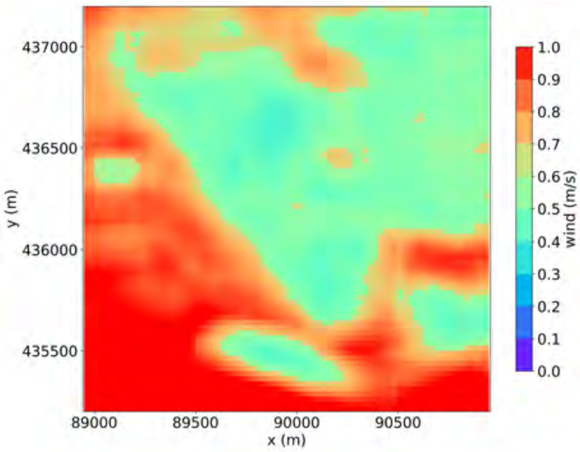
First the wind calculation is executed. On the 1st of Juli there is only wind coming from the east, see Figure 7.4.



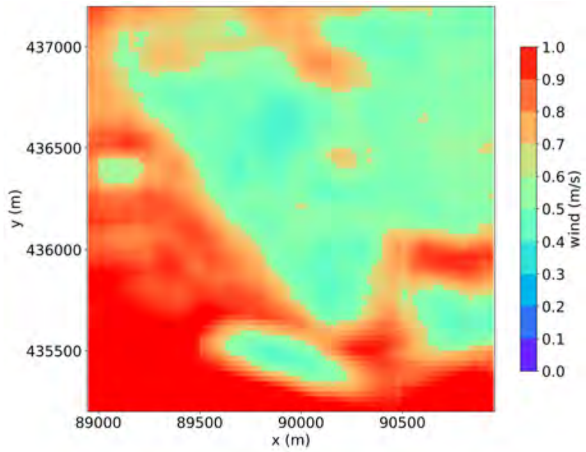
(a) 9:00



(b) 12:00



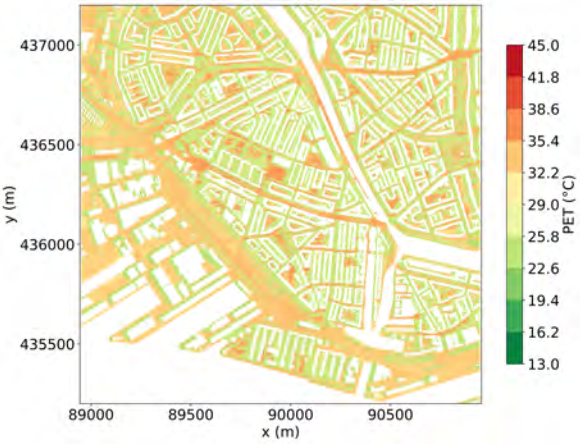
(c) 15:00



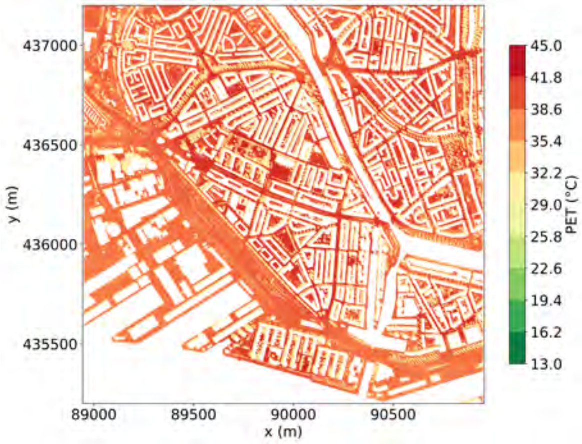
(d) vegetation fraction

Figure 7.4: 18:00

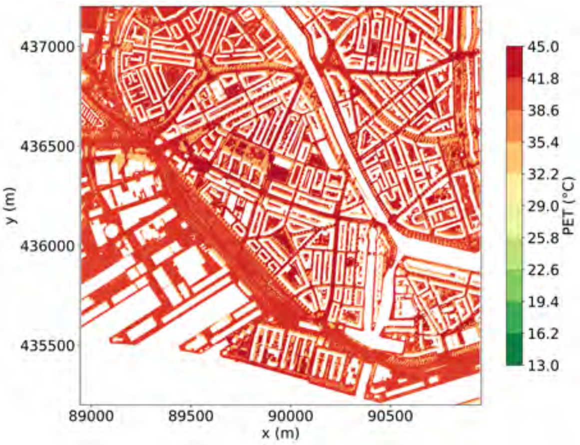
The PET is determined with these influences, , see Figure 7.5.



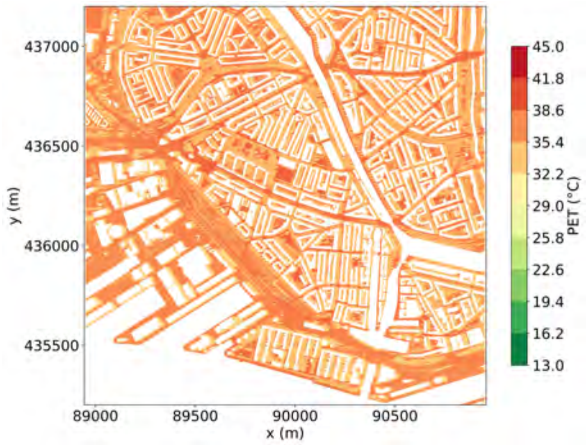
(a) 9:00



(b) 12:00



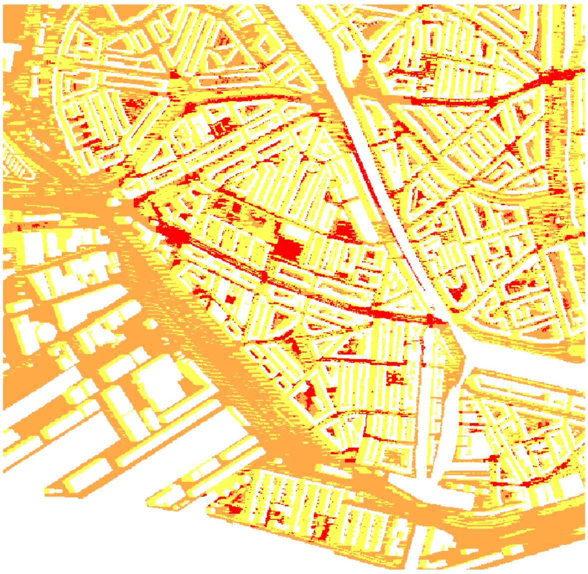
(c) 15:00



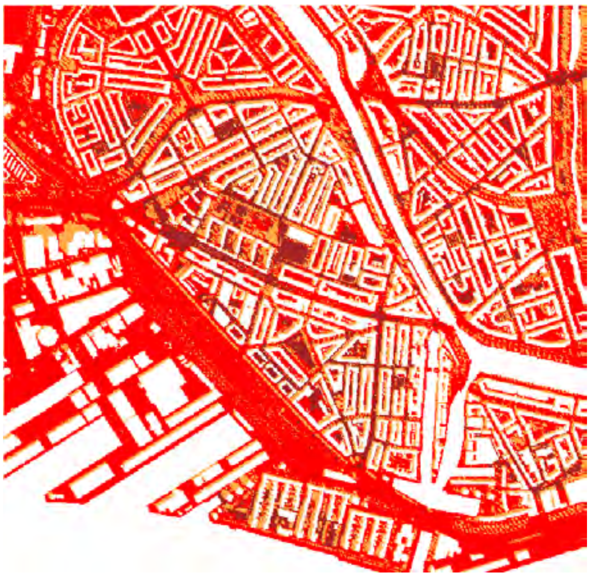
(d) vegetation fraction

Figure 7.5: 18:00

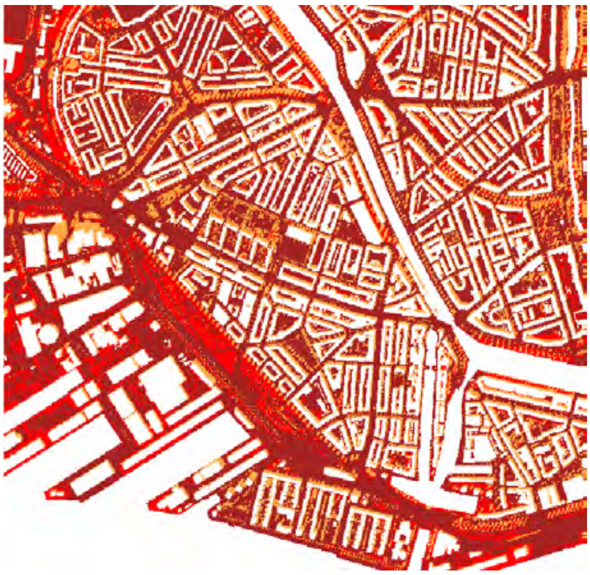
In the color styling of the PET classes, see Figure 7.6. , see Figure 7.3 showcases the legend.



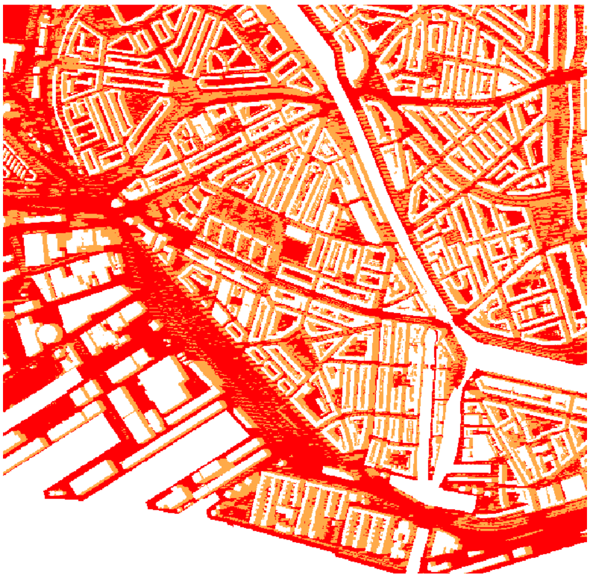
(a) 9:00



(b) 12:00



(c) 15:00



(d) vegetation fraction

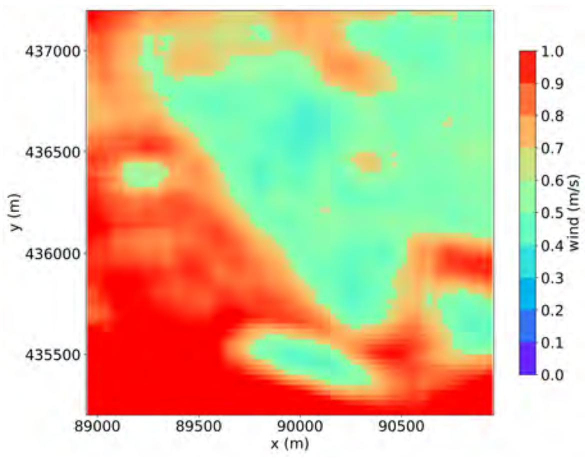
Figure 7.6: Color classes of PET on the 1st of July 2015

PET	Thermal perception	Grade of physiological stress	color code
13 - 18 °C	Slightly cool	Slight cold stress	
18 - 23 °C	Comfortable	No thermal stress	
23 - 29 °C	Slightly warm	Slight heat stress	
29 - 35 °C	Warm	Moderate heat stress	
35 - 41 °C	Hot	Strong heat stress	
>41 °C	Very hot	Extreme heat stress	

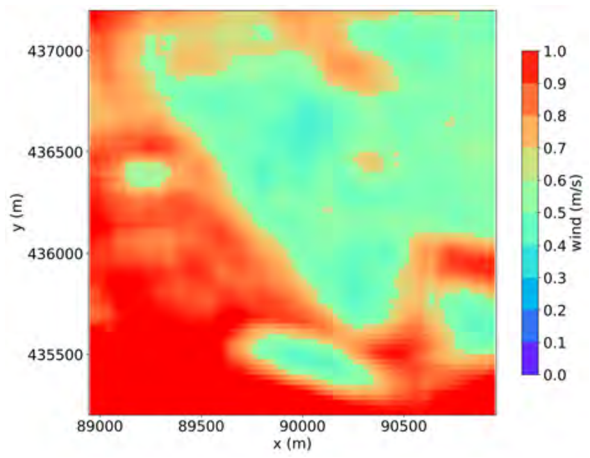
Table 7.3: Temperature and corresponding thermal perception

29th of June 2015

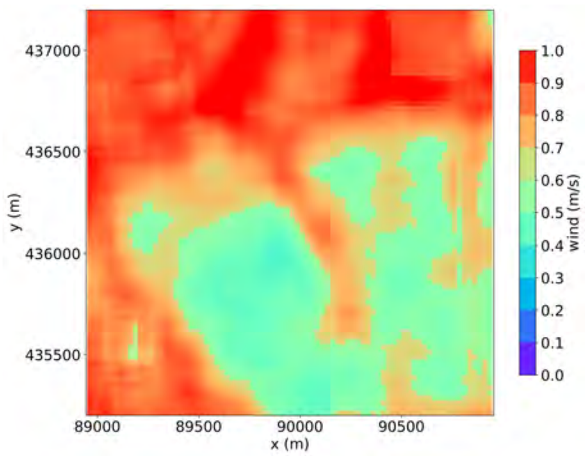
First the wind calculation is executed. On the 29th of June 2015 there is only wind coming from the west and north, see Figure 7.7.



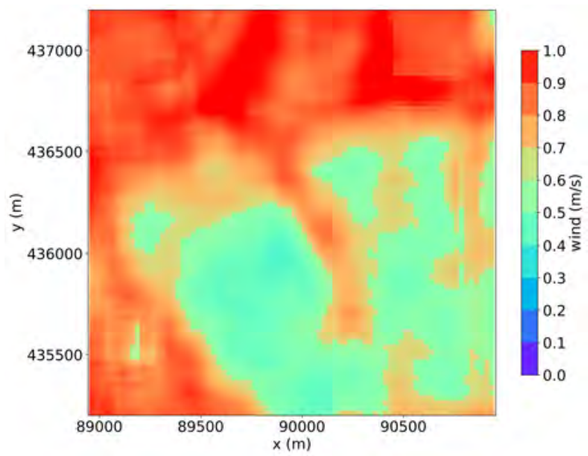
(a) 9:00



(b) 12:00



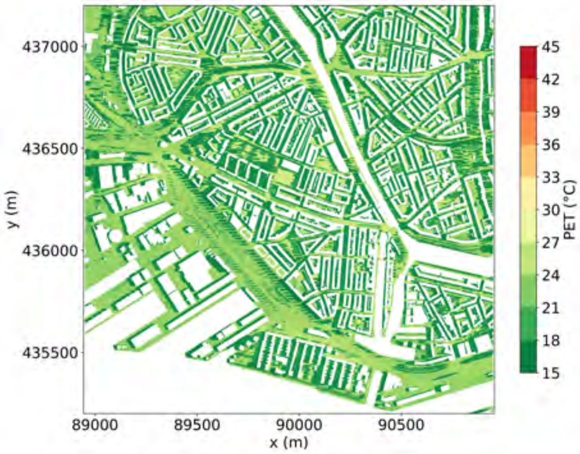
(c) 15:00



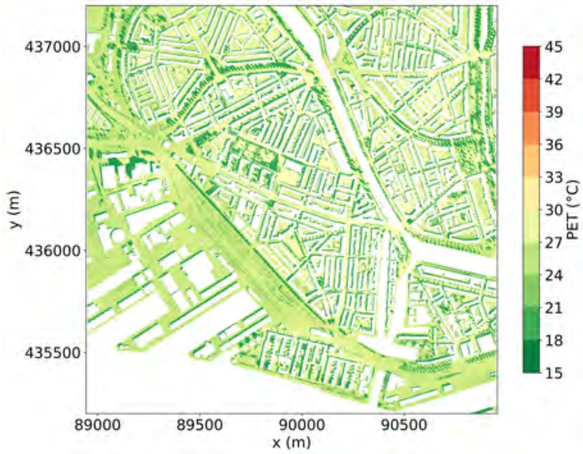
(d) vegetation fraction

Figure 7.7: 18:00

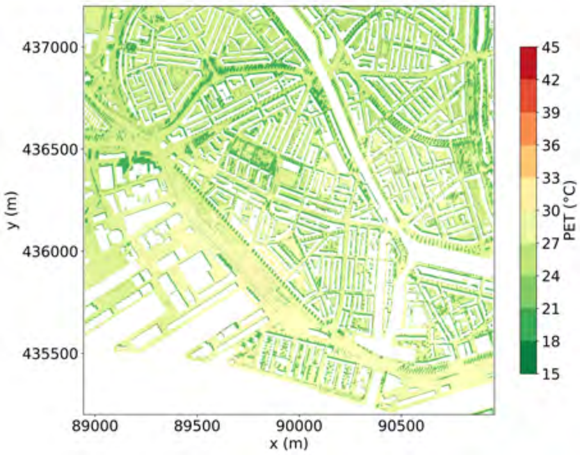
The PET is determined with these influences, see Figure 7.8.



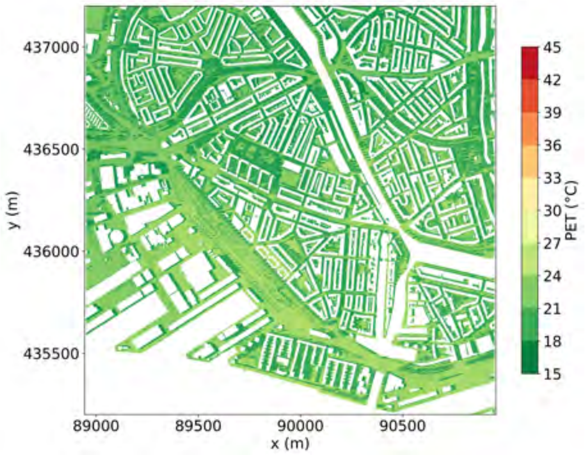
(a) 9:00



(b) 12:00



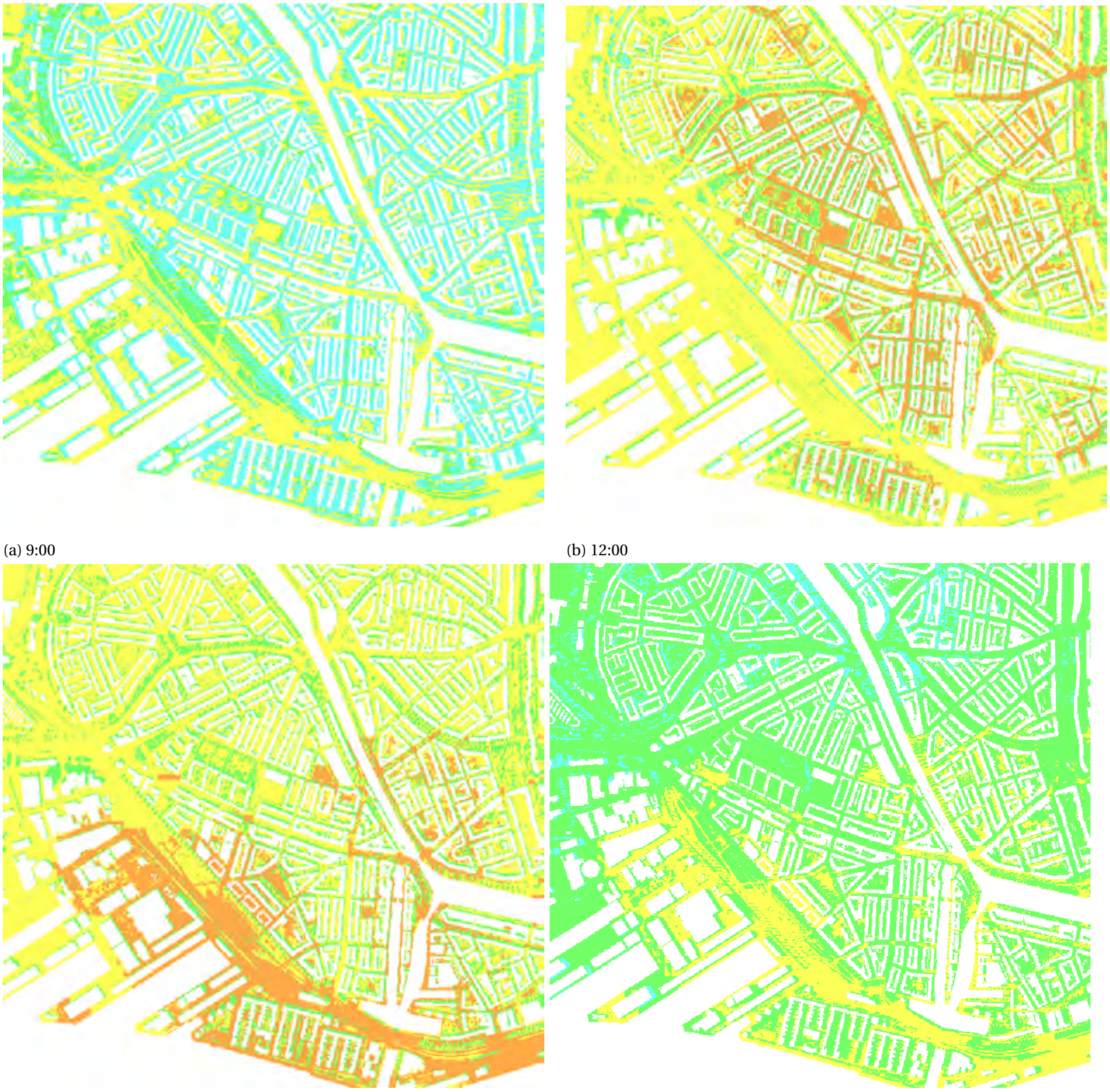
(c) 15:00



(d) vegetation fraction

Figure 7.8: 18:00

In the color styling of the PET classes, see Figure 7.9. For the legend see Figure 7.4



(a) 9:00 (b) 12:00 (c) 15:00 (d) vegetation fraction

Figure 7.9: Color classes of PET on the 29th of June 2015

PET	Thermal perception	Grade of physiological stress	color code
13 - 18 °C	Slightly cool	Slight cold stress	
18 - 23 °C	Comfortable	No thermal stress	
23 - 29 °C	Slightly warm	Slight heat stress	
29 - 35 °C	Warm	Moderate heat stress	
35 - 41 °C	Hot	Strong heat stress	
>41 °C	Very hot	Extreme heat stress	

Table 7.4: Temperature and corresponding thermal perception

7.2. Applications

Determining thermal accessibility

To see the performance of the accessibility on the PET heat grid, to calculate walking accessibility, the decision was made to use the generated maps of PET per hour as input. While vector integration could have been an option, the variation in PET on a small scale means that an average value for a path segment wouldn't accurately represent the whole picture. Additionally, there is no pedestrian network of line segments available; therefore, the raster represents the area to traverse. Another decision could be to add the values on a vector pedestrian network. Due to the limited time and lack of finding a good program to add the values of PET on a pedestrian network, the raster data was used. Next to that, raster data is also more storage-efficient for large continuous datasets since it only stores data values at each grid cell, unlike vector data which requires explicit storage of individual vector features and can be more memory-intensive. Raster data also enables the creation of visually appealing maps, especially when rendering continuous data for the accessibility of places in an isochrone manner.

Eventually the tool r.walk is used from the Grass package in QGIS. Input that is given is the DEM on which people can walk upon. The cost layer is the PET map, but is first translated to a normalized friction cost map.

Listing 7.1: Normalization in QGIS Raster Calculator

```
    ("pet_hour" - 21) / (45 - 21)
```

The preferred accumulation cost will be the temperature experience of 21 PET °C within an estimated walking distance of 500 m for elderly people and 200 m for young children. The maximum friction cost will depend on the target groups. It's possible to adjust this as needed. Starting points are essential, and parks are provided as an example. However, accessibility can change throughout the day. The r.walk function calculates the cumulative cost of moving between different geographical locations on an elevation raster map. The output includes two raster maps: one showing the lowest cumulative cost (time) of moving from each cell to user-specified starting points, and another illustrating the direction of movement to the subsequent cell along the path back to the starting point as movement direction. In comparison to r.cost, this function takes into account not only the friction map but also anisotropic travel time. This considers variations in walking speed associated with both downhill and uphill movements. Figure 7.10, 7.11, and 7.12 showcase the service area of the Dakpark and park 1943, the cumsum from the Visserijplein market square, and the cumsum from several playgrounds in the neighborhood on warm days of the 29th of June and the 1st of July in 2015. It's worth noting that the thermal accessibility service area of the market square and the parks are not covered all the time in the whole neighborhood, and on the 1st of July, they both shrink in area. In contrast, the playgrounds, which are frequently represented in the neighborhood, are covered the most at all times. This could be a potential strategy to invest in the nodes along the network before transforming the street network.

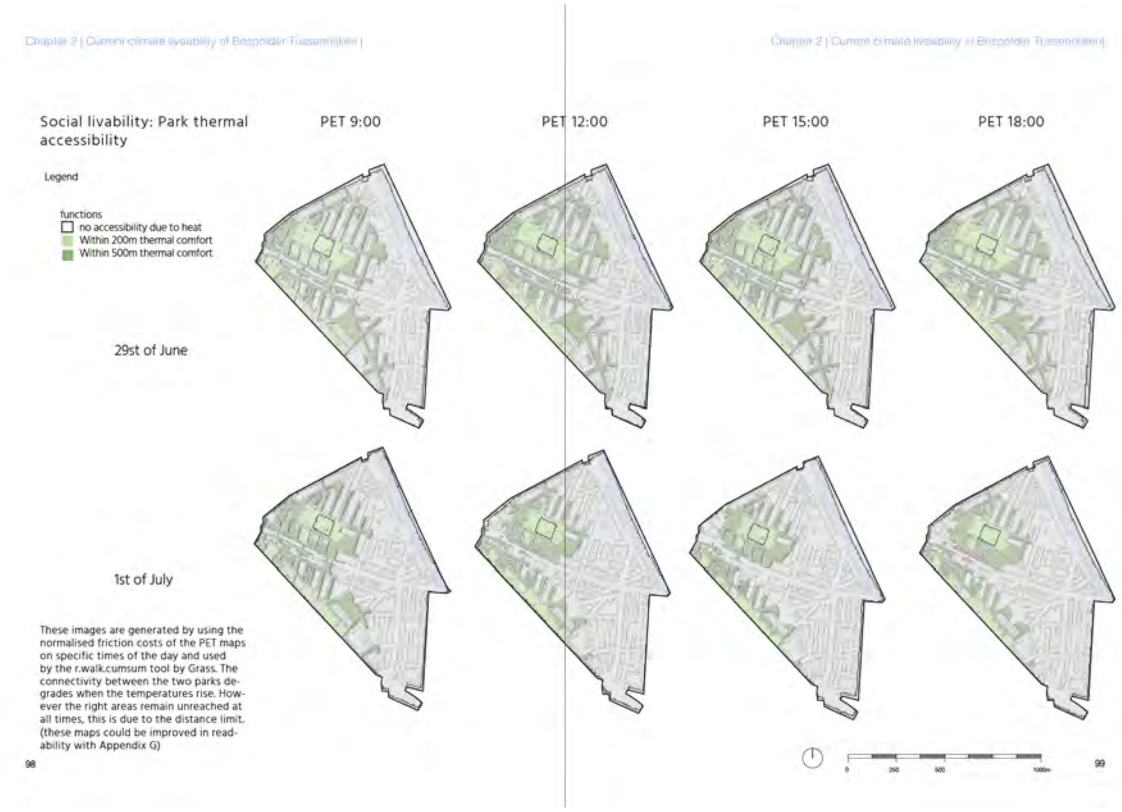


Figure 7.10: Cumulative cost of walking with thermal comfort to parks with 500m and 200m thermal comfort accessibility

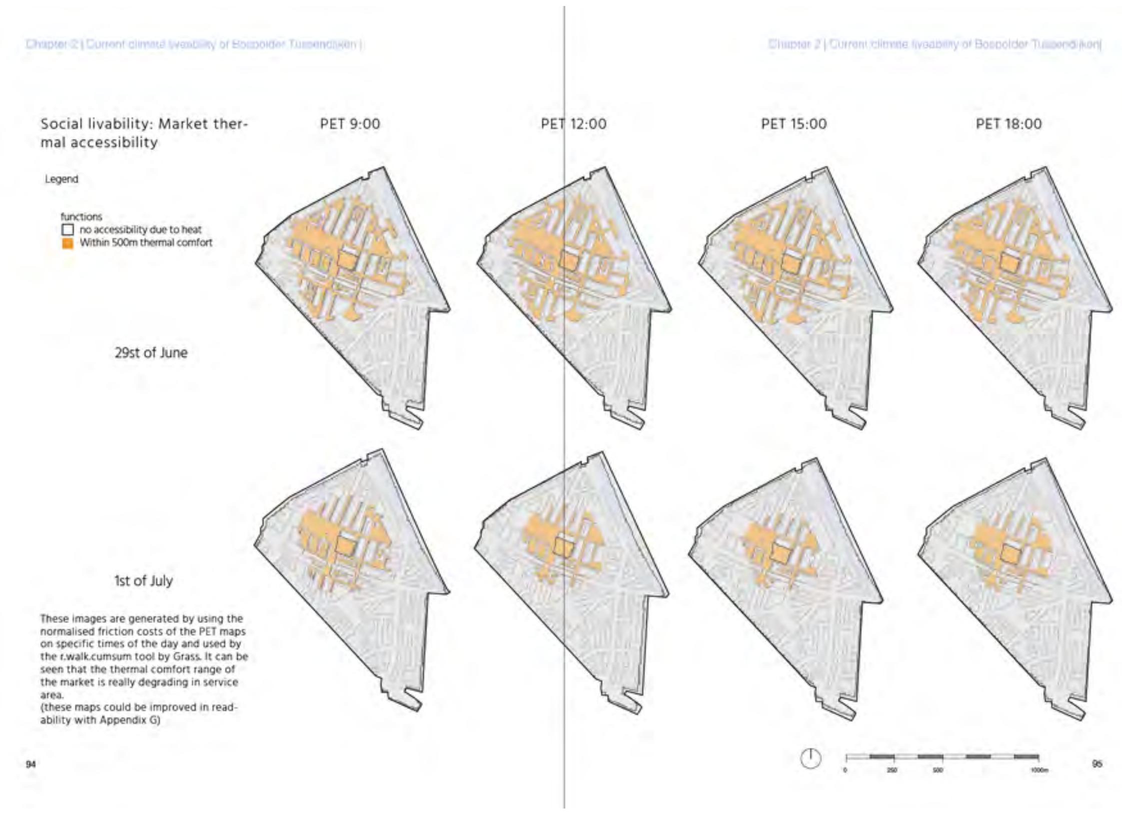


Figure 7.11: Cumulative cost of walking with thermal comfort to market with 500m and 200m thermal comfort accessibility

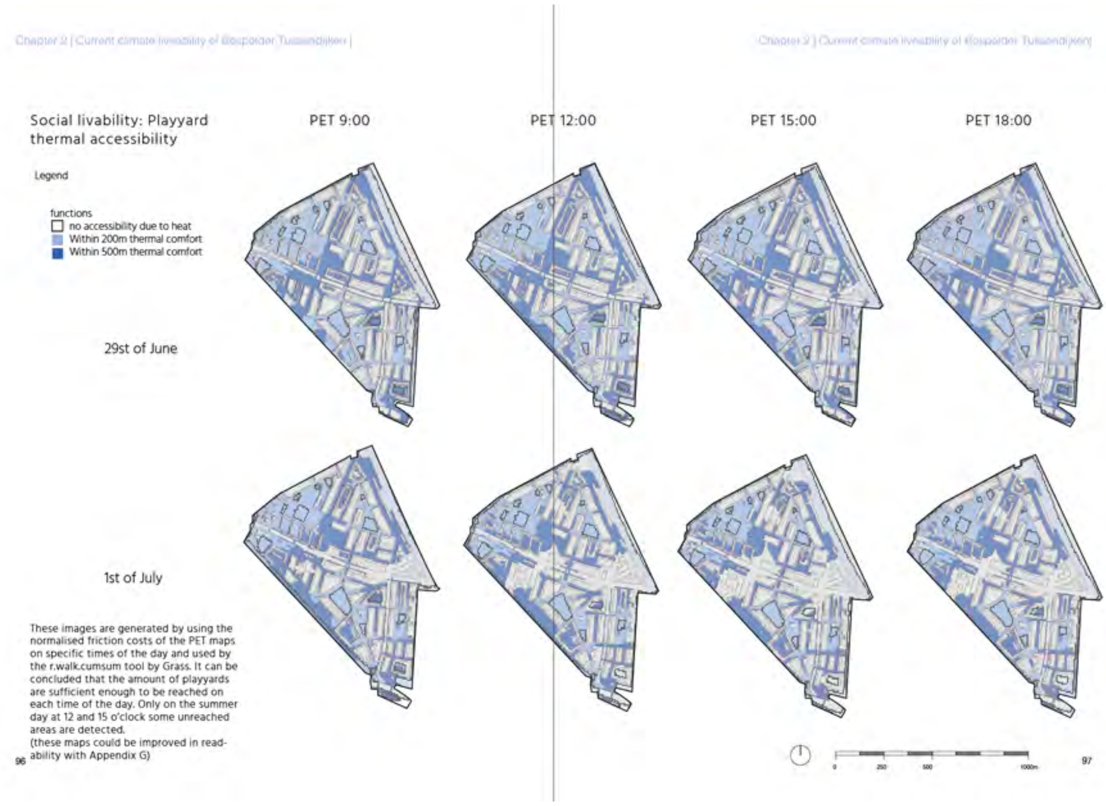


Figure 7.12: Cumulative cost of walking with thermal comfort to playgrounds with 500m and 200m thermal comfort accessibility

7.3. Testing the design interventions

For the testing of the design interventions, the current situation needs to be modified to the new situation which need to be tested, see figures 7.13, 7.14. The procedure is written down below.

Figure 7.13: Adding greenery and replacing parking spaces

Figure 7.14: Trees added / updated by size

For enhancing the model with vegetation, the NDVI and RGB input files need to be modified:

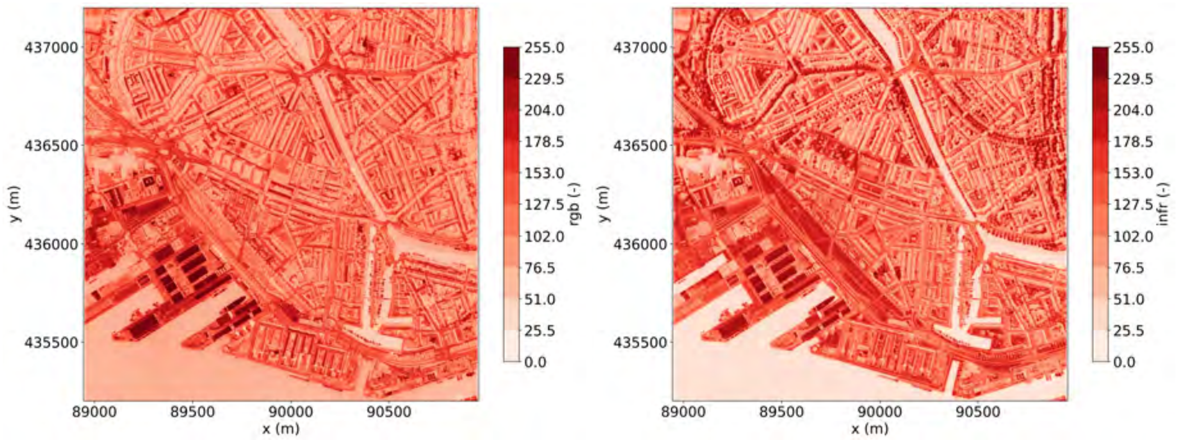
```
Enhancing the simulation model with vegetation:

1 Utilize a shapefile to depict greenery on a separate layer. This can be
    achieved by either referencing an RGB image or the bgt wegdeel layer to
     match the existing landscape.
2 Navigate to the menu and select Raster > Conversion > Rasterize (Vector
    to Raster).
Within the Rasterize calculator dialog:
a. Choose the polygon layer intended for rasterization as the input vector
    layer and set it to value 1.
b. Define the output raster size, extent, and resolution.
c. During rasterization, areas lacking values will be assigned a nodata
    value. Thus, it's essential to employ the raster tool 'Fill NoData
    cells' and assign a value of 0.
3 Proceed by setting the red value of the new layer from 1 to 40 ("Output
    raster@1"= 1) * 40.
a. Subsequently, generate two additional layers filled with 0. Merge the 40
     band with the other two layers of 0 using raster > miscellaneous >
```

```
    merge, opting for the "Place each input file into a separate band"
    option.
  b. Go to Layer > Create Layer > New Raster Layer from the menu. Specify
    dimensions, extent, and resolution for the new raster layer, ensuring
    alignment with the existing ndvi_infr image. Designate three bands for
    the new raster layer.
  c. Utilize the raster calculator tool (accessible from the Processing
    Toolbox) and input the expression 40 * (band@1 > -1), where band@1
    represents the first band of the new raster layer. This expression will
     assign a value of 40 to all pixels in the first band. Repeat this
    process for the remaining two bands if specific values are required.
  4 Combine the new raster layer with existing layers, such as ndvi_infr@1
    and ndvi_infrz@1, creating a new band named "infrnew_add."

  To merge with the existing RGB image and rgb_infr, follow steps 3 to 5 as
    outlined above.
```

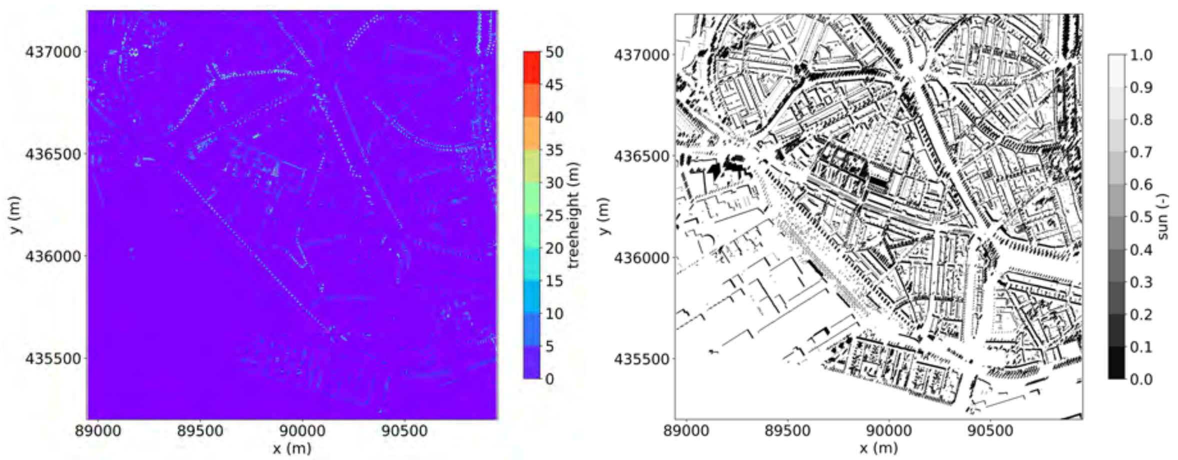
These are the adapted rgb and infr input files cropped to the research area, Figure 7.15.



(a) rgb changed

(b) infr changed

Figure 7.15: rgb and infr changed in values on specific streets and visserijplein

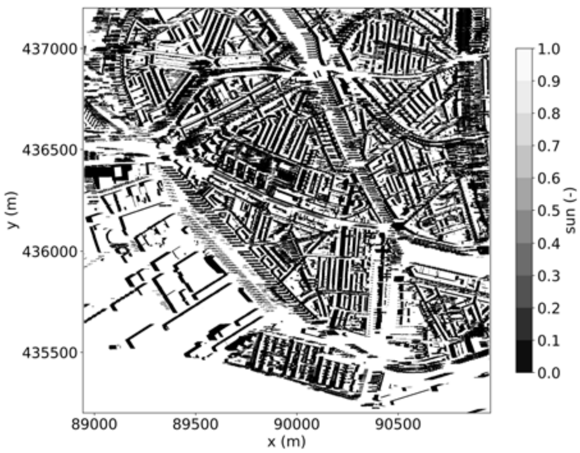


(a) tree height adapted

(b) sun areas at 15:00 from tree height

Figure 7.16: Shadow influence at 15:00

The generated shadow pattern after updating the treemask and tree height with the new trees of 2m radial see figure 7.17. For artificial constructutions the modifications were made directly in the shadow files. This is for the Visserijplein and Schiedamseweg the case.



(a) 9:00

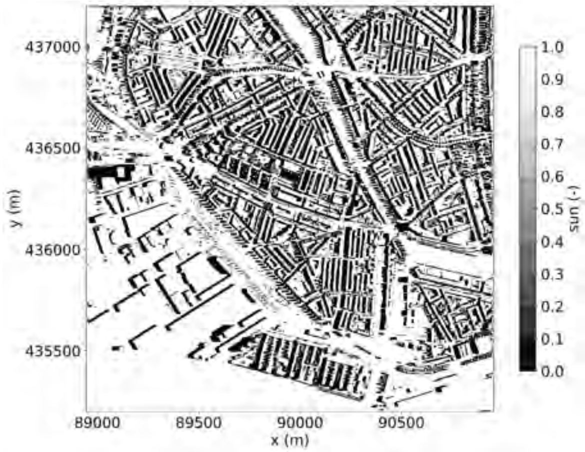
(b) 12:00

(c) 15:00

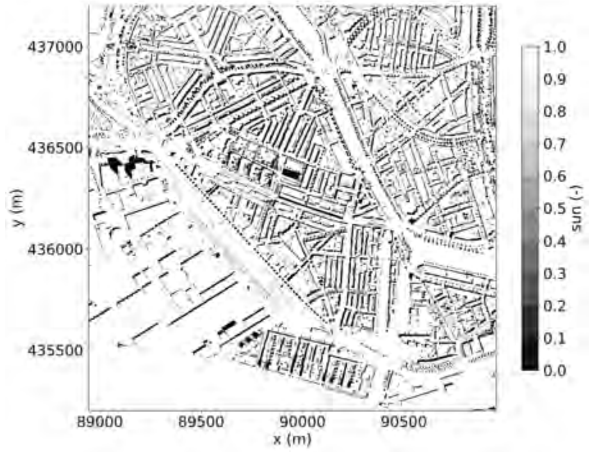
(d) 18:00

Figure 7.17: Sun pattern over the day with design interventions of adaptation of trees

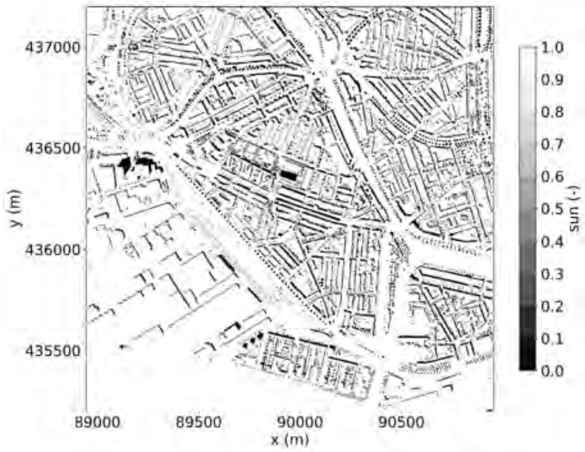
.



(a) 9:00



(b) 12:00

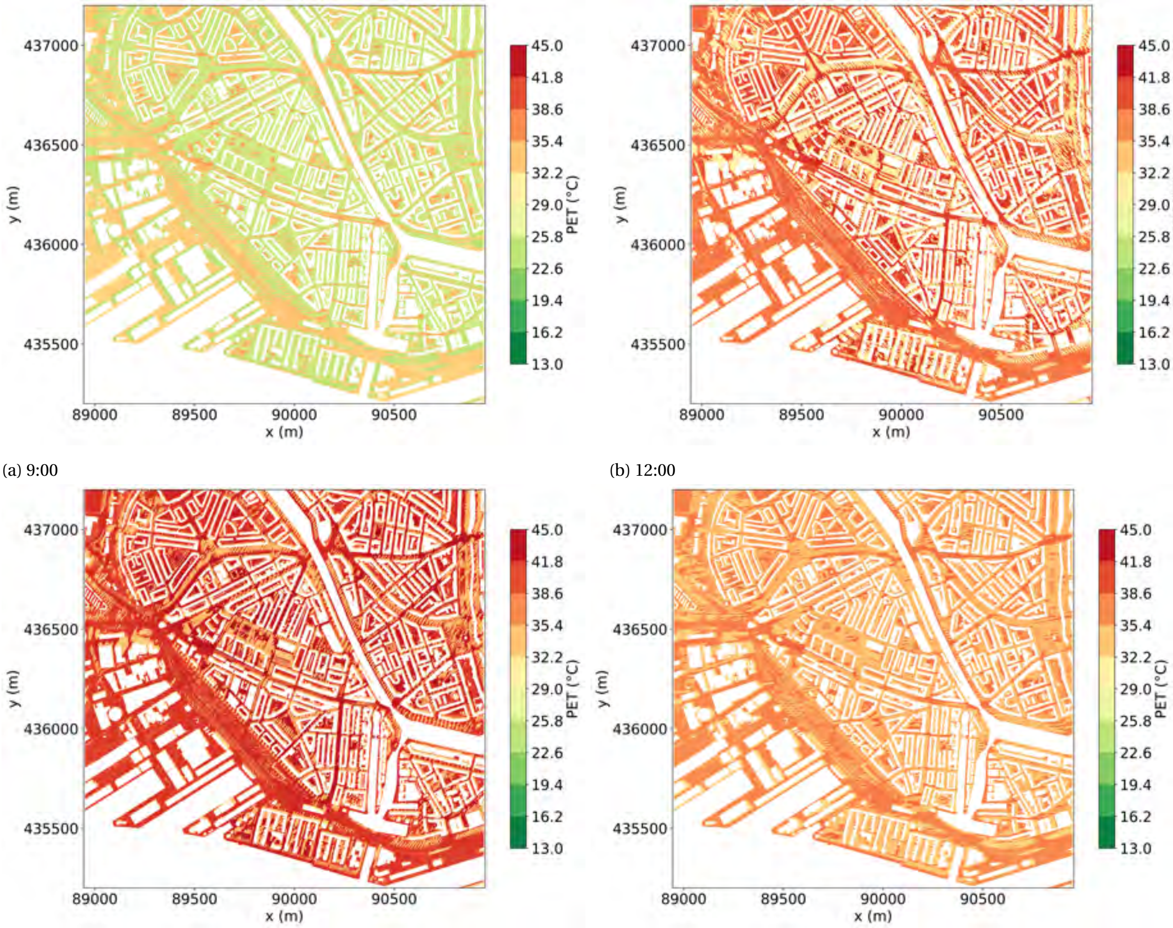


(c) 15:00

(d) 18:00

Figure 7.18: Sun pattern over the day with design interventions of adaptation of trees

The generated PETs are in Figure 7.18.

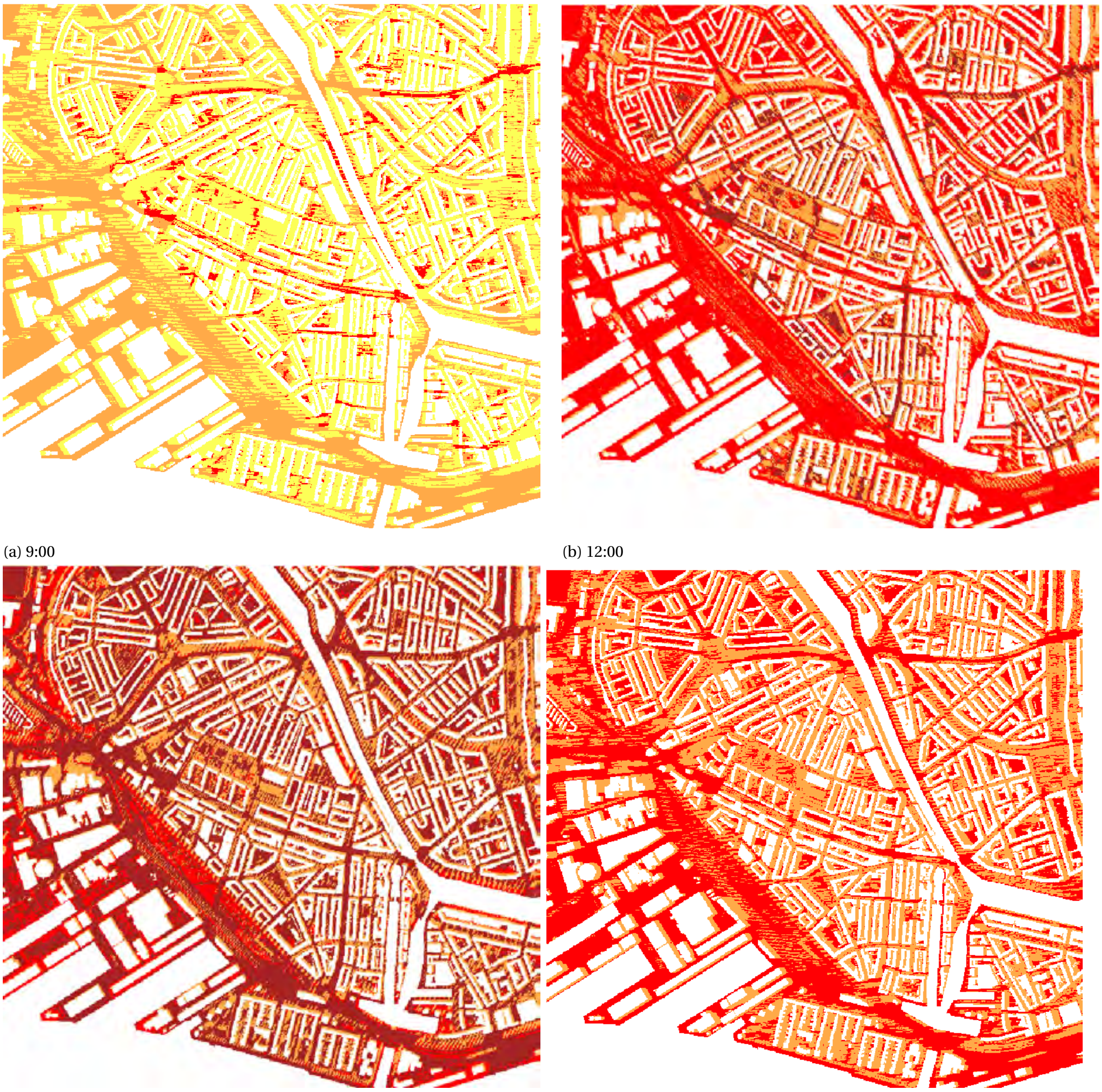


(a) 9:00 (b) 12:00

(c) 15:00 (d) vegetation fraction

Figure 7.19: 18:00

In the color styling of the PET classes, see Figure 7.19.

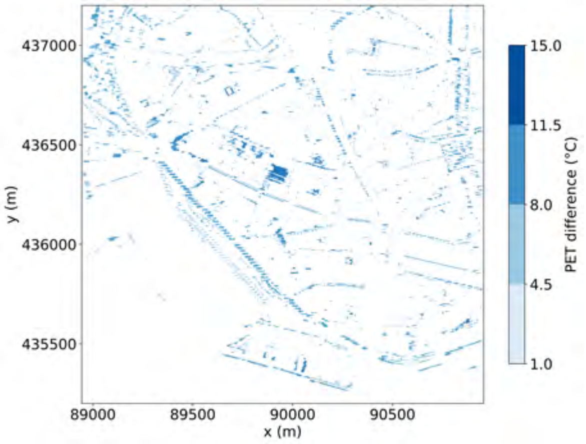


(a) 9:00 (b) 12:00

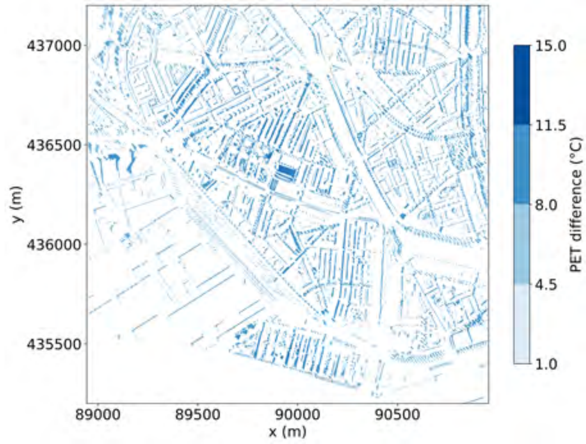
(c) 15:00 (d) vegetation fraction

Figure 7.20: Color classes of PET on the design interventions on the 1st of July 2015

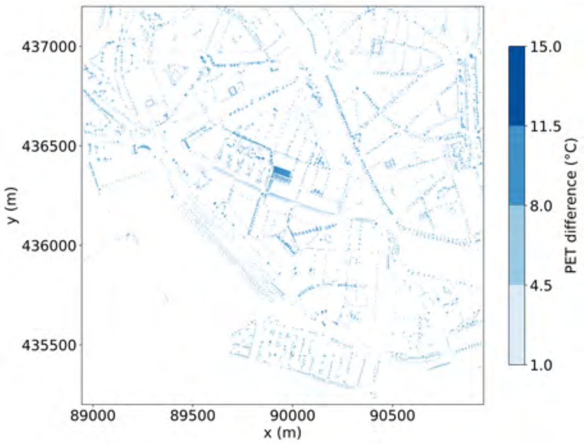
The difference in PETS in comparison before the interventions are marked in figure Figure 7.20.



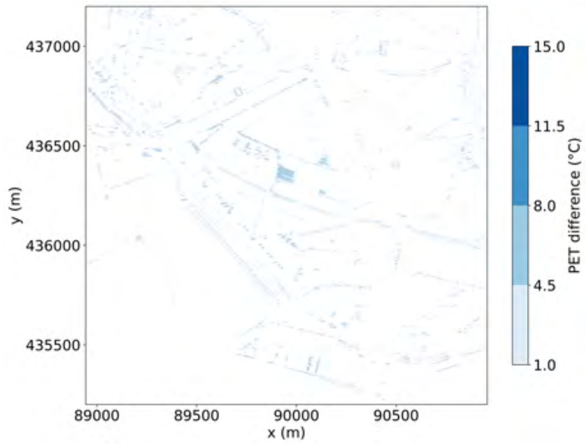
(a) 9:00



(b) 12:00



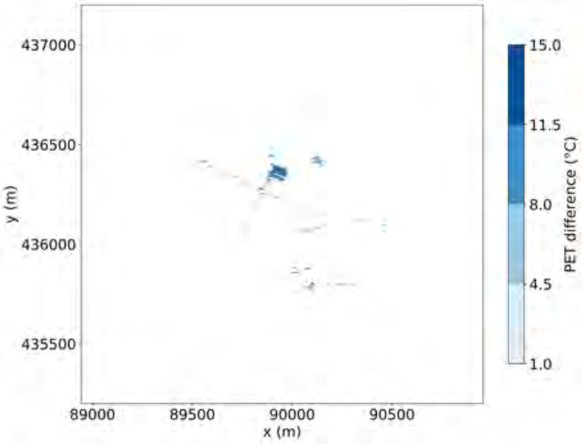
(c) 15:00



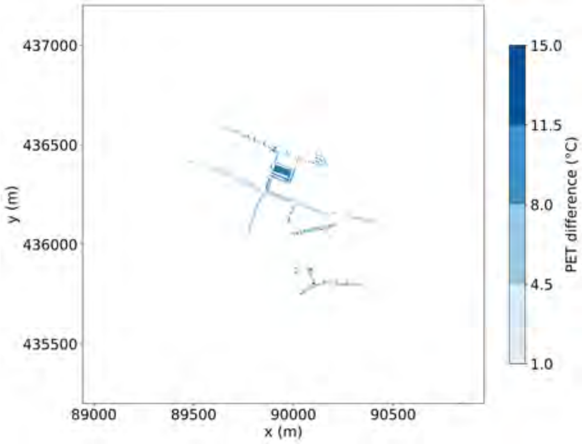
(d) vegetation fraction

Figure 7.21: 18:00

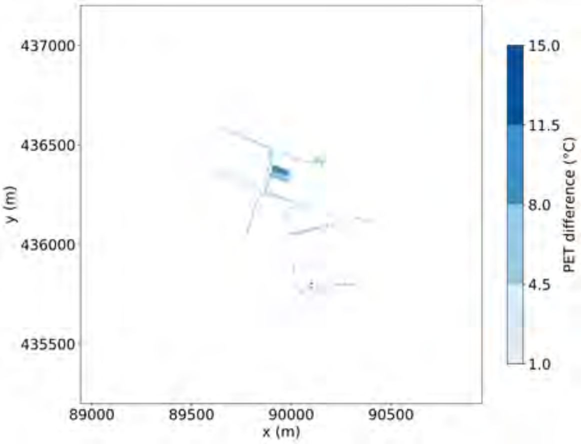
The difference in PETS in comparison before the interventions are marked in figure 7.21 on the places of intervention.



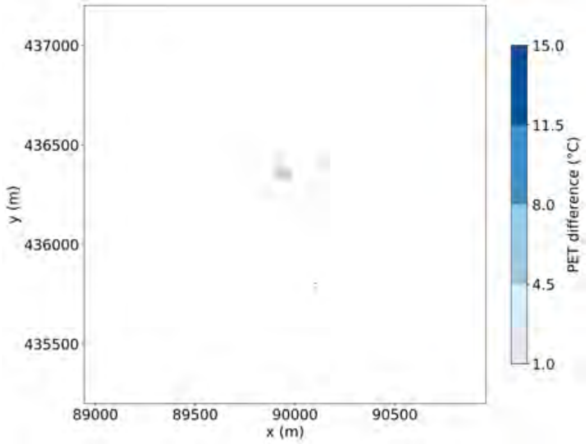
(a) 9:00



(b) 12:00



(c) 15:00



(d) vegetation fraction

Figure 7.22: 18:00

For a better closeup of the public spaces where the interventions took place are Figure 7.22. There is a mitigating effect on the 1st of July.

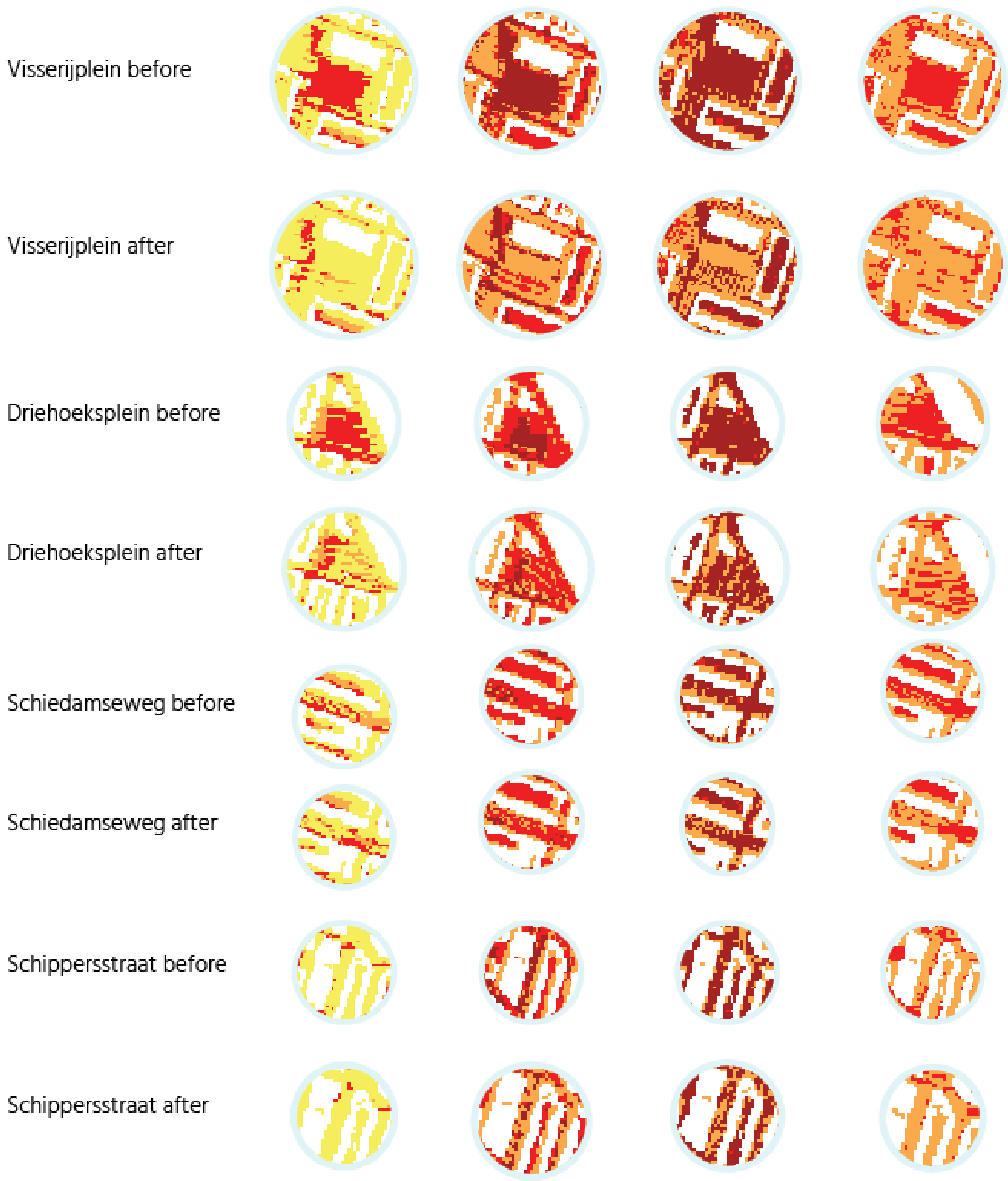


Figure 7.23: Comparison of public spaces after heat mitigation measures

Design interventions tested with physiological equivalent models offer a powerful means to simulate and understand how changes to the built environment impact human health. These models allow researchers and designers to analyze complex interactions between environmental factors and physiological responses in a controlled setting. This method poses speculative design scenario's to be tested which will serve the human comfort and health. At the moment it is possible to adjust the greenery and the tree and building height for the simulations. The input pre-processing phase can be smoother. Collaboration among architects, planners, engineers, and health experts is facilitated by these interdisciplinary tools, leading to optimized designs that benefit diverse populations. Ultimately, integrating physiological equivalent models into design processes enhances the overall quality and sustainability of built environments, fostering healthier communities.

8

PETs evaluation

8.1. Reproducability

Input data

The input data is focused on the datasets required to run the method in order to conduct the results. The input data is categorised in whenever they are in non-proprietary format, if third party reuse is possible, if the guidelines are referenced to the data. The datasets provided are in non-proprietary formats and include Geo tiff, text, and vector datasets in Geopackage format. The spatial data consists of raster GeoTiff and vector datasets, while the climate data is in text format. The text file is derived from [KNMI, 0000] and contains hourly data. It includes atmospheric temperature (TT), wind speed (FF), wind direction (DD), global solar radiation (Q), relative humidity (RH), and minimum and maximum temperatures (Tmin and Tmax) between 8:00 UTC and 9:00 UTC of the following hour. It also includes the average daily wind speed (U). The file has been modified with pysolar.py to calculate Qdif, generate Sunalt, activate the Day/Night switch, and display the diurnal factor on an hourly basis, making it not immediately repeatable for other users. In addition to the paper of Koopmans there is an improvement on third party reuse, since two of the input data are now open access resources. The vector data, including building envelopes, trees, and water, are derived from [Geofabrik, 2020] and trees from [**?**], saved as Geo-packages, and eventually rasterized as Tiffs in QGIS. The workaround for Bomenregister is needed to make it more reproducable. For Rotterdam the data of trees can be retrieved from `https://diensten.rotterdam.nl/arcgis/rest/services/SB_Infra` the tree point coordinates can be retrieved with additional attribute information. Relevant attribute information are height and crown size. With preparation actions in QGIS the points can be buffered and rasterized according to half the crown size and the height of the tree_mask can be assigned to the specific rasters. In order to retrieve the Skyview factor data an API code must be made available. This code for transferring information of the webservice towards a raster data on their own device requires a script to be written to retrieve this information. The code get_svf.py retrieves the input values of svf maps needed for the calculation of the svf calculation. The code to transform the text file to the attributed required parameters is done through pysolar.py and get_svf.py for retrieving the Sky view factor tiles and the trees with crownsize by [diensten Rotterdam, 2023].

Methods

The method section is subdivided into pre-processing, method, analysis and processing, and computational environment. The software is open and available via GitHub or a plugin of QGIS. This was due to the lack of amount of money to create reproducible software for third-party use. The PET simulator is available through © 2024 by Marieke van Esch is licensed under CC BY-SA 4.0 (created with `https://chooser-beta.creativecommons.org/`) via `https://github.com/mariekeve/pet_simulator.git` see Figure 8.1, therefore this reproducible software is for third-party use.

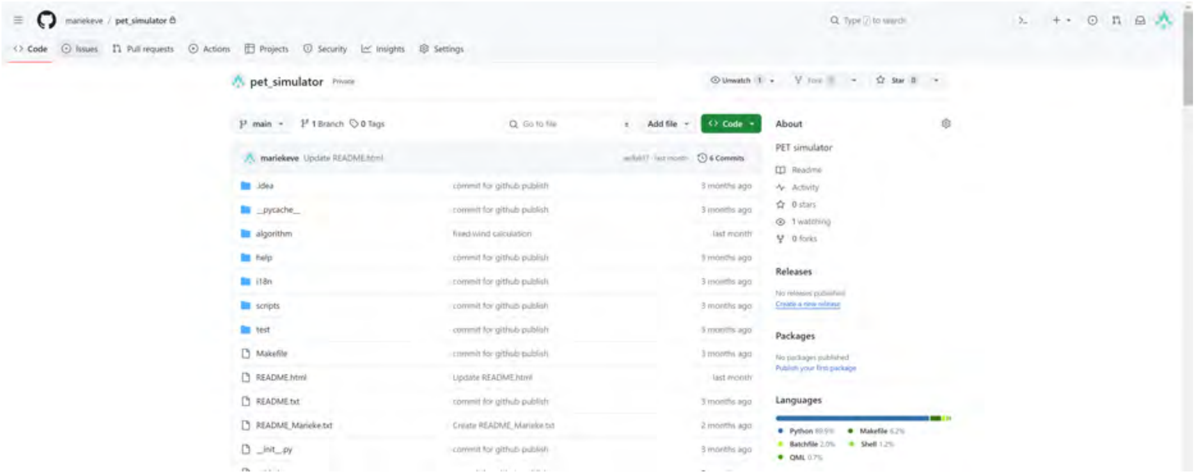


Figure 8.1: Fig. Github page for retrieving the PET simulator plugin repository

The pre-processing reproduction steps are documented in the User manual Chapter C. As well as the Wageningen test area and the difference Wageningen test area. All parameters are provided: The parameters are obtained by the spatial and dynamic parameters section. The dynamic section entails the converted KNMI hourly values. And the static parameters are obtained by giving the wished spatial frame for your output, those are summurised in pet_parameters.py. For the method, the approach of calculating the PET is intended to calculate the wind by the MacDonald method validated for the Dutch context (to be more specific in the Wageningen Herwijnen context). `https://github.com/mariekeve/pet_simulator.git` contains a README.txt file explaining all the python files separately and their intermediate results. For the analysis part, the same in-between output should be generated and reproduced through other parties. The processing involves using Python software for computational steps, along with importing libraries such as bindings PyQt for Qt designer, geospatial libraries like from osgeo import gdal, osr, ogr. Next up calculation libraries like numpy, pandas, multiprocessing, datetime, time, matplotlib, PIL, csv and pvlib. There are extension of Python files: ndvi_calculator, pet_parameter, geotiffcreator, svf_footprint, vegfra_footprint, fraction_area_buildings_treeregr, PET_simulator, urban_heat, get_svf.py, pysolarv1 and PET_calculate. These Python files are interconnected, leading to jointed results. The ndvi_calculator is used to calculate areas that qualify as evaporative surfaces and contain a Bowen ratio. svf_footprint and vegfra_footprint depend on wind direction to average the values on a 25m resolution. fraction_area_buildings_treeregr is for calculating wind. PET_calculate combines output files of intermediate steps and climate dynamic data to calculate PET in sunny and shady locations. Computational environments are documented and provided: the computational environment is Python and are documented. Next to this QGIS is used as a visualisation and communication environment to use for different third parties. This is possible through the QGIS plugin throughQT designer QGIS plugin developer.

The visualization environment is QGIS. This is an graphical environment used by urban designers. Through the integration of PyQt the intermediate results are immediately put in the QGIS project. Therefore the transparancy of the intermediate output files is upgraded for third party users with expertise and not. Versions of relevant software components (libraries, packages are provided). The version of GDAL 3.7.1 needs to be installed on both QGIS 3.30 and your Python 3.9 environment. Also the newest version of UMEP 4.0.4 needs to be installed on your device. The run was ran with The HP Zbook with Intel Core i7 delivers high performance with its powerful CPU, boasting a 2.2 GHz base frequency and up to 4.1 GHz maximum Turbo Boost frequency across its 6 cores. It has an installed RAM of 16.0 GB which is a substantial amount of running separate tasks. The permanent storage capacity of the PC is 475GB.

Results

The results of the code have been verified for the Wageningen area, and the names of the services for download are provided. The software has been assessed through interaction with the publishers. A camera-ready paper will be published after the submission of the thesis. This thesis is reviewed by two other mentors and is published whilst it was finished. The software is available through the GIT `https://github.com/mariekeve/pet_simulator.git`.

Input data		2
Methods	pre-processing	1
	method, analysis, processing	2
	computational environment	3
	visualisation	2
Results		3

8.2. Assessment reproducibility.

Table **??** deals with the reproducibility of the refactored code and the integration of a QGIS plugin. The main points of improvement are to improve the accessibility of the input datasets. Unfortunately, modifying input files to test alternative designs is still an intensive task for third-party use, but not impossible. A description is given in chapter 7. For the methods, the calculation workflow is more integrated with pet_simulator, the parameters are in pet_parameters and the geospatial transformations are done in geotiff_creator. The result and intermediate results of all calculations are provided by ndvi_calculator, svf_footprint, vegfra_footprint, fraction_area_buildings_treeregr, urban_heat, get_svf.py, pysolarv1 and PET_calculate. The computing environment is minimized to Python and QGIS as the visual environment. The advantage of the plugin is that the intermediate results are also made available in the QGIS project to do applications like testing the design and integrating other techniques like testing design interventions after modifying the original input files, street orientation, attraction betweeness to determine the most walked streets for multiple destinations. The plugin is publicly available via a GIT publication for use by third parties.

9

Discussions and limitations

9.1. Discussion

Validation

Due to the reproducability requirements and the refactoring of the computation model, a decision was made to adopt a fixed block size of 25 meters for the computation, in contrast to the variable block sizes of 25 and 35 meters utilized in the computation model proposed by [Koopmans et al., 2020]. Consequently, the fraction density factor of buildings needed adjustment to accommodate this newly specified block size. Subsequently, validation of the data was conducted.

Verification

The validation method resulted in the standardization of the fraction density factor. This model, adapted from Wageningen, was also applied to the context of Rotterdam. However, to ensure its suitability for this specific scenario, a verification method could have been employed, such as field measurements, to validate the model's appropriateness.

Interactivity of the graphical user interface of the QGIS plugin

The user interface was configured to accommodate the spatial and weather information requirements for the specific location. Eventually, a screen displays the various Python procedures that have been executed. Currently, specific directories need to be filled in to read the CSV file with spatial and weather data. It would be beneficial to have API's connected to facilitate the immediate creation of base maps for specific information by a web server. Additionally, the KNMI pysolar is set up solely in Python for creating the .CSV files from the KNMI .text files. This functionality could also be incorporated.

9.2. Limitations

Accuracy of open data for trees

Due to the restriction on accessing private information from [NEO and Geodan, 2024], the trees, along with their individual additional information such as tree height and tree crown, were generated from openly accessible data provided by [diensten Rotterdam, 2023]. In this scenario, the area of the tree crown could potentially be inaccurately represented in size compared to reality.

Computation memory

For run4 for the case study of Rotterdam, 23 GB is reserved for having the base maps for modeling 1 hour. For the other hour days of the day specifically each 234 KB each have to be generated for the area. It is necessary to have such amount of space available on your computer. The run was ran with The HP Zbook with Intel Core which has a RAM of 16GB with 6 cores, with the potential to run calculations separately. The permanent storage capacity of the PC is 475GB.

10

Conclusions

This research aimed to address the question: "How can a strategy be developed for mitigating heat stress through Physiological Equivalent Temperature model while ensuring a livable environment for vulnerable groups in Bospolder Tussendijken, Rotterdam, the Netherlands?"
The objective was twofold: to create an interactive tool indicating PET heat stress in urban areas of the Netherlands and to design a strategy specifically tailored to Bospolder Tussendijken. This part of the joint thesis focused on reproducable tool to indicate the PET in Dutch cities..
The tool aims to model the Physiological Equivalent Temperature (PET) for outdoor thermal comfort. An analysis of available software, particularly the PET developed by Deltaplan at Wageningen University [Programme, 2018], was conducted. To enhance reproducability Agile guidelines are integrated. Sharing data via an open platform was deemed optimal, facilitated by a QGIS plugin opening the Python code. A sensitivity analysis for wind modulation was performed, and PET was applied to assess thermal comfort in the area.

10.1. Sub research questions answered

1. **Which thermal comfort models do express heat stress?**
   Several models have evolved from the well-known Physiological Equivalent Temperature (PET) model, ranging from thermostatically PMV and MEMI to a more universally comprehensible PET model across disciplines. These models consider three key influences: dynamic climate data, static built environment data, and standardized physiological performances. Given the standardization of the PET model in the Netherlands, it remains the appropriate choice for modeling the thermal comfort of citizens in the country. PET serves as a comparison between complex outdoor conditions and a typical steady-state indoor environment, aligning indoor energy balance with outdoor mean skin temperature and sweat rate for simplified thermal comfort assessment. However, PET is a static model for indoor thermal environments, whereas UTCI and WBGT incorporate factors such as clothing and metabolic rate, providing more comprehensive overview.

2. **Which software is available for open use for modeling heat stress?**
   The software requirements were assessed if it was a reproducible manner of retrieving the information with the connection between knowing, wanting and acting see Table3.2. Therefore it is necessary to indicate the critic areas and also being able to intervene in the public space. Next to that it should be reproducible for a broader audience. Therefore the AGILE requirements of reproducability are important which are divided in input, methods and results. Also the requirements of the influencing factors of the urban environment which can be changed by the urban designer should be integrated in the software. Small fluctuations of evaporation surfaces or shadow are important to model. The usability for multiple users the scalability of the area is important as well as the runtime of the software.

3. **In what way could the reproducability of [Koopmans et al., 2020] be improved?**
   The Wageningen University scientific research institute has incorporated reproducability measures in its PET research. A conclusion assessment, rated from 0 to 3 on reproducability, is presented in Table 4.2. To enhance reproducability in input, methods and results. Input datasets are well documented but not all publicly available. For the methods, various pre-processing steps are necessary for data

preparation. The method and processing steps are well-documented in [Koopmans et al., 2020], yet due to the lack of funding prohibits making the software open-source for third-party use. Tools like ndvi_calculator, svf_footprint, and others (detailed in Appendix H) are employed, with QGIS modifications posing workflow challenges. Parameters are favored for re-factorization. The computing environment involves QGIS, Python, UMEP plugin, and Excel, with Python for calculations and Excel for weather data. QGIS is solely utilized for visualization. Results, available in Appendix A of [Koopmans et al., 2020], are accessible upon request. For input data, as methods as results improvements could be made. In the context of agile reproducability, each improvement enhances the sharing of information across multiple disciplines.

4. **What is the sensitivity of the wind computation?**
   The wind sensitivity on block size resulted different resolutions with high positive correlation. for the operability for larger scale research areas the wind sensitivity with the block size of 25 meter could be easily used to determine a brief overview of the results. Due to the refactoring the fdf factor of the buildings needed to be adjusted to a lower value to be calibrated with the end result. Refinement in the fdf building and other in-between steps in the process are required in order to come to a higher PET resemblance with the code of Koopmans (2020).

5. **How can the PET be applied on in Rotterdam for urban design interventions?**
   With the QGIS plugin, urban planners can conduct spatial-temporal analysis for areas up to 10 km2. Various models are used to assess the current situation and test proposed heat mitigation measures outlined in the spatial report. The PET simulator model of Bospolder Tussendijken is used to simulate heat stress on both summer and warm days. Additionally, a model is created to determine thermal accessibility based on a thermal comfort level of 21 PET °C, suggesting mitigation measures for specific roads. The urban design requirements are tested on the influence of heat mitigation measures, emphasizing radiation reduction, evaporative materials and considering scale dependencies. Ultimately, the PET model and r.walk are used to assess the goals outlined in the spatial report, allowing for scenario planning and serving as a open access communication tool for stakeholders involved in urban mitigation efforts. Design interventions can be tested by modifying base map input data. However, this process requires a good understanding of adapting the input base maps, which are well documented in this thesis.

10.2. Conclusion

The utilization of thermal comfort models, including the Physiological Equivalent Temperature (PET) model and its variations, plays a crucial role in expressing heat stress. These models incorporate dynamic climate data, static built environment data, and standardized physiological performances to assess thermal comfort. While PET remains a standard choice for modeling thermal comfort in the Netherlands due to its standardization and comparison between indoor and outdoor environments, models like UTCI and WBGT provide a more comprehensive overview by considering factors such as clothing and metabolic rate.

Regarding available software for open use in modeling heat stress, the reproducability of the software is essential for broader accessibility and intervention in public spaces. Software should meet AGILE requirements for reproducability, considering input, methods, and results, as well as integrate factors influencing the urban environment that can be modified by urban designers. Usability for multiple users, scalability of the area, and runtime of the software are also crucial factors to consider.

While efforts have been made to incorporate reproducability measures in PET research, improvements are needed in input, methods, and results to enhance reproducability further. This includes making input datasets publicly available, documenting pre-processing steps for data preparation, and addressing challenges in software accessibility due to funding constraints.

The wind sensitivity on block size resulted different resolutions with high positive correlation. for the operability for larger scale research areas the wind sensitivity with the block size of 25 meter could be easily used to determine a brief overview of the results. Due to the refactoring the fdf factor of the buildings needed to be adjusted to a lower value to be calibrated with the end result. Refinement in the fdf building and other in-between steps in the process are required in order to come to a higher PET resemblance with the code of Koopmans (2020).

Spatial-temporal modeling using tools like QGIS plugin enables urban planners to analyze areas for heat stress and assess proposed mitigation measures. By simulating heat stress and determining thermal accessibility, intervention areas in public spaces can be identified and tested for effectiveness. Design interventions focused on radiation reduction, wind promotion, and evaporative materials can be evaluated using PET models, facilitating scenario planning and communication among stakeholders involved in urban mitigation efforts.

In conclusion, a reproducible PET tool can significantly aid in testing and designing for heat mitigation by providing comprehensive assessments of thermal comfort, identifying intervention areas in public spaces, and evaluating the effectiveness of mitigation measures. However, continuous improvements in software reproducability, sensitivity analysis, and spatial-temporal modeling are necessary to enhance the tool's utility and accessibility for urban planning and design. Also other influences next to solar radiation, evaporation and wind as mentioned in [van Esch, 2015] could be implemented to enhance other mitigation measures. It evaluates urban designs using the Physiological Equivalent Temperature plugin, with future potential applications in modeling PET night urban heat island simulations and improving communication among stakeholders. The research aligns with field Geomatics, using GIS and spatial analysis techniques to address urban environmental challenges. The project contributes to understanding the health implications of urban micro climates and the potential effects of temperature increases, informing policymakers and urban planners about creating healthy and sustainable urban environments.

10.3. Additional Points of Growth from this Research

Through this thesis, I have also learned to interact with various experts in the field, including academics from Wageningen, Sytse Koopmans, and Gert-Jan Steeneveld. The networking event at the HvA symposium "Hot Issues" also contributed to the perspective of different municipalities and their approach to heat management in their cities [Hogeschool van Amsterdam, 2023]. Additionally, discussions with researchers at the municipality of Rotterdam, such as Merel Scheltema, and advisor Andre de Wit at Witteveen en Bos, provided an interesting interdisciplinary mix of information alongside my interdisciplinary background in the study Geomatics and Urbanism on this issue. Noteworthy in this report is also the alternation of research by design. Through my interaction with the evidence-based modeling of PET, there is a significant analytical aspect to this research. The design partly awaited the outcomes of the PET. Therefore, the design part entered the process later. This allowed me to discover firsthand how research by design took place in the design.

10.4. Conclusion joint degree

The aim was to develop an reproducible spatial-temporal tool for indicating thermal comfort in urbanized areas in the Netherlands, as well as to create a strategic design for the context-specific area for Bospolder Tussendijken in Rotterdam. The research was part of a cycle of 3 steps (see figure 8.1). First the development of the PET simulator tool which made it possible to have reproducability for third-party use. Second it created the PET heat stress maps for analysis for the urban design. Third step were the urbanism requirements for design and the creation of the design. Third part was the reflection for further development of the PET tool and future work. The PET simulator tool helped eventually to model the heat stress in the application case study of Rotterdam. Through the analysis of the input datasets, methods and results, it emerged that the methods should be publicly available with integration of computational environment. A plugin has been created in QGIS to open the Python code to a larger audience. A sensitivity analysis has been carried out for the wind modulation. Ultimately, the PET was made readable and applied to the accessibility of the area. For designing the urbanism part formulated liveability requirements for design implementations. From the literature liveability is subdivided in physical liveability and social liveability. The physical liveability is accessibility should be guaranteed despite the increase of days above 25 degrees for vulnerable groups. Next to that the continuity of the mitigation measures are the most effective since it is scale dependent. Also to keep the mitigating effects functioning it is important that the mitigation measures are durable depended on the practical implementations. To make it social appropriate a walk able environment should be supported and enough social spaces should be available for vulnerable groups. Thirdly the tool evaluated the design implementations on their effectiveness which leads to additional research of the design and future work. At the moment shadow is the most contributing factor for heat mitigation. Future work to improve heat mitigation is the integration of additional heat mitigation measures, next to solar radiation, and vegetation, measures or improving the wind in the PET simulator design could enhance its performance. In addition, PET simulator should be better design and analysis integrated without too much effort for modifying the input files for the designer, in order to make it more third-party use proof. The plugin has great prospects for future potential applications in modeling PET such as night urban heat island simulations and improving communication among stakeholders. The research aligns with field Geomatics and Urbanism, using GIS and spatial analysis techniques to address urban environmental challenges. The project contributes to understanding the health implications of urban micro climates and the potential effects of temperature increases, informing policymakers and urban planners about action for creating healthy and sustainable urban environments.

11

Future research

The identification of areas for improvement and the emergence of new research questions serve as the basis for generating recommendations for future research. This section delves into these recommendations and proposes potential inquiries for each of the identified research topics.

11.1. Points of improvement

Refinement input data trees

This research is based on reproducability. Another open source was used for the trees. Since the area has an influence on the frontal density area for the wind, a more accurate representation of trees would be suitable. Through point cloud segmentation of trees this could be achieved.

- To what extent could tree point cloud segmentation result in calculating an accurate and open accessible PET?

Refinement wind

The current wind modeling only takes into account four wind directions and no wind. However, it's possible that diagonal wind flows may occur. By following upcoming steps, the horizontal and vertical components of a given wind direction are determined.

1. Calculate Components:

$$\text{Horizontal Component} = \text{Magnitude} \times \sin\left(\frac{\theta}{180}\pi\right)$$
$$\text{Vertical Component} = \text{Magnitude} \times \cos\left(\frac{\theta}{180}\pi\right)$$

2. Magnitude Calculation (if needed):

$$\text{Magnitude} = \sqrt{\text{Horizontal Component}^2 + \text{Vertical Component}^2}$$

3. Normalization (if needed): If you want to normalize the resulting vector to have a unit magnitude:

$$\text{Normalized Component} = \frac{\text{Component}}{\text{Magnitude}}$$

The current software models only take into account the effect of wind based on the variations in slope of buildings and trees within a large averaged area using the Macdonald method (Macdonald, 1998). However, this approach does not accurately represent the real wind flow. Incorporating computational fluid dynamics into the research would provide a more accurate model of real wind flows. In de Jongh's master thesis [de Jongh, 2021], he suggests a method to integrate a Voronoi approach to estimate the computational fluid dynamic model of wind flow in a QGIS environment. His research is also based in Rotterdam. Implementing this calculation method could lead to a more accurate modeling of wind flow through streets by accounting for skimming flows which are described in several literature of urban design requirements [van Esch, 2015] and [Lenzholzer, 2018].

– To what extent could (voronoi) CFD modeling improve the wind calculation in the PET simulator?

Health experts integrated in research

This research could have more of a societal value if there were a link between health experts and the understanding of a better urban environment. This research attempted to research accessibility based on thermal comfort. If there is a link between to what extent people can endure heat there would be more of a scientific use of the PET modeling. Right now, ENVI-MET developed a pedestrian dynamic comfort linking multiple models like PET and WBGT to model the thermo-physiological experience to the urban environment. [Bruse, 2023].

– To what extent could participants validate the endure times of different PET values in the urban environment?

Sky view factor updated design model

The comparison between shadow and no shadow in the street using Sandra Lenzholzer's model helps determine whether design decisions should focus on public spaces or be addressed with buildings [Lenzholzer, 2018]. The creation of shadows and obstruction of the sky lead to higher heat storage in the streetscape. Currently, only the shadows are being updated, not the skyview factor.

– How could the skyview factor have influence on the calculation of the urban morphology calculation for updating design interventions?

Pedestrian walking choice based on heat exposure in the street

The research aimed to reduce heat on the most frequently used routes in the neighborhood, focusing on the shortest path to the destination. However, pedestrians may not always choose the shortest route. Therefore, further research is required to understand the factors that influence pedestrians' decisions when choosing which streets to walk. This understanding could help identify pedestrian preferences for implementing heat reduction methods.

– How are pedestrians influenced in order to take/change roads towards destinations on a summer day in comparison to a warm day?

Climate scenarios integrated in research

The code provided by [Koopmans et al., 2020] also had an prediction for the possible different climate scenarios. This was left out in the research.

– What is the remarkable change in climate data with the KNMI climate scenarios in contrast to current situation?

Computation larger areas

When dealing with larger areas, Python may not provide sufficient computational capabilities. In such cases, using C++ can be highly beneficial for dividing the computation task of computing Physiological Equivalent Temperature (PET) for larger regions, like the Netherlands. By incorporating parallel processing techniques, it becomes necessary to divide the Netherlands into smaller tiles or regions, with each tile representing a manageable portion of the entire area.

C++ offers robust support for multi-threading, enabling the creation and management of multiple threads of execution within a single program. Leveraging this capability, multi-threading can be employed to distribute the computation of PET across numerous tiles concurrently. Each thread can then independently compute PET for a specific tile, thereby utilizing the multi-core architecture of modern CPUs to significantly enhance performance.

This approach not only speeds up the computation process but also optimally utilizes the available computational resources. Additionally, it allows for efficient scaling, enabling the handling of even larger areas or datasets with minimal additional effort. By seamlessly integrating parallel processing techniques, C++ empowers researchers and practitioners to tackle complex computational tasks with unparalleled efficiency and effectiveness.

– To what extent could C++ improve the computation time of the PET calculation?

Geospatial database

Storing files directly on the device can be challenging when handling large files and can limit functionality. QGIS faces difficulties in effectively managing and storing raster data. According to [Langran, 1989], GIS architecture issues include storage, modeling spatial changes, clustering, data access, algorithms, and system design individuality.

GIS architecture is inefficient for storage and management tasks. Updating files for spatial modeling requires manual effort and is not understandable by all third party users. GIS still has inefficient clustering techniques, which hinder parallel processing and indexing. Implementing improvements in this area could enhance scalability and performance in large-scale temporal GIS applications. Efficient algorithms are crucial for quick data access and responsive query times. Unlike GIS, geospatial databases are available and can be integrated to achieve spatial-temporal accuracy. Geospatial databases have the capability to store and manage data more effectively. Integrating the current plugin involves writing Python code to establish connections with geospatial databases like PostGIS. Storing data in such databases makes it possible to seamlessly update spatial and temporal information for multiple users. Consequently, QGIS plugins can effectively operate with the data stored in these databases. Steps to integrate this in the QGIS plugin would be:

1. Establishing Connection with PostGIS: Utilize Python along with the psycopg2 library to establish a connection with your PostGIS database from within your QGIS plugin. Ensure you have the connection parameters such as host, database, username, and password.

2. Retrieving and Visualizing Data: Upon successfully connecting to the PostGIS database, execute SQL queries to retrieve the desired raster data. Subsequently, visualize this data in QGIS by adding them as layers to the map canvas.

3. Adding Interaction: Enhance the functionality of your QGIS plugin by incorporating interaction capabilities, such as data filtering, conducting analyses, or editing data within the PostGIS database.

4. Publishing Changes to PostGIS: If your QGIS plugin allows for editing data retrieved from PostGIS, ensure that you send any modifications back to the database. This may involve executing SQL update or insert queries to enact the changes.

Future research could implement this strategy.

– In what way can POSTGIS be connected to PET Simulator plugin in order to improve the computation of the scalability of the modeling area?

Performance of vegetation for urban heat

Through satellite imagery data the performance of vegetation, NDVI in urban environments could be measured throughout the summer period and its potential influence on cooling the urban environment. In order to take a more holistic approach, design interventions are also needed to take a more holistic approach to maintaining the health of this vegetation.

11.2. Transferability of the Research

The findings of the research can be applied to other areas in the Netherlands. The reproducability is increased and therefore better to execute on other location, with the required input files. Therefore this research holds great prospects for other applications such as modeling the night situation of urban heat island effect. However, it must be said to be a good design tool several steps in the pre-processing must be adapted.

Bibliography

Jacques Bertin. *Semiology of Graphics.* Esri Press, 1 2011.

Krzysztof Blazejczyk, Gerd Jendritzky, Peter Brode, Dusan Fiala, George Havenith, Yoram Epstein, Agnieszka Psikuta, and Bernhard Kampmann. An introduction to the universal thermal climate index (utci). *Geographia Polonica*, 86(1):5–10, 2013.

Daniela Bruse. Dynamic thermal comfort model for pedestrians, jun 2023. URL https://www.envi-met.com/new-bio-met-dynamic-thermal-comfort/.

Grahame M. Budd. Wet-bulb globe temperature (wbgt)—its history and its limitations. *Journal of Science and Medicine in Sport*, 11(1):20–32, jan 2008. ISSN 14402440. doi: 10.1016/j.jsams.2007.07.003. URL https://linkinghub.elsevier.com/retrieve/pii/S1440244007001478.

CAS. Kaartviewer - klimaateffectatlas, 2020. URL https://www.klimaateffectatlas.nl/nl/.

Leighton Cochran and Russ Derickson. Low-rise buildings and architectural aerodynamics. *Architectural Science Review*, 48(3):265–276, sep 2005. ISSN 0003-8628, 1758-9622. doi: 10.3763/asre.2005.4833. URL http://www.tandfonline.com/doi/abs/10.3763/asre.2005.4833.

H. Daanen. Hete hangijzers: plenaire gedeelte. In *Hete hangijzers: plenaire gedeelte*, Amsterdam, Netherlands, jul 2023. Hogeschool van Amsterdam.

W. de Jongh. Urban morphological analysis for wind potential, 2021. URL https://repository.tudelft.nl/islandora/object/uuid%3Afdbff288-fede-4796-9972-54627af0db77.

Deltares. Crc tool - climate resilient cities deltares, 2020. URL https://www.deltares.nl/en/software-and-data/products/crc-tool-climate-resilient-cities.

diensten Rotterdam. Sb infra bomen mapserver, 2023. URL https://diensten.rotterdam.nl/arcgis/rest/services/SBInfra/Bomen/MapServer.

Anthony Dunne and Fiona Raby. *Speculative Design: Design, Fiction, and Social Dreaming.* MIT Press, Cambridge, MA, 2013.

eesa. Vegetation indices and their interpretation: Ndvi, gndvi, msavi2, ndre, and ndwi, 2024. URL https://www.auravant.com/en/articles/precision-agriculture/vegetation-indices-and-their-interpretation-ndvi-gndvi-msavi2-ndre-and-ndwi/.

Poul O. Fanger. *Thermal comfort: Analysis and applications in environmental engineering.* Danish Technical Pr, Copenhagen, 1970. ISBN 9788757103410.

Dusan Fiala, George Havenith, Peter Bröde, Bernhard Kampmann, and Gerd Jendritzky. Utci-fiala multi-node model of human heat transfer and temperature regulation. *International Journal of Biometeorology*, 56(3):429–441, may 2012. ISSN 0020-7128, 1432-1254. doi: 10.1007/s00484-011-0424-7. URL http://link.springer.com/10.1007/s00484-011-0424-7.

Open Science Framework. Agile reproducible paper guidelines. https://osf.io/cb7z8/, 2022.

Geofabrik. Geofabrik // home, 2020. URL https://www.geofabrik.de/.

Guardian. Spain braced for record april temperature of 39°c as extreme heat causes misery, 2023. URL https://www.theguardian.com/world/2023/apr/27/spain-braced-for-record-april-temperature-of-39c-as-heatwave-causes-misery.

G Havenith. Heat balance when wearing protective clothing. *The Annals of Occupational Hygiene*, 43(5):289–296, jul 1999. ISSN 00034878. doi: 10.1016/S0003-4878(99)00051-4. URL https://linkinghub.elsevier.com/retrieve/pii/S0003487899000514.

Bert G. Heusinkveld, G. J. Steeneveld, L. W. A. Van Hove, C. M. J. Jacobs, and A. A. M. Holtslag. Spatial variability of the Rotterdam urban heat island as influenced by urban land use. *Journal of Geophysical Research: Atmospheres*, 119(2):677–692, jan 2014. ISSN 2169-897X, 2169-8996. doi: 10.1002/2012JD019399. URL https://agupubs.onlinelibrary.wiley.com/doi/10.1002/2012JD019399.

J. Hofman. Keep your hague cool: Mitigating heat stress and the urban heat island effect through urban design, 2022. URL http://resolver.tudelft.nl/uuid:17f937a9-b5e5-4fde-b149-4b1dc004ea51.

Hogeschool van Amsterdam. Terugblik: Hitte in de stad symposium, 2023. URL https://www.hva.nl/city-net-zero/gedeelde-content/nieuws/nieuwsberichten/2023/07/terugblik-hitte-in-de-stad-symposium.html. Accessed: February 23, 2024.

HVA. Coolkit - hva, 2020. URL https://www.hva.nl/kc-techniek/gedeelde-content/contentgroep/klimaatbestendige-stad/resultaten/coolkit.html.

Martin Hämmerle, Tamás Gál, J. Unger, and Andreas Matzarakis. Introducing a script for calculating the sky view factor used for urban climate investigations. *ACTA CLIMATOLOGICA ET CHOROLOGICA*, 44-45:83–92, 01 2011.

P. Höppe. The physiological equivalent temperature - a universal index for the biometeorological assessment of the thermal environment. *International Journal of Biometeorology*, 43(2):71–75, October 1999. ISSN 0020-7128, 1432-1254. doi: 10.1007/s004840050118. URL http://link.springer.com/10.1007/s004840050118.

Kadaster. Pdok download viewer, 2023. URL https://app.pdok.nl/lv/bgt/download-viewer/.

Kadaster. Introductie pdok, 2024. URL https://www.pdok.nl/introductie/-/article/basisregistratie-topografie-brt-topnl.

KNMI. Uurwaarden van weerstations, 0000. URL https://daggegevens.knmi.nl/klimatologie/uurgegevens.

KNMI. Sky view factor of the netherlands - knmi data platform, 2023. URL https://dataplatform.knmi.nl/dataset/access/svf-nl-3.

S. Koopmans, B.G. Heusinkveld, and G.J. Steeneveld. A standardized physical equivalent temperature urban heat map at 1-m spatial resolution to facilitate climate stress tests in the netherlands. *Building and Environment*, 181:106984, aug 2020. ISSN 03601323. doi: 10.1016/j.buildenv.2020.106984. URL https://linkinghub.elsevier.com/retrieve/pii/S0360132320303644.

G. Langran. A review of temporal database research and its use in gis applications. *International journal of geographical information systems*, 3(3):215–232, jul 1989. ISSN 0269-3798. doi: 10.1080/02693798908941509. URL http://www.tandfonline.com/doi/abs/10.1080/02693798908941509.

Joel Lawhead. *QGIS Python Programming Cookbook - Second Edition: Automating Geospatial Development*. Packt Publishing, 2018. ISBN 1787124835.

S. Lenzholzer. *Weather in the city*. NAI booksellers, 2018. ISBN 9789462081987.

Fredrik Lindberg, C.S.B. Grimmond, Andrew Gabey, Bei Huang, Christoph W. Kent, Ting Sun, Natalie E. Theeuwes, Leena Järvi, Helen C. Ward, I. Capel-Timms, Yuanyong Chang, Per Jonsson, Niklas Krave, Dongwei Liu, D. Meyer, K. Frans G. Olofson, Jianguo Tan, Dag Wästberg, Lingbo Xue, and Zhe Zhang. Urban multi-scale environmental predictor (umep): An integrated tool for city-based climate services. *Environmental Modelling and Software*, 99:70–87, Januari 2018. ISSN 13648152. doi: 10.1016/j.envsoft.2017.09.020. URL https://linkinghub.elsevier.com/retrieve/pii/S1364815217304140.

R.W. Macdonald, R.F. Griffiths, and D.J. Hall. An improved method for the estimation of surface roughness of obstacle arrays. *Atmospheric Environment*, 32(11):1857–1864, jun 1998. ISSN 13522310. doi: 10.1016/S1352-2310(97)00403-2. URL https://linkinghub.elsevier.com/retrieve/pii/S1352231097004032.

Andreas Matzarakis and Bas Amelung. Physiological equivalent temperature as indicator for impacts of climate change on thermal comfort of humans. In Madeleine C. Thomson, Ricardo Garcia-Herrera, and Martin Beniston, editors, *Seasonal Forecasts, Climatic Change and Human Health*, pages 161–172. Springer Netherlands, Dordrecht, 2008. ISBN 9781402068768 9781402068775. doi: 10.1007/978-1-4020-6877-5_10. URL http://link.springer.com/10.1007/978-1-4020-6877-5_10.

H. Mayer and P. Hoppe. Thermal comfort of man in different urban environments. *Theoretical and Applied Climatology*, 38(1):43–49, 1987a. ISSN 0177-798X, 1434-4483. doi: 10.1007/BF00866252. URL http://link.springer.com/10.1007/BF00866252.

H. Mayer and P. Hoppe. Thermal comfort of man in different urban environments. *Theoretical and Applied Climatology*, 38(1):43–49, 1987b. ISSN 0177-798X, 1434-4483. doi: 10.1007/BF00866252. URL http://link.springer.com/10.1007/BF00866252.

ENVI met GMBH. High-resolution 3d modeling of urban microclimate with envi-met software, n.d. URL https://www.envi-met.com/.

A. Millyard, J. D. Layden, D. B. Pyne, A. M. Edwards, and S. R. Bloxham. Impairments to thermoregulation in the elderly during heat exposure events. *Gerontol Geriatr Med*, 6:2333721420932432, Jan-Dec 2020. doi: 10.1177/2333721420932432. URL https://doi.org/10.1177/2333721420932432. Published online 2020 Jun 15.

Parham A. Mirzaei. Cfd modeling of micro and urban climates: Problems to be solved in the new decade. *Sustainable Cities and Society*, 69:102839, June 2021. ISSN 22106707. doi: 10.1016/j.scs.2021.102839. URL https://linkinghub.elsevier.com/retrieve/pii/S2210670721001293.

MIT. ud software development urban microclimate, 0000. URL https://urbanmicroclimate.scripts.mit.edu/umc.php.

Peter Moonen, Thijs Defraeye, Viktor Dorer, Bert Blocken, and Jan Carmeliet. Urban physics: Effect of the micro-climate on comfort, health and energy demand. *Frontiers of Architectural Research*, 1(3):197–228, sep 2012. ISSN 20952635. doi: 10.1016/j.foar.2012.05.002. URL https://linkinghub.elsevier.com/retrieve/pii/S2095263512000301.

Wageningen University NEO and Geodan. Boomregister, 2024. URL http://boomregister.nl/.

Ministery of Infrastructure and Waterboard. Deltaplan en ruimtelijke adaptatie. December 2018. URL https://www.deltaprogramma.nl/themas/ruimtelijkeadaptatie/deltaplan.

T. R. Oke. The energetic basis of the urban heat island. *Quarterly Journal of the Royal Meteorological Society*, 108(455):1–24, jan 1982. ISSN 0035-9009, 1477-870X. doi: 10.1002/qj.49710845502. URL https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.49710845502.

Timothy R Oke. *Boundary layer climates*. Routledge, 2002.

Delta Programme. Delta programme, 2018. URL https://www.deltaprogramma.nl/deltaprogramma.

RIVM. Achtergronddocument WGBT en PHS bij GGD-richtlijn mmk: hitte en gezondheid | RIVM, 2023. URL https://www.rivm.nl/documenten/achtergronddocument-wgbt-en-phs-bij-ggd-richtlijn-mmk-hitte-en-gezondheid.

Gianna Stavroulaki, Daniel Koch, Ann Legeby, Lars Hilding Marcus, Alexander Ståhle, and Meta Berghauser Pont. Documentation pst 20191122, 2019. URL http://rgdoi.net/10.13140/RG.2.2.25718.55364.

Marieke van Esch. From thermal comfort to heat mitigation necessity: Informed strategies for mitigating pet heat stress in public spaces for vulnerable groups – a rotterdam case study, 2024. URL https://www.tudelft.nl/.

Marjolein van Esch. Designing the urban microclimate. *A+BE Architecture and the Built Environment*, pages 1–308 Pages, june 2015. doi: 10.7480/ABE.2015.6.905. URL https://journals.open.tudelft.nl/abe/article/view/pijpers.

A

Symbols

symbol	description	unit
A	parameter for interpolation wind profile	-
B	parameter for interpolation wind profile	-
B_b	bowen ratio (sensible heat flux / latent heat flux)	-
d	zero-plane displacement	m
FF10	10-m wind at reference station	ms^{-1}
F_{veg}	vegetation fraction	-
φ	zero-plane displacement	m
$\lambda_{\mathrm{building}}$	frontal area density for buildings	-
λ_{tree}	frontal area density for trees	-
λ_{tot}	total frontal area density	-
H	building height	m
I	infrared value of aerial photo (INFR)	-
PET	Physiological Equivalent Temperature	°C
ϕ	relative humidity at reference station	%
Q_d	diffuse irradiation	$\mathrm{W\,m}^{-2}$
Q_s	solar irradiation at reference station	$\mathrm{W\,m}^{-2}$
R	red value of aerial photo (RGBI)	-
σ	Stefan-Boltzmann constant	$\mathrm{W\,m}^{-2}\,\mathrm{K}^{-4}$
$S_{\downarrow}$	daily average solar irradiation (in kinematic units)	$\mathrm{K\,ms}^{-1}$
Svf	sky view factor	-
τ_a	transmissivity	-
T_a	air temperature	°C
T_{gem}	daily average air temperature	°C
T_{max}	daily average maximum temperature	°C
T_{min}	daily average minimum temperature	°C
T_{ref}	air temperature at reference station	°C
T_w	wet bulb temperature	°C
U	daily average wind speed at reference station	ms^{-1}
$u_{1.2}$	wind reduction at 1.2 m relative to $u_{10} = 1\ \mathrm{ms}^{-1}$	ms^{-1}
u_{10}	reference normalized wind of 1 ms^{-1} representative for open terrain	ms^{-1}
u_{60}	wind at 60-min height (relative to $u_{10} = 1\ \mathrm{ms}^{-1}$), mesowind	ms^{-1}
UHI	urban heat island	°C
UHI_{max}	daily maximum urban heat island	°C
u^{*}	friction velocity	ms^{-1}
u_h	wind speed at roof height	ms^{-1}
z_0	(surface) roughness length	m
z_w	top of the roughness layer	m

B
Python code

B.1. python/pet_parameters.py

```python
#from IPython import get_ipython
#get_ipython().magic('reset -sf')

import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
#------------------------------------------------------------------------------------------

# petcalculate
# purpose: calculate the PET
# input: shadow, urbanheat, wind, svf, svf_mask, ndvi_crop_mask, ndvi_tree_mask
# output: pets

def PET_calculate(stat, dyn, im1, im2, im3, im4, im5, im6, im7):

    TT = dyn.TT                 #TT:    Temperatuur (in 0.1 graden Celsius) op 1.50 m hoogte tijdens de waarneming
    FF = dyn.FF                 #FF:    Windsnelheid (in 0.1 m/s) gemiddeld over de laatste 10 minuten van het afgelopen uur
    Q = dyn.Q                   #Q:     Global solar irradiationGlobale straling (in J/cm2) per uurvak
    Qdif = dyn.Qdif             #Qdif:  Difuse radiation
    sunalt = dyn.sunalt         #sunalt:solar elevation angle
    RH = dyn.RH                 #RH:    Relative Humidity
    diurnal = dyn.diurnal       #diurnal correction factor UHI for Ta

    print('PET.Calculator')
    Bveg = 0.4
    Bnoveg = 3
    stef = 5.67 * 10 ** -8

    sun, meta = GeotifToArray(im1, 1)  # added anders geen ref in shadow
    urban, meta = GeotifToArray(im2, 1)
    wind, meta = GeotifToArray(im3, 1)
    svf, meta = GeotifToArray(im4, 1)
    svf_mask, meta = GeotifToArray(im5, 1)
    mask_vegfra, meta = GeotifToArray(im6, 1)
    trees_2m, meta = GeotifToArray(im7, 1)

    # with open("D:\\tmp\\test.txt", 'wt') as f:
    #     f.write(f"sun, meta {sun, meta}\\n")
    #     f.write(f"urban, meta {urban, meta}\\n")
    #     f.write(f"wind, meta {wind, meta}\\n")
    #     f.write(f"svf, meta {svf, meta}\\n")
    #     f.write(f"svf_mask, meta {svf_mask, meta}\\n")
    #     f.write(f"mask_vegfra, meta {mask_vegfra, meta}\\n")
    #     f.write(f"trees_2m, meta {trees_2m, meta}\\n")


    Ta = urban[:] * diurnal + TT
    Tw = TT * np.arctan(0.15198 * (RH + 8.3137) ** 0.5) + np.arctan(TT + RH) - np.arctan(
        RH - 1.676) + 0.0039184 * RH ** 1.5 * np.arctan(0.023101 * RH) - 4.686

    wind = ((wind - 0.125) * 0.5829 + 0.125) * FF
    wind[wind < 0.5] = 0.5
    wind_temp = np.ravel(wind)
    #wind_res = np.array(wind_temp).transpose()

```

```
    # day
    if Q > 0:
        sun_temp, meta = GeotifToArray(im1, 1)
        sun = sun_temp * (1 - trees_2m[:])

        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw
             + 0.015 * svf[:] * Qdif +
                    0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1
                        - sun[:]) * svf_mask[:]
        PETveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.
                sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bveg + 1.94 * svf[:]) *
                mask_vegfra[:] * sun[:] * svf_mask[:]
        PETnoveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.
                sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bnoveg + 1.94 * svf[:]) * (1 -
                mask_vegfra[:]) * sun[:] * svf_mask[:]

        PET = PETshade + PETveg + PETnoveg

    # night
    else:
        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw
             + 0.015 * svf[:] * Qdif
                    + 0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) *
                        (1 - sun[:]) * svf_mask[:]

        PET = PETshade

    im8 = ArrayToGeotif(PET, meta)
    sun = urban = wind = svf = svf_mask = mask_vegfra = trees_2m = PET = None

    return im8
```

B.2. python/geotiff_creator.py

```
#from IPython import get_ipython
#get_ipython().magic('reset -sf')

import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
#----------------------------------------------------------------------------------------------------

# petcalculate
# purpose: calculate the PET
# input: shadow, urbanheat, wind, svf, svf_mask, ndvi_crop_mask, ndvi_tree_mask
# output: pets

def PET_calculate(stat, dyn, im1, im2, im3, im4, im5, im6, im7):

    TT = dyn.TT                   #TT:    Temperatuur (in 0.1 graden Celsius) op 1.50 m hoogte tijdens de waarneming
    FF = dyn.FF                   #FF:    Windsnelheid (in 0.1 m/s) gemiddeld over de laatste 10 minuten van het afgelopen uur
    Q = dyn.Q                     #Q:     Global solar irradiationGlobale straling (in J/cm2) per uurvak
    Qdif = dyn.Qdif               #Qdif:  Difuse radiation
    sunalt = dyn.sunalt           #sunalt:solar elevation angle
    RH = dyn.RH                   #RH:    Relative Humidity
    diurnal = dyn.diurnal         #diurnal correction factor UHI for Ta

    print('PET.Calculator')
    Bveg = 0.4
    Bnoveg = 3
    stef = 5.67 * 10 ** -8

    sun, meta = GeotifToArray(im1, 1)  # added anders geen ref in shadow
    urban, meta = GeotifToArray(im2, 1)
    wind, meta = GeotifToArray(im3, 1)
    svf, meta = GeotifToArray(im4, 1)
    svf_mask, meta = GeotifToArray(im5, 1)
    mask_vegfra, meta = GeotifToArray(im6, 1)
    trees_2m, meta = GeotifToArray(im7, 1)

    # with open("D:\\tmp\\test.txt", 'wt') as f:
    #     f.write(f"sun, meta {sun, meta}\\n")
    #     f.write(f"urban, meta {urban, meta}\\n")
    #     f.write(f"wind, meta {wind, meta}\\n")
    #     f.write(f"svf, meta {svf, meta}\\n")
    #     f.write(f"svf_mask, meta {svf_mask, meta}\\n")
    #     f.write(f"mask_vegfra, meta {mask_vegfra, meta}\\n")
    #     f.write(f"trees_2m, meta {trees_2m, meta}\\n")


    Ta = urban[:] * diurnal + TT
    Tw = TT * np.arctan(0.15198 * (RH + 8.3137) ** 0.5) + np.arctan(TT + RH) - np.arctan(
        RH - 1.676) + 0.0039184 * RH ** 1.5 * np.arctan(0.023101 * RH) - 4.686

    wind = ((wind - 0.125) * 0.5829 + 0.125) * FF
    wind[wind < 0.5] = 0.5
    wind_temp = np.ravel(wind)
    #wind_res = np.array(wind_temp).transpose()

```

```
    # day
    if Q > 0:
        sun_temp, meta = GeotifToArray(im1, 1)
        sun = sun_temp * (1 - trees_2m[:])

        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw + 0.015 * svf[:] * Qdif +
                    0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1 - sun[:]) * svf_mask[:]
        PETveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bveg + 1.94 * svf[:]) * mask_vegfra[:] * sun[:] * svf_mask[:]
        PETnoveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bnoveg + 1.94 * svf[:]) * (1 - mask_vegfra[:]) * sun[:] * svf_mask[:]

        PET = PETshade + PETveg + PETnoveg

    # night
    else:
        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw + 0.015 * svf[:] * Qdif
                    + 0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1 - sun[:]) * svf_mask[:]

        PET = PETshade

    im8 = ArrayToGeotif(PET, meta)
    sun = urban = wind = svf = svf_mask = mask_vegfra = trees_2m = PET = None

    return im8
```

B.3. python/pysolar1.py

```
# Importing packages

import pvlib
from datetime import datetime as dt
from datetime import timedelta
import pandas as pd
import numpy as np

# -----------------------------------------------------------
# Loading in total knmi file
df_tot = pd.read_csv('Rotterdam_1juli_2015_knmi.csv', parse_dates=['YYYYMMDD'])
# substracting the last line
df_KNMI = df_tot[df_tot.H < 24]
print(df_KNMI)

# -----------------------------------------------------------
# Setting date with hour values
date_time = []
solar_elevation = np.zeros(len(df_KNMI.index))
# calculating the solar altitude and the diffuse irradiation
# Location coordinates for Amsterdam (latitude, longitude)


for i in range(len(df_KNMI.index)):
    date_time.append(
        dt(df_KNMI['YYYYMMDD'].iloc[i].year, df_KNMI['YYYYMMDD'].iloc[i].month,
             df_KNMI['YYYYMMDD'].iloc[i].day,
           df_KNMI['H'].iloc[i], 0, 0))

latitude = 52.3667
longitude = 4.8945

solar_position = pvlib.solarposition.get_solarposition(date_time, latitude,
    longitude)

# Extract solar elevation angle
solar_elevation = solar_position['elevation'].values

# -----------------------------------------------------------
# Calculating the average Watt per square meter from the Q
Qs_av = np.zeros(len(df_KNMI.index))

for i in range(len(df_KNMI.index) - 1):
    Qs_av[i] = 10000 / 3600 * ((df_KNMI['    Q'].iloc[i + 1] - df_KNMI['    Q
        '].iloc[i]) / 2 + df_KNMI['    Q'].iloc[i])

# Calculating atmospheric transmissivity (tau_a)
tau_a = Qs_av / (1367.0 * np.sin(solar_elevation * np.pi / 180))

# Calculating the diffuse irradiation
Qd = np.zeros(len(df_KNMI.index))

for i in range(len(df_KNMI.index)):
    if tau_a[i] < 0.3:
        Qd[i] = Qs_av[i]
    elif tau_a[i] > 0.7:
        Qd[i] = 0.2 * Qs_av[i]
    else:
        Qd[i] = (1.6 - 2 * tau_a[i]) * Qs_av[i]
```

```python

df_KNMI['Qdif'] = Qd


# ---------------------------------------------------------
# calulating the wind, WE and wind direction
def wind_direction(dd, FF):
    if FF >= 1.5:
        wind = True
    else:
        wind = False
    # wind = FF >= 1.5
    if dd < 45 or dd > 315:
        WE = False
        winddir = 'N'
    elif dd < 135:
        WE = True
        winddir = 'E'
    elif dd < 225:
        WE = False
        winddir = 'S'
    elif dd < 315:
        WE = True
        winddir = 'W'
    else:
        winddir = 'C'
    return wind, WE, winddir


# addind the wind, WE and wind direction into pandas series through lists
windlist = []
WElist = []
windirlist = []

for i in range(len(df_KNMI.index)):
    wind, WE, winddir = wind_direction(df_KNMI['   DD'].iloc[i], df_KNMI['   FF'].iloc[i] / 10)
    windlist.append(wind)
    WElist.append(WE)
    windirlist.append(winddir)

df_KNMI['wind'] = windlist
df_KNMI['WE'] = WElist
df_KNMI['winddir'] = windirlist

# ---------------------------------------------------------
# Adding the station names
df_KNMI['station'] = ['Rotterdam'] * len(df_KNMI.index)
# drop unnecessary columns like STN and U
df_KNMI = df_KNMI.drop(columns=['STN'])
# converting the wind and temperature columns
df_KNMI['   FF'] = df_KNMI['   FF'] / 10
df_KNMI['    T'] = df_KNMI['    T'] / 10
# ---------------------------------------------------------
# Diurnal calculation
df_UHI = pd.read_csv('UHI_factors.csv')


def day_night(dates_KNMI, hour_KNMI):
    dateslist = [dt(year=dates_KNMI.year, month=4, day=1), dt(year=dates_KNMI.year, month=4, day=13),
```

```
                 dt(year=dates_KNMI.year, month=4, day=20), dt(year=dates_KNMI.year, month=5, day=20),
                 dt(year=dates_KNMI.year, month=5, day=26), dt(year=dates_KNMI.year, month=7, day=11),
                 dt(year=dates_KNMI.year, month=7, day=31), dt(year=dates_KNMI.year, month=8, day=22),
                 dt(year=dates_KNMI.year, month=8, day=31), dt(year=dates_KNMI.year, month=9, day=25),
                 dt(year=dates_KNMI.year, month=9, day=28), dt(year=dates_KNMI.year, month=9, day=30)]
    UHIlist = ['5/18', '5/19', '4/19', '4/20', '3/20', '4/20', '4/19', '5/19', '5/18', '5/17', '6/17']
    for i in range(len(dateslist) - 1):
        if dates_KNMI >= dateslist[i] and dates_KNMI < dateslist[i + 1]:
            diurnal = df_UHI[UHIlist[i]][hour_KNMI]
            sunrise, sunset = UHIlist[i].split('/')
            print(sunrise, sunset)
            if hour_KNMI >= int(sunrise) and hour_KNMI <= int(sunset):
                daynight = 'day'
            break
        else:
            daynight = 'night'
            diurnal = 1

    return daynight, diurnal


# addind the wind, WE and wind direction into pandas series through lists
daynightlist = []
diurnallist = []

for i in range(len(df_KNMI.index)):
    daynight, diurnal = day_night(df_KNMI['YYYYMMDD'].iloc[i], df_KNMI['H'].iloc[i])
    daynightlist.append(daynight)
    diurnallist.append(diurnal)

df_KNMI['daynight'] = daynightlist
df_KNMI['diurnal'] = diurnallist


# --------------------------------------------------------

def min_max(df_KNMI, date_time):
    # date = date_time[0]

    list_temperature_inperiod = []
    list_wind_inperiod = []
    list_max_temp = []
    list_min_temp = []
    list_av_wind = []

    for j in range(0, len(df_KNMI.index), 24):
        date = date_time[j]
        print(f'date {date}')

        av_wind_cum = 0
        temperature_inperiod = []
        wind_inperiod = []
        for i in range(len(df_KNMI.index)):
```

```
            # Calculate period start
            period_start = dt(year=date.year, month=date.month, day=date.day,
                hour=9)

            # Calculate period end
            period_end = date + timedelta(days=1)
            period_end = period_end.replace(hour=8)

            if date_time[i] >= period_start and date_time[i] <= period_end:
                temperature_inperiod.append(df_KNMI['    T'].iloc[i])
                wind_inperiod.append(df_KNMI['   FF'].iloc[i])
                av_wind_cum += df_KNMI['   FF'].iloc[i]
                #   print(date, wind_inperiod)

        max_temp = np.max(np.array([temperature_inperiod]))
        min_temp = np.min(np.array([temperature_inperiod]))
        av_wind = av_wind_cum / len(wind_inperiod)

        list_max_temp.append(max_temp)
        list_min_temp.append(min_temp)
        list_av_wind.append(av_wind)
    list_temperature_inperiod.append(temperature_inperiod)
    list_wind_inperiod.append(wind_inperiod)
    # print('length', list_wind_inperiod)
    return list_max_temp, list_min_temp, list_av_wind


list_max_temp, list_min_temp, list_av_wind = min_max(df_KNMI, date_time)

for i, max_temp in enumerate(list_max_temp):
    # Filter timestamps for the current day
    mask = (df_KNMI['YYYYMMDD'].dt.date == df_KNMI.loc[i * 24, 'YYYYMMDD'].date
        ())
    # Assign the daily maximum temperature to all hourly timestamps for the
        current day
    df_KNMI.loc[mask, 'Tmax'] = max_temp

for i, min_temp in enumerate(list_min_temp):
    # Filter timestamps for the current day
    mask = (df_KNMI['YYYYMMDD'].dt.date == df_KNMI.loc[i * 24, 'YYYYMMDD'].date
        ())
    # Assign the daily maximum temperature to all hourly timestamps for the
        current day
    df_KNMI.loc[mask, 'Tmin'] = min_temp

for i, av_wind in enumerate(list_av_wind):
    # Filter timestamps for the current day
    mask = (df_KNMI['YYYYMMDD'].dt.date == df_KNMI.loc[i * 24, 'YYYYMMDD'].date
        ())
    # Assign the daily maximum temperature to all hourly timestamps for the
        current day
    df_KNMI.loc[mask, 'FFavg'] = av_wind
# -------------------------------------------------------
# Writing the csv away
df_KNMI.to_csv('Qd_results.csv')
```

B.4. python/get_svf.py

```
import requests
import sys


def download_file_from_temporary_download_url(download_url, filename):
    try:
        with requests.get(download_url, stream=True) as r:
            r.raise_for_status()
            with open(filename, "wb") as f:
                for chunk in r.iter_content(chunk_size=8192):
                    f.write(chunk)
    except Exception:
        sys.exit(1)

    print(f"Successfully downloaded dataset file to {filename}")


def main():
    # Parameters
    base_url = "https://api.dataplatform.knmi.nl/open-data/v1"
    api_key = "
        eyJvcmciOiI1ZTU1NGUxOTI3NGE5NjAwMDEyYTNlYjEiLCJpZCI6ImE3NDdjMjVjMWRlNTQ3ZjdhMjM3Zm
        "
    dataset_name = "SVF_NL"
    dataset_version = "3"

    files = [
        "37EZ2.tif",
        "37FZ1.tif",
        "37FZ2.tif",
        "37GN2.tif",
        "37HN1.tif",
        "37HN2.tif",
    ]

    for filename in files:
        filename = filename.lower()
        filename = "SVF_r" + filename

        # get temporary download url
        endpoint = f"{base_url}/datasets/{dataset_name}/versions/{
            dataset_version}/files/{filename}/url"
        print(endpoint)
        get_file_response = requests.get(endpoint, headers={"Authorization":
            api_key})
        j = get_file_response.json()
        url = j['temporaryDownloadUrl']

        # with the url download the file
        download_file_from_temporary_download_url(url, filename)


if __name__ == "__main__":
    main()
```

B.5. python/fraction_area_buildings_treeregr.py

```python
import numpy as np
from PIL import Image
import multiprocessing as mp
from .pet_parameters import window_footprint, writer, wind_direction
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite, ArrayWrite
#-----------------------------------------------------------------------------------------------
# fractionareabuildingstreeregr
# purpose: calculate wind speed u1.2
# input: buildings_mask, buildings_height, trees_ahn, trees_mask
# output: wind_direction
#-----------------------------------------------------------------------------------------------


def meancal(a, size):

    mean = 0
    for j in range(size):
        mean += a[j]
    return mean / size

def myMean(A):

    m,n = A.shape
    pool = mp.Pool()
    rowMean = [pool.apply(meancal, args=(A[i,:], n)) for i in range(m)]
    mean = meancal(rowMean, m)
    pool.close()
    return mean

def FaBuildingTree(stat, dyn, im1, im2, im3, im4):

    print('FaBuildingTree.Calculator')

    #f = open('d:/tmp/aab.dat', 'wt')

    # parameters
    k = 0.4
    z0_grass = 0.03
    refwind = 1 / 0.63501
    red_grass = np.round(refwind * np.log(1.2 / z0_grass) / np.log(10 / z0_grass), 2)
    red_60_10 = np.log(10 / z0_grass) / np.log(60 / z0_grass)
    buildingfactor = 0.2 #was 0.6
    treefactor = 0.27 #was 0.27
    winddir = dyn.winddir
    WE = dyn.WE
    wind_on = dyn.wind
    FF = dyn.FF

    # fine scale extended area = research area + boundary
    # size must by the same for im1, im2, im3, im4
    building_height_fine, meta1 = GeotifToArray(im1, 1)
    mask_building_fine, meta2 = GeotifToArray(im2, 1)
    tree_height_fine, meta3 = GeotifToArray(im3, 1)
    mask_tree_fine, meta4 = GeotifToArray(im4, 1)
    metafine = meta1

```

```
    # check fine scale extended area
    for i in range(metafine[3]):
        for j in range(metafine[4]):
            if building_height_fine[i,j] < 1e-3:
                building_height_fine[i, j] = 0
            else:
                mask_building_fine[i, j] = 1
            if tree_height_fine[i,j] < 1e-3:
                tree_height_fine[i, j] = 0
            else:
                mask_tree_fine[i, j] = 1

    '''
    metafine = [3,5,1,16,18]
    building_height = np.zeros((metafine[3],metafine[4])) #nrow,ncol y,x
    mask_building = np.zeros((metafine[3],metafine[4]))
    tree_height = np.zeros((metafine[3],metafine[4]))
    mask_tree = np.zeros((metafine[3],metafine[4]))
    building_height_fine[5,3] = 20
    mask_building_fine[5,3] = 1
    tree_height_fine[5,3] = 20
    mask_tree_fine[5,3] = 1
    building_height_fine[6, 6] = 10
    mask_building_fine[6, 6] = 1
    tree_height_fine[6, 6] = 10
    mask_tree_fine[6, 6] = 1
    building_height_fine[8, 5] = 30
    mask_building_fine[8, 5] = 1
    tree_height_fine[8, 5] = 30
    mask_tree_fine[8, 5] = 1
    stat.nrow = 4
    stat.ncol = 6
    stat.cellsize = 1
    stat.blocksize = 2
    stat.xmin = 9
    stat.ymin = 11
    dyn.winddir = 'E'
    dyn.upwind = 6
    dyn.sidewind = 2
    dyn.downwind = 4
    dyn.nowind = 100
    '''

    # transform fine scale extended area to coarse scale extended area
    scale = int(stat.blocksize / stat.cellsize)
    nrow = int(metafine[3] / scale)
    ncol = int(metafine[4] / scale)
    meta = [metafine[0], metafine[1], stat.blocksize, nrow, ncol]
    building_height = np.zeros((meta[3], meta[4]))
    mask_building = np.zeros((meta[3], meta[4]))
    tree_height = np.zeros((meta[3], meta[4]))
    mask_tree = np.zeros((meta[3], meta[4]))
    building_weight = np.zeros((meta[3], meta[4]))
    tree_weight = np.zeros((meta[3], meta[4]))

    for i in range(meta[3]):
        istart = i * scale
        iend = istart + scale - 1
        iiend = iend
        if i < meta[3] - 1:
            iiend = iend + 1
```

```
        for j in range(meta[4]):
            jstart = j * scale
            jend = jstart + scale - 1
            jjend = jend
            if j < meta[4] - 1:
                jjend = jend + 1

            building_area = np.mean(mask_building_fine[istart: iend + 1, jstart
                : jend + 1])
            if building_area > 1e-2:
                building_height[i,j] = np.mean(building_height_fine[istart:
                    iend + 1, jstart: jend + 1]) / building_area
                mask_building[i, j] = 1.0
            tree_area = np.mean(mask_tree_fine[istart: iend + 1, jstart: jend +
                 1])
            if tree_area > 1e-2:
                tree_height[i, j] = np.mean(tree_height_fine[istart: iend + 1,
                    jstart: jend + 1]) / tree_area
                mask_tree[i, j] = 1

            if wind_on:
                if WE: # east-west or west-east wind
                    for m in range(istart, iend + 1, 1):
                        for n in range(jstart, jjend, 1):
                            building_weight[i, j] += abs(building_height_fine[m
                                , n + 1] - building_height_fine[m, n]) * 0.5
                            tree_weight[i, j] += abs(tree_height_fine[m, n + 1]
                                 - tree_height_fine[m, n]) * 0.5

                else: # north-south or south-north wind
                    for n in range(jstart, jend + 1, 1):
                        for m in range(istart, iiend, 1):
                            building_weight[i, j] += abs(building_height_fine[m
                                 + 1, n] - building_height_fine[m, n]) * 0.5
                            tree_weight[i, j] += abs(tree_height_fine[m + 1, n]
                                 - tree_height_fine[m, n]) * 0.5

            else: # no wind
                for m in range(istart, iend + 1, 1):
                    for n in range(jstart, jjend, 1):
                        building_weight[i, j] += abs(building_height_fine[m, n
                            + 1] - building_height_fine[m, n]) * 0.5
                        tree_weight[i, j] += abs(tree_height_fine[m, n + 1] -
                            tree_height_fine[m, n]) * 0.5

                for n in range(jstart, jend + 1, 1):
                    for m in range(istart, iiend, 1):
                        building_weight[i, j] += abs(building_height_fine[m +
                            1, n] - building_height_fine[m, n]) * 0.5
                        tree_weight[i, j] += abs(tree_height_fine[m + 1, n] -
                            tree_height_fine[m, n]) * 0.5

            #f.write(f'i {i} j {j} -> {istart} {iend} - {jstart} {jend} ->
                building {building_weight[i, j]} tree {tree_weight[i, j]}\n')



    #  research area coarse
    nrow = int(stat.nrow / scale)
    ncol = int(stat.ncol / scale)
    metadata = [stat.xmin, stat.ymin, stat.blocksize, nrow, ncol]
```

```
    wind_2d = np.zeros((nrow, ncol))

    # (moving) footprint area coarse
    jleft, jright, iup, idown = window_footprint(dyn.winddir, dyn.upwind, dyn.sidewind, dyn.downwind, dyn.nowind, stat.blocksize)
    total_area = (jleft + jright + 1) * (iup + idown + 1) * scale**2 # number of large blocks in footprint area

    # upper left cell of the research area in extended research area coordinates
    iref = int((stat.ymin - meta[1]) / meta[2])
    jref = int((stat.xmin - meta[0]) / meta[2])

    # calculate wind scaling map
    for i in range(nrow):
        istart = i + iref - idown
        iend = i + iref + iup
        for j in range(ncol):
            jstart = j + jref - jleft
            jend = j + jref + jright

            switch = False
            building_area = np.mean(mask_building[istart: iend + 1, jstart: jend + 1])
            tree_area = np.mean(mask_tree[istart: iend + 1, jstart: jend + 1])

            if building_area > 0:
                building_height_mean = np.mean(building_height[istart: iend + 1, jstart: jend + 1]) / building_area
                switch = True
            else:
                building_height_mean = 0

            if tree_area > 0:
                tree_height_mean = np.mean(tree_height[istart: iend + 1, jstart: jend + 1]) / tree_area
                tree_height_regr = np.max(7.721 * tree_height_mean ** 0.5, 0)
                switch = True
            else:
                tree_height_mean = 0
                tree_height_regr = 0

            if switch == True:
                height_com_pre = max((building_height_mean * building_area + tree_height_regr * tree_area * treefactor /
                                      buildingfactor) / (building_area + tree_area * treefactor / buildingfactor), 4)
            else:
                height_com_pre = 4.0

            # calculate building and tree fronts for a cell using its window (1 no blockage, 0 fully blocked)
            tree_front = 0
            building_front = 0

            for m in range(istart, iend + 1, 1):
                for n in range(jstart, jend + 1, 1):
                    #====================================================================
                     building_front += building_weight[m, n] * buildingfactor
```

```
                    tree_front += tree_weight[m, n] * treefactor

            # fit for ahn tree to treefile (bomenbestand)
            tree_regr = 45.45 * (tree_front ** 0.5)
            front_regr = building_front + tree_regr

            if front_regr > 25 and switch:  # was 25  bij hele kleine oppervlakten gewoon op 0 laten, moet hoogte hebben zit ook in BW script
                height_com = max(height_com_pre, 4)
                lambda1 = min(front_regr / total_area + 0.015, 0.33)

                # frontal surface density
                if lambda1 < 0.08:
                    z0 = 0.048 * height_com # (surface roughness length)
                    d = 0.066 * height_com # (zero-plane displacement)
                    zw = 2 * height_com # (top of the roughness layer)
                    A = -0.35 * height_com # parameter for interpolation wind profile
                    B = 0.56 # parameter for interpolation wind profile
                elif lambda1 < 0.135:
                    z0 = 0.071 * height_com
                    d = 0.26 * height_com
                    zw = 2.5 * height_com
                    A = -0.35 * height_com
                    B = 0.50
                elif lambda1 < 0.18:
                    z0 = 0.084 * height_com
                    d = 0.32 * height_com
                    zw = 2.7 * height_com
                    A = -0.34 * height_com
                    B = 0.48
                elif lambda1 < 0.265:
                    z0 = 0.08 * height_com
                    d = 0.42 * height_com
                    zw = 1.5 * height_com
                    A = -0.56 * height_com
                    B = 0.66
                else:
                    z0 = 0.077 * height_com
                    d = 0.57 * height_com
                    zw = 1.2 * height_com
                    A = -0.85 * height_com
                    B = 0.92

                # some additional computations
                ustar = refwind / red_60_10 * k / np.log((60 - d) / z0)
                uzw = ustar / k * np.log((zw - d) / z0)
                uh = uzw - ustar / B * np.log((A + B * zw) / (A + B * height_com))
                wind_2d[i, j] = min(uh * np.exp(9.6 * lambda1 * (1.2 / height_com - 1)), red_grass)
            else:
                wind_2d[i, j] = red_grass

    im = ArrayToGeotif(wind_2d, metadata)
    building_height = tree_height = mask_tree = mask_building = wind_2d = wind_notree_2d = wind_tree_2d = None

    #f.close()

```

```
    return im
```

B.6. python/ndvi_infr_large.py

```
import numpy as np
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
#-----------------------------------------------------------------------------------------------

# ndvi_infra_large
# purpose: create the ndvi from rgb and infr imagery
# input: lufo_rgb, lufo_infr, water_mask, tree_mask
# output: 'ndvi', 'vegfra', 'ndvi_crop_mask', ndvi_tree_mask'
#-----------------------------------------------------------------------------------------------

def Ndvi_infr_large(stat_parameters, dyn_parameters, rgb, infr, water_mask, tree_mask):

    print('Ndvi_infr_large.Calculator')

    wind_2d = np.zeros(shape=(0, 3))

    xmin = stat_parameters.xmin
    xmax = stat_parameters.xmax
    ymin = stat_parameters.ymin
    ymax = stat_parameters.ymax

    ndvi_infr_2d = np.zeros(shape=(0, 3))
    lufo_rgb, meta = GeotifToArray(rgb, 3)
    lufo_infr, meta = GeotifToArray(infr, 3)
    r = lufo_rgb[:, :, 0].astype(int)
    g = lufo_rgb[:, :, 1].astype(int)
    b = lufo_rgb[:, :, 2].astype(int)
    infr = lufo_infr[:, :, 0].astype(int)
    ndvi_infr = (infr - r) / (infr + r)
    ndvi_infr[ndvi_infr < 0] = 0
    arr = ndvi_infr

    im1 = ArrayToGeotif(arr, meta)
    h = meta[3]
    w = meta[4]

    water_mask, meta = GeotifToArray(water_mask, 1)
    day = np.zeros((h, w), dtype=float)
    night = np.zeros((h, w), dtype=float)
    for i in range(h):
        for j in range(w):
            if arr[i, j] > 0.16:
                night[i, j] = 1
                day[i, j] = 1
            if water_mask[i, j] == 1:
                night[i, j] = 0
                day[i, j] = 1

    if dyn_parameters.daynight == 'day':
        im2 = ArrayToGeotif(day, meta)
    elif dyn_parameters.daynight == 'night':
        im2 = ArrayToGeotif(night, meta)

    tree_mask, meta = GeotifToArray(tree_mask, 1)

    crop = np.copy(night)
    tree = np.copy(night)

```

```
    for i in range(h):
        for j in range(w):
            if night[i, j] == 1:
                if tree_mask[i, j] == 1:
                    crop[i, j] = 0
                else:
                    tree[i, j] = 0

    im3 = ArrayToGeotif(crop, meta)
    im4 = ArrayToGeotif(tree, meta)

    arr = day = night = tree = crop = None
    return im1, im2, im3, im4
```

B.7. python/vegetation_footprints.py

```
import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
from numba import jit, prange
#-------------------------------------------------------------------------------
# vegetation_footprint
# purpose: vegetation footprint calculation for urban heat map
# input: vegfra
# output: vegfra_2d
#-------------------------------------------------------------------------------
#@jit(parallel=True)
def Vegetation_footprints(stat, dyn, im):

    print('Vegetation_footprints.Calculator')

    f = open('d:/tmp/veg.dat', 'wt')

    vegfra, meta = GeotifToArray(im, 1) # analyse gebied met randen

    nrow = int(stat.nrow * stat.cellsize / stat.blocksize)
    ncol = int(stat.ncol * stat.cellsize / stat.blocksize)
    metadata = [stat.xmin, stat.ymin, stat.blocksize, nrow, ncol]
    jleft, jright, iup, idown = window_footprint(dyn.winddir, dyn.upveg, dyn.sideveg, dyn.downveg, dyn.noveg, stat.blocksize)
    iref = int((stat.ymin - meta[1]) / meta[2])
    jref = int((stat.xmin - meta[0]) / meta[2])

    f.write(f'{metadata[0]} {metadata[1]} {metadata[2]} {metadata[3]} {metadata[4]}\n')
    f.write(f'{nrow} {ncol} {meta[0]} {meta[1]} {meta[2]} {meta[3]} {meta[4]}\n')
    f.write(f'{jleft} {jright} {iup} {idown} {iref} {jref}\n')
    f.close()

    vegfra_2d = np.zeros((nrow, ncol))
    for i in range(nrow):
        istart = i + iref - idown
        iend = i + iref + iup
        for j in range(ncol):
            jstart = j + jref - jleft
            jend = j + jref + jright
            vegfra_2d[i, j] = np.mean(vegfra[istart: iend+1, jstart: jend+1])

    im1 = ArrayToGeotif(vegfra_2d, metadata)
    vegfra_2d = None

    return im1
```

B.8. python/skyview_footprints.py

```
import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
#-----------------------------------------------------------------------------------------------
# skyview_footprint
# purpose: skyview footprint calculation for urban heat map
# input: skyview
# output: skyview_2d
#-----------------------------------------------------------------------------------------------
def Skyview_footprints(stat, dyn, im):

    print('SkyView.Calculator')

    svf_2d = np.array(im)
    svf, meta = GeotifToArray(im, 1)                                          # analyse gebied met randen

    nrow = int(stat.nrow * stat.cellsize / stat.blocksize)
    ncol = int(stat.ncol * stat.cellsize / stat.blocksize)
    metadata = [stat.xmin, stat.ymin, stat.blocksize, nrow, ncol]
    jleft, jright, iup, idown = window_footprint(dyn.winddir, dyn.upveg, dyn.sideveg, dyn.downveg, dyn.noveg, stat.blocksize)
    iref = int((stat.ymin - meta[1]) / meta[2])
    jref = int((stat.xmin - meta[0]) / meta[2])
    h = nrow
    w = ncol

    mean_svf = np.zeros((h, w))
    for i in range(h):
        istart = i + iref - idown
        iend = i + iref + iup
        for j in range(w):
            jstart = j + jref - jleft
            jend = j + jref + jright
            perc = (np.mean(svf[istart: iend+1, jstart: jend+1]) > 0) / (np.sum(svf[istart: iend+1, jstart: jend+1]) > -1)
            if perc >= 0.2:
                mean_svf[i, j] = np.mean(svf[istart: iend+1, jstart: jend+1])
            elif perc >= 0.1:  # linearize between svf=1 for 0.1 and svf as executed above
                mean_pre_svf = np.mean(svf[istart: iend+1, jstart: jend+1])
                mean_svf[i, j] = ((perc - 0.1) / 0.1) * mean_pre_svf + (1 - (perc - 0.1) / 0.1) * 1
            else:
                mean_svf[i, j] = 1

    im1 = ArrayToGeotif(mean_svf, metadata)
    mean_svf = None

    return im1
```

B.9. python/urban_heat.py

```
import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
import pandas as pd

#------------------
# urbanheat
# python code: urban_heat
# input: vegfra_wind, svf_wind
# output: urban_heat

def Urban_heat(stat, dyn, im1, im2):

    print('Urban_heat.Calculator')

    S = dyn.S
    U = dyn.U
    Tmin = dyn.Tmin
    Tmax = dyn.Tmax

    vegfra, meta = GeotifToArray(im1, 1)
    svf, meta = GeotifToArray(im2, 1)
    h = np.shape(vegfra)[0]  # y
    w = np.shape(vegfra)[1]  # x
    uhi = np.ones((h, w))
    uhi *= 2
    uhi = uhi - vegfra - svf
    factor = (S * (Tmax - Tmin) ** 3 / U) ** (1 / 4)
    uhi *= factor

    im3 = ArrayToGeotif(uhi, meta)
    vegfra = svf = None

    return im3
```

B.10. python/pet_calculate.py

```
#from IPython import get_ipython
#get_ipython().magic('reset -sf')

import numpy as np
from .pet_parameters import window_footprint
from .geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite
#-----------------------------------------------------------------------------------------------------------

# petcalculate
# purpose: calculate the PET
# input: shadow, urbanheat, wind, svf, svf_mask, ndvi_crop_mask, ndvi_tree_mask
# output: pets

def PET_calculate(stat, dyn, im1, im2, im3, im4, im5, im6, im7):

    TT = dyn.TT                 #TT:    Temperatuur (in 0.1 graden Celsius) op 1.50 m hoogte tijdens de waarneming
    FF = dyn.FF                 #FF:    Windsnelheid (in 0.1 m/s) gemiddeld over de laatste 10 minuten van het afgelopen uur
    Q = dyn.Q                   #Q:     Global solar irradiationGlobale straling (in J/cm2) per uurvak
    Qdif = dyn.Qdif             #Qdif:  Difuse radiation
    sunalt = dyn.sunalt         #sunalt:solar elevation angle
    RH = dyn.RH                 #RH:    Relative Humidity
    diurnal = dyn.diurnal       #diurnal correction factor UHI for Ta

    print('PET.Calculator')
    Bveg = 0.4
    Bnoveg = 3
    stef = 5.67 * 10 ** -8

    sun, meta = GeotifToArray(im1, 1)  # added anders geen ref in shadow
    urban, meta = GeotifToArray(im2, 1)
    wind, meta = GeotifToArray(im3, 1)
    svf, meta = GeotifToArray(im4, 1)
    svf_mask, meta = GeotifToArray(im5, 1)
    mask_vegfra, meta = GeotifToArray(im6, 1)
    trees_2m, meta = GeotifToArray(im7, 1)

    # with open("D:\\tmp\\test.txt", 'wt') as f:
    #     f.write(f"sun, meta {sun, meta}\\n")
    #     f.write(f"urban, meta {urban, meta}\\n")
    #     f.write(f"wind, meta {wind, meta}\\n")
    #     f.write(f"svf, meta {svf, meta}\\n")
    #     f.write(f"svf_mask, meta {svf_mask, meta}\\n")
    #     f.write(f"mask_vegfra, meta {mask_vegfra, meta}\\n")
    #     f.write(f"trees_2m, meta {trees_2m, meta}\\n")


    Ta = urban[:] * diurnal + TT
    Tw = TT * np.arctan(0.15198 * (RH + 8.3137) ** 0.5) + np.arctan(TT + RH) - np.arctan(
        RH - 1.676) + 0.0039184 * RH ** 1.5 * np.arctan(0.023101 * RH) - 4.686

    wind = ((wind - 0.125) * 0.5829 + 0.125) * FF
    wind[wind < 0.5] = 0.5
    wind_temp = np.ravel(wind)
    #wind_res = np.array(wind_temp).transpose()

```

```
    # day
    if Q > 0:
        sun_temp, meta = GeotifToArray(im1, 1)
        sun = sun_temp * (1 - trees_2m[:])

        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw
             + 0.015 * svf[:] * Qdif +
                    0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) * (1
                        - sun[:]) * svf_mask[:]
        PETveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.
                sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bveg + 1.94 * svf[:]) *
                mask_vegfra[:] * sun[:] * svf_mask[:]
        PETnoveg = (-13.26 + 1.25 * Ta[:] + 0.011 * Q - 3.37 * np.log(
            wind[:]) + 0.078 * Tw + 0.0055 * Q * np.log(wind[:]) + 5.56 * np.
                sin(
            sunalt / 360 * 2 * np.pi) - 0.0103 * Q * np.log(wind[:]) * np.sin(
            sunalt / 360 * 2 * np.pi) + 0.546 * Bnoveg + 1.94 * svf[:]) * (1 -
                mask_vegfra[:]) * sun[:] * svf_mask[:]

        PET = PETshade + PETveg + PETnoveg

    # night
    else:
        PETshade = (-12.14 + 1.25 * Ta[:] - 1.47 * np.log(wind[:]) + 0.060 * Tw
             + 0.015 * svf[:] * Qdif
                    + 0.0060 * (1 - svf[:]) * stef * (Ta[:] + 273.15) ** 4) *
                        (1 - sun[:]) * svf_mask[:]

        PET = PETshade

    im8 = ArrayToGeotif(PET, meta)
    sun = urban = wind = svf = svf_mask = mask_vegfra = trees_2m = PET = None

    return im8
```

B.11. python/pet_simulator.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 PetUi
                                 A QGIS plugin
 Physiological Equivalent Temperature Simulator
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                              -------------------
        begin                : 2023-08-02
        git sha              : $Format:%H$
        copyright            : (C) 2023 by Marieke van Esch, student TU Delft, the Netherlands
        email                : marieke.vanesch@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/

"""

from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication #Qdate
from qgis.core import QgsRasterLayer
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction
from qgis.core import QgsProject, QgsRectangle
from osgeo import gdal, osr, ogr

# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .pet_simulator_dialog import PetUiDialog
import os.path
import numpy as np
import pandas as pd
import datetime
import time
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib.image as mpimg

from .algorithm.pet_parameters import StatParameters, writer
from .algorithm.pet_parameters import DynParameters
from .algorithm.pet_parameters import window_footprint, wind_direction
from .algorithm.geotiff_creator import ArrayToGeotif, GeotifToArray, GeotifWrite, ArrayWrite, ArrayWriteG

class PetUi:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
```

```
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'PetUi_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&PET Simulator')

        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None

        self.weather = DynParameters(Date=20150701, decade=1, hour=12, min=0, TT=28, FF=6, dd=100, Q=794.444, Qdif=158.88,
                 sunalt=55.3, RH=48, diurnal=0.03, Tmin= 24, Tmax = 34, U = 6)

        self.spatial = StatParameters(xmin=172075, xmax=172075 + 6, ymin=440675, ymax=440675 + 5, cellsize=1,
                                 station='herwijnen', station_lat=51.859, station_lon=5.146)

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('PetUi', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
```

```
        status_tip=None,
        whats_this=None,
        parent=None):

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            # Adds plugin icon to Plugins toolbar
            self.iface.addToolBarIcon(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/pet_simulator/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'PETS'),
            callback=self.run,
            parent=self.iface.mainWindow())

        # will be set False in run()
        self.first_start = True

    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&PET Simulator'),
                action)
            self.iface.removeToolBarIcon(action)

    def clipping(self):

        self.exportdata() # read data from line edits

        root = QgsProject.instance().layerTreeRoot()
        for i in range(11):
            if i == 0:
                name = 'ahn'
            elif i == 1:
                name = 'building_height'
            elif i == 2:
                name = 'building_mask'
```

```
            elif i == 3:
                name = 'ndvi_infr'
            elif i == 4:
                name = 'ndvi_rgb'
            elif i == 5:
                name = 'Shadow_20150701_0900_LST' #Shadow_20150701_1000_LST #Shadow_20150701_1200_LST
            elif i == 6:
                name = 'svf'
            elif i == 7:
                name = 'svf_mask'
            elif i == 8:
                name = 'tree_height'
            elif i == 9:
                name = 'tree_mask'
            elif i == 10:
                name = 'water_mask'

            intiff = gdal.Open(f'{self.spatial.directory_in}{name}.tif')  # input from file

            up = max(self.weather.upwind, self.weather.upveg)
            side = max(self.weather.sidewind, self.weather.sideveg)
            down = max(self.weather.downwind, self.weather.downveg)
            now = max(self.weather.nowind, self.weather.noveg)
            ileft, iright, iup, idown = window_footprint(self.weather.winddir, up, side, down, now, self.spatial.cellsize)
            xleft = ileft * self.spatial.cellsize
            xright = iright * self.spatial.cellsize
            yup = iup * self.spatial.cellsize
            ydown = idown * self.spatial.cellsize

            # clip to maximal extended window
            outputfile = f'{self.spatial.directory_out}input\\{self.spatial.label}_{name}.tif'
            bounds = (self.spatial.xmin-xleft, self.spatial.ymin-ydown, self.spatial.xmax+xright, self.spatial.ymax+yup)
            gdal.Warp(outputfile, intiff, outputBounds=bounds)  # output to file

            self.TifToJPG(self.spatial.directory_out, 'input', f'{self.spatial.label}_{name}', large=True)

            if self.dlg.checkBox.checkState():
                ArrayWriteG(f'{self.testin}', f'{self.spatial.label}_{name}', f'{outputfile}')

            intiff = None
            raster_layer = QgsRasterLayer(outputfile, f'{name}', 'gdal')  # input from file
            if not raster_layer.isValid():
                print('Error: Invalid raster layer.')
            else:
                QgsProject.instance().addMapLayer(raster_layer)
            #layer = QgsProject.instance().mapLayersByName(f'{name}')[0]
            #myLayerNode = root.findLayer(layer.id())
            #myLayerNode.setExpanded(False)
            #myLayerNode.setItemVisibilityChecked(False)
```

```
    def addGttiffLayer(self, directory, name, im, driver, root):

        outputfile = f'{directory}{self.spatial.label}_{name}.tif'
        driver.CreateCopy(outputfile, im, strict=0)
        raster_layer = QgsRasterLayer(outputfile, f'{name}', 'gdal')  # input from file
        if not raster_layer.isValid():
            print('Error: Invalid raster layer.')
        else:
            QgsProject.instance().addMapLayer(raster_layer)
        layer = QgsProject.instance().mapLayersByName(f'{name}')[0]
        myLayerNode = root.findLayer(layer.id())
        myLayerNode.setExpanded(False)
        myLayerNode.setItemVisibilityChecked(False)

    def clipper(self, basedirectory, subdirectory, filename):

        intiff = gdal.Open(f'{basedirectory}{subdirectory}\\{filename}')
        outputfile = f'{basedirectory}clip\\{filename}'
        bounds = (self.spatial.xmin, self.spatial.ymin, self.spatial.xmax, self.spatial.ymax) #small
        gdal.Warp(outputfile, intiff, outputBounds=bounds)
        outtiff = gdal.Open(outputfile)
        return outtiff

    def TifToJPG(self, basedirectory, subdirectory, filename, binary=False, ticks= not None, large=False):
        tif = gdal.Open(f'{basedirectory}{subdirectory}\\{filename}.tif')
        tifArray = tif.ReadAsArray()
        data, metadata = GeotifToArray(tif, 1)

        #[xmin, ymin, cellsize, nrow, ncol]

        extent = metadata[0], metadata[0] + metadata[4] * metadata[2], metadata[1], metadata[1] + metadata[3] *metadata[2]
        if binary is True:
            plt.matshow(tifArray, cmap='gray', extent=extent)
            colorarr = np.linspace(np.min(tifArray), np.max(tifArray), 11)
            plt.colorbar(ticks=colorarr)
        else:
            plt.matshow(data, cmap='rainbow', extent=extent)
            colorarr = np.linspace(np.min(tifArray), np.max(tifArray), 11)
            plt.colorbar(ticks=colorarr, shrink=0.8)

        plt.title(filename)
        plt.xlabel('x') #lon
        plt.ylabel('y') #lat
        plt.axis('equal')
        plt.gca().xaxis.tick_bottom()
        plt.ticklabel_format(useOffset=False)

        if large:
            plt.savefig(f'{basedirectory}tif\\{filename}_large.jpg', bbox_inches='tight')
        else:
            plt.savefig(f'{basedirectory}tif\\{filename}.jpg', bbox_inches='tight')
        #plt.show()

    def timecalculator(self, timers, name, flag):

```

```
        elapsed_time_flag1 = flag[1] - flag[0]
        elapsed_time_flag2 = flag[2] - flag[1]
        elapsed_time_flag3 = flag[3] - flag[2]
        elapsed_time = elapsed_time_flag1 + elapsed_time_flag2 +
            elapsed_time_flag3
        timers[f'Elapsed time {name} (s)'] = elapsed_time
        timers[f'--- flag1 {name} read (s)'] = elapsed_time_flag1
        timers[f'--- flag2 {name} calculate (s)'] = elapsed_time_flag2
        timers[f'--- flag3 {name} write (s)'] = elapsed_time_flag3

    def timewriter(self, filename, timers):

        with open(filename, 'w') as f:
            # sum = timers.items()[1].sum()
            sum1 = sum(timers.values())/2
            for key, value in timers.items():
                # sum += value
                f.write(f'{key:35} :  {value:6.3f} : {((value / sum1) * 100)
                    :6.2f} % \n')
            f.write(f'Total time (s): {sum1:.6f}')

    def toTif(self, basedirectory):

        for i in range(11):
            if i == 0:
                name = 'ahn'
            elif i == 1:
                name = 'building_height'
            elif i == 2:
                name = 'building_mask'
            elif i == 3:
                name = 'ndvi_infr'
            elif i == 4:
                name = 'ndvi_rgb'
            elif i == 5:
                name = 'Shadow_20150701_0900_LST'
            elif i == 6:
                name = 'svf'
            elif i == 7:
                name = 'svf_mask'
            elif i == 8:
                name = 'tree_height'
            elif i == 9:
                name = 'tree_mask'
            elif i == 10:
                name = 'water_mask'
            name = f'{self.spatial.label}_{name}.tif'
            image = gdal.Open(f'{self.spatial.directory_out}clip\\{name}')
            data, metadata = GeotifToArray(image, 1)

            # only for testing
            #ArrayWrite(f'{self.spatial.directory_out}tif\\{name}', data,
                metadata)

    def calculate(self):

        self.exportdata()

        root = QgsProject.instance().layerTreeRoot()
        driver = gdal.GetDriverByName('GTiff')

```

```
        # ------------------------------------------------------------------------

        timers = dict()

        from .algorithm.fraction_area_buildings_treeregr import FaBuildingTree
        flag = []
        # import geotiffs
        flag.append(time.perf_counter())
        im1 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_building_height.tif') # large
        im2 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_building_mask.tif') # large
        im3 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_tree_height.tif') # large
        im4 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_tree_mask.tif') # large

        #calculate
        flag.append(time.perf_counter())
        #if not os.path.isfile(f'{self.spatial.directory_out}output\\wind.tiff'):
        im5 = FaBuildingTree(self.spatial, self.weather, im1, im2, im3, im4) # large

        # upscale coarse to fine
        name = 'wind_coarse'
        self.addGttiffLayer(f'{self.spatial.directory_out}scale\\', name, im5, driver, root) # test
        scaled = f'{self.spatial.directory_out}output\\{self.spatial.label}_wind.tif'  # large
        #gdal.Warp(scaled, im5, xRes=self.spatial.cellsize, yRes=self.spatial.cellsize, outputType=gdal.GDT_Float32, resampleAlg="average")
        #im5 = gdal.Open(scaled)

        # downscale coarse to fine
        data_type = gdal.GDT_Float32
        driver = gdal.GetDriverByName('GTiff')
        in_band = im5.GetRasterBand(1)
        out_ds = driver.Create(scaled, self.spatial.ncol, self.spatial.nrow, bands=1, eType=data_type)
        out_ds.SetProjection(im5.GetProjection())
        geotransform = list(im5.GetGeoTransform())
        geotransform[1] /= self.spatial.blocksize / self.spatial.cellsize
        geotransform[5] /= self.spatial.blocksize / self.spatial.cellsize
        out_ds.SetGeoTransform(geotransform)
        data = in_band.ReadAsArray(buf_xsize=self.spatial.ncol,buf_ysize=self.spatial.nrow)
        out_band = out_ds.GetRasterBand(1)
        out_band.WriteArray(data)
        im5 = out_ds

        #add layer and geotifs
        name = 'wind'
        flag.append(time.perf_counter())
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im5, driver, root)
        im1 = im2 = im3 = im4 = im5 = None
        self.dlg.label_18.setText('checked')

```

```
        #self.dlg.show()
        self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_building_height.tif')
        self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_building_mask.tif')
        self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_tree_height.tif')
        self.clipper(self.spatial.directory_out, 'input', f'{self.spatial.label}_tree_mask.tif')

        self.TifToJPG(self.spatial.directory_out, 'clip', f'{self.spatial.label}_building_height')
        self.TifToJPG(self.spatial.directory_out, 'clip', f'{self.spatial.label}_building_mask', binary=True)
        self.TifToJPG(self.spatial.directory_out, 'clip', f'{self.spatial.label}_tree_height')
        self.TifToJPG(self.spatial.directory_out, 'clip', f'{self.spatial.label}_tree_mask', binary=True)
        self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.label}_wind')
        flag.append(time.perf_counter())

        # array write (only with testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_{name}', f'{self.spatial.directory_out}output\\{self.spatial.label}_wind.tif')

        self.timecalculator(timers, name, flag)

        # ---------------------------------------------------------------------------------------------------

        from .algorithm.ndvi_infr_large import Ndvi_infr_large
        flag = []

        #import geotiffs
        flag.append(time.perf_counter())
        im1 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_ndvi_rgb.tif') # large
        im2 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_ndvi_infr.tif') # large
        im3 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_water_mask.tif') # large
        im4 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.label}_tree_mask.tif') # large

        # calculate
        flag.append(time.perf_counter())
        im5, im6, im7, im8 = Ndvi_infr_large(self.spatial, self.weather, im1, im2, im3, im4)  # large

        # add tif and layer
        flag.append(time.perf_counter())
        name = 'ndvi'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im5, driver, root)
        name = 'vegfra'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im6, driver, root)
```

```
        name = ’ndvi_crop_mask’
        self.addGttiffLayer(f’{self.spatial.directory_out}output\\’, name, im7,
             driver, root)
        name = ’ndvi_tree_mask’
        self.addGttiffLayer(f’{self.spatial.directory_out}output\\’, name, im8,
             driver, root)
        im1 = im2 = im3 = im4 = im5 = im6 = im7 = im8 = None
        self.dlg.label_13.setText(’checked’)

        self.clipper(self.spatial.directory_out, ’input’, f’{self.spatial.label
            }_ndvi_rgb.tif’)
        self.clipper(self.spatial.directory_out, ’input’, f’{self.spatial.label
            }_ndvi_infr.tif’)
        self.clipper(self.spatial.directory_out, ’input’, f’{self.spatial.label
            }_water_mask.tif’)
        self.clipper(self.spatial.directory_out, ’output’, f’{self.spatial.
            label}_ndvi.tif’)
        self.clipper(self.spatial.directory_out, ’output’, f’{self.spatial.
            label}_vegfra.tif’)
        self.clipper(self.spatial.directory_out, ’output’, f’{self.spatial.
            label}_ndvi_crop_mask.tif’)
        self.clipper(self.spatial.directory_out, ’output’, f’{self.spatial.
            label}_ndvi_tree_mask.tif’)

        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_ndvi_rgb’)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_ndvi_infr’)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_water_mask’, binary=True)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_ndvi’)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_vegfra’)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_ndvi_crop_mask’)
        self.TifToJPG(self.spatial.directory_out, ’clip’, f’{self.spatial.label
            }_ndvi_tree_mask’)
        flag.append(time.perf_counter())

        # write array (only for testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f’{self.testout}’, f’{self.spatial.label}_ndvi’,
                        f’{self.spatial.directory_out}output\\{self.spatial.
                            label}_ndvi.tif’)
            ArrayWriteG(f’{self.testout}’, f’{self.spatial.label}_vegfra’,
                        f’{self.spatial.directory_out}output\\{self.spatial.
                            label}_vegfra.tif’)
            ArrayWriteG(f’{self.testout}’, f’{self.spatial.label}
                _ndvi_crop_mask.tif’,
                        f’{self.spatial.directory_out}output\\{self.spatial.
                            label}_ndvi_crop_mask.tif’)
            ArrayWriteG(f’{self.testout}’, f’{self.spatial.label}
                _ndvi_tree_mask’,
                        f’{self.spatial.directory_out}output\\{self.spatial.
                            label}_ndvi_tree_mask.tif’)

        self.timecalculator(timers, name, flag)

        #
            ---------------------------------------------------------------------------
```

```

        from .algorithm.vegetation_footprints import Vegetation_footprints
        flag = []
        #import geotiffs
        flag.append(time.perf_counter())
        im1 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.
            label}_vegfra.tif') # large

        # upscale to blocksize fine to coarse
        scaled = f'{self.spatial.directory_out}scale\\{self.spatial.label}
            _vegfra.tif'
        gdal.Warp(scaled, im1, xRes=self.spatial.blocksize, yRes=self.spatial.
            blocksize, resampleAlg="average")
        im1 = gdal.Open(scaled)

        #calculate
        flag.append(time.perf_counter())
        im2 = Vegetation_footprints(self.spatial, self.weather, im1) # small

        #downscale coarse to fine
        data_type = gdal.GDT_Float32
        driver = gdal.GetDriverByName('GTiff')
        in_band = im2.GetRasterBand(1)
        out_ds = driver.Create(scaled, self.spatial.ncol, self.spatial.nrow,
            bands=1, eType=data_type)
        out_ds.SetProjection(im2.GetProjection())
        geotransform = list(im2.GetGeoTransform())
        geotransform[1] /= self.spatial.blocksize / self.spatial.cellsize
        geotransform[5] /= self.spatial.blocksize / self.spatial.cellsize
        out_ds.SetGeoTransform(geotransform)
        data = in_band.ReadAsArray(buf_xsize=self.spatial.ncol, buf_ysize=self.
            spatial.nrow)
        out_band = out_ds.GetRasterBand(1)
        out_band.WriteArray(data)
        im2 = out_ds

        #add layer and geotiffs
        flag.append(time.perf_counter())
        name = 'vegfra_wind'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im2,
             driver, root)
        im1 = im2 = None
        self.dlg.label_14.setText('checked')
        self.clipper(self.spatial.directory_out, 'output', f'{self.spatial.
            label}_vegfra.tif')
        self.TifToJPG(self.spatial.directory_out, 'clip', f'{self.spatial.label
            }_vegfra')
        self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.
            label}_vegfra_wind')
        flag.append(time.perf_counter())

        # write array (only for testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_vegfra_wind',
                        f'{self.spatial.directory_out}output\\{self.spatial.label}
                            _vegfra_wind.tif')

        self.timecalculator(timers, name, flag)

```

```
        #
            ----------------------------------------------------------------------------

        from .algorithm.skyview_footprints import Skyview_footprints
        flag = []

        #import geotif
        flag.append(time.perf_counter())
        im1 = gdal.Open(f'{self.spatial.directory_out}input\\{self.spatial.
            label}_svf.tif') # large

        # scale to blocksize
        scaled = f'{self.spatial.directory_out}scale\\{self.spatial.label}_svf.
            tif'
        gdal.Warp(scaled, im1, xRes=self.spatial.blocksize, yRes=self.spatial.
            blocksize, resampleAlg="average")
        im1 = gdal.Open(scaled)

        # calculate
        flag.append(time.perf_counter())
        im2 = Skyview_footprints(self.spatial, self.weather, im1) # small

        # downscale coarse to fine
        data_type = gdal.GDT_Float32
        driver = gdal.GetDriverByName('GTiff')
        in_band = im2.GetRasterBand(1)
        out_ds = driver.Create(scaled, self.spatial.ncol, self.spatial.nrow,
            bands=1, eType=data_type)
        out_ds.SetProjection(im2.GetProjection())
        geotransform = list(im2.GetGeoTransform())
        geotransform[1] /= self.spatial.blocksize / self.spatial.cellsize
        geotransform[5] /= self.spatial.blocksize / self.spatial.cellsize
        out_ds.SetGeoTransform(geotransform)
        data = in_band.ReadAsArray(buf_xsize=self.spatial.ncol, buf_ysize=self.
            spatial.nrow)
        out_band = out_ds.GetRasterBand(1)
        out_band.WriteArray(data)
        im2 = out_ds

        #add layer and write geotiffs
        flag.append(time.perf_counter())
        name = 'svf_wind'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im2,
             driver, root)
        im1 = im2 = None
        self.dlg.label_15.setText('checked')
        self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.
            label}_svf_wind')
        flag.append(time.perf_counter())

        #write array (only  for testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_svf_wind',
                    f'{self.spatial.directory_out}output\\{self.spatial.label}
                        _svf_wind.tif')

        self.timecalculator(timers, name, flag)

        #
            ----------------------------------------------------------------------------
```

```
        from .algorithm.urban_heat import Urban_heat
        flag = []

        # import geotiff
        flag.append(time.perf_counter())
        im1 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.
            label}_vegfra_wind.tif') # small
        im2 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.
            label}_svf_wind.tif') # small
        self.dlg.label_16.setText('imported')
        #self.dlg.show()  refresh ??

        # calculate
        flag.append(time.perf_counter())
        im3 = Urban_heat(self.spatial, self.weather, im1, im2)
        end_time_flag2 = time.perf_counter()

        # add layer and write geotiffs
        flag.append(time.perf_counter())
        name = 'urban_heat'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im3,
             driver, root)
        im1 = im2 = im3 = None
        self.dlg.label_16.setText('checked')
        self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.
            label}_urban_heat')
        flag.append(time.perf_counter())

        # write array (only for testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_urban_heat',
                    f'{self.spatial.directory_out}output\\{self.spatial.label}
                        _urban_heat.tif')

        self.timecalculator(timers, name, flag)

        #
            ------------------------------------------------------------------------------------------------------------


        from .algorithm.pet_calculate import PET_calculate
        flag = []

        # import geotiff
        flag.append(time.perf_counter())
        name = f'Shadow_{self.weather.year}{self.weather.month:02d}{self.
            weather.day:02d}_{self.weather.hour:02d}{self.weather.min:02d}_LST'
        name = "Shadow_20150701_0900_LST"

        im1 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial
            .label}_{name}.tif') # small
        im2 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.
            label}_urban_heat.tif') # small
        im3 = gdal.Open(f'{self.spatial.directory_out}output\\{self.spatial.
            label}_wind.tif') # small
        im4 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial
            .label}_svf.tif')  # small
        im5 = self.clipper(self.spatial.directory_out, 'input', f'{self.spatial
            .label}_svf_mask.tif')  # small
```

```
        im6 = self.clipper(self.spatial.directory_out, 'output', f'{self.spatial.label}_ndvi_crop_mask.tif')  # small
        im7 = self.clipper(self.spatial.directory_out, 'output', f'{self.spatial.label}_ndvi_tree_mask.tif')  # small


        # calculate
        flag.append(time.perf_counter())
        im8 = PET_calculate(self.spatial, self.weather, im1, im2, im3, im4, im5, im6, im7) # small #nonetype

        # add layer and write geotiffs
        flag.append(time.perf_counter())
        name = 'pets'
        self.addGttiffLayer(f'{self.spatial.directory_out}output\\', name, im8, driver, root)
        im1 = im2 = im3 = im4 = im5 = im6 = im7 = None
        self.dlg.label_17.setText('checked')
        flag.append(time.perf_counter())
        self.TifToJPG(self.spatial.directory_out, 'output', f'{self.spatial.label}_pets')
        flag.append(time.perf_counter())

        # write array (only for testing)
        if self.dlg.checkBox.checkState():
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_svf.tif',
                        f'{self.spatial.directory_out}clip\\{self.spatial.label}_svf.tif')
            ArrayWriteG(f'{self.testout}', f'{self.spatial.label}_pets',
                        f'{self.spatial.directory_out}output\\{self.spatial.label}_pets.tif')

        self.timecalculator(timers, name, flag)
        self.timewriter(f'{self.spatial.directory_out}timewriterv1.txt', timers)


        # -----------------------------------------------------------------------------
    def importdata(self):

        self.spatial.directory_in = self.dlg.lineEdit_3.text()
        self.spatial.directory_out = self.dlg.lineEdit_2.text()
        self.spatial.label = self.dlg.lineEdit_1.text()

        with open(f'{self.spatial.directory_out}set.csv', 'r') as fp:
            lines = fp.readlines()
            lines = [line.strip() for line in lines]
            lines = [line.split(',') for line in lines]
            self.spatial.station = lines[3][1]
            self.spatial.ymax = float(lines[4][1])
            self.spatial.xmax = float(lines[5][1])
            self.spatial.ymin = float(lines[6][1])
            self.spatial.xmin = float(lines[7][1])
            self.spatial.cellsize = float(lines[8][1])
            self.spatial.blocksize = float(lines[9][1])
            self.spatial.Resize()
            self.weather.TT = float(lines[10][1])
            self.weather.FF = float(lines[11][1])
            self.weather.dd = float(lines[12][1])
```

```
            self.weather.wind, self.weather.WE, self.weather.winddir = wind_direction(self.weather.dd, self.weather.FF)
            self.weather.Q = float(lines[13][1])
            self.weather.Qdif = float(lines[14][1])
            self.weather.sunalt = float(lines[15][1])
            self.weather.RH = float(lines[16][1])
            self.weather.diurnal = float(lines[21][1])

        self.dlg.lineEdit_7.setText(f'{self.spatial.ymax}')  # north
        self.dlg.lineEdit_6.setText(f'{self.spatial.xmax}')  # east
        self.dlg.lineEdit_5.setText(f'{self.spatial.ymin}')  # south
        self.dlg.lineEdit_4.setText(f'{self.spatial.xmin}')  # west
        self.dlg.lineEdit_17.setText(f'{self.spatial.cellsize}')  # south
        self.dlg.lineEdit_16.setText(f'{self.spatial.blocksize}')  # west
        self.dlg.lineEdit_3.setText(f'{self.spatial.directory_in}')
        self.dlg.lineEdit_2.setText(f'{self.spatial.directory_out}')
        self.dlg.lineEdit_1.setText(f'{self.spatial.label}')
        self.dlg.lineEdit_15.setText(f'{self.spatial.station}')
        self.dlg.lineEdit_8.setText(f'{self.weather.TT}')
        self.dlg.lineEdit_9.setText(f'{self.weather.FF}')
        self.dlg.lineEdit_10.setText(f'{self.weather.dd}')
        self.dlg.lineEdit_12.setText(f'{self.weather.Q}')
        self.dlg.lineEdit_13.setText(f'{self.weather.Qdif}')
        self.dlg.lineEdit_14.setText(f'{self.weather.sunalt}')
        self.dlg.lineEdit_11.setText(f'{self.weather.RH}')

        """
        f = open('D:\\tmp\\aba.txt', 'wt')
        df_KNMI = pd.read_csv(f'{self.spatial.directory_in}\\knmi_results.csv')

        yyyymmdd = f'{self.dlg.dateTimeEdit1.date()}'
        f.write(f' yyyymmdd {type(yyyymmdd)} {yyyymmdd}\n')
        hhmmss = f'{self.dlg.dateTimeEdit1.time()}'
        f.write(f' hhmmss {type(hhmmss)} {hhmmss}\n')
        station = self.dlg.lineEdit_15.setText(f'{self.spatial.station}')
        original_format = "%YYYY-%mm-%dd"
        parsed_date = datetime.strptime(yyyymmdd, original_format)
        desired_format = "%dd/%mm/%YYYY"


        parsed_time = datetime.strptime(hhmmss, "%H:%M:%S")
        hour = parsed_time.hour

        formatted_date = parsed_date.strftime(desired_format)


        date_string = self.dlg.dateTimeEdit1.date()
        parsed_date = eval(date_string)  # Evaluate the string to create a QDate object

        year = parsed_date.year()
        month = parsed_date.month()
        day = parsed_date.day()



        f.write(f' yearmonthday {year} {month} {day}\n')
        """


```

```
    #
        ----------------------------------------------------------------------------
    def importknmi(self):

        # knmi file -> self.weather
        df_KNMI = pd.read_csv(f'{self.spatial.directory_in}\\knmi_results.csv')
        yyyymmdd = f'{self.dlg.dateTimeEdit1.date()}'
        hhmmss = f'{self.dlg.dateTimeEdit1.time()}'
        station = self.dlg.lineEdit_15.setText(f'{self.spatial.station}')


        f = open('D:\\tmp\\aba.txt', 'wt')
        f.write(f'{type(yyyymmdd)} {yyyymmdd}\n')

        original_format = "%Y-%m-%d"
        parsed_date = datetime.strptime(yyyymmdd, original_format)
        desired_format = "%d/%m/%Y"

        parsed_time = datetime.strptime(hhmmss, "%H:%M:%S")
        hour = parsed_time.hour

        formatted_date = parsed_date.strftime(desired_format)


        condition = (df_KNMI['YYYYMMDD'] == formatted_date) & (df_KNMI['H'] ==
            hour) & (df_KNMI['station'] == station)
        filtered_rows = df_KNMI[condition]
        """
        self.dlg.lineEdit_8.setText(f'{self.weather.TT}')
        self.dlg.lineEdit_9.setText(f'{self.weather.FF}')
        self.dlg.lineEdit_10.setText(f'{self.weather.dd}')
        self.dlg.lineEdit_12.setText(f'{self.weather.Q}')
        self.dlg.lineEdit_13.setText(f'{self.weather.Qdif}')
        self.dlg.lineEdit_14.setText(f'{self.weather.sunalt}')
        self.dlg.lineEdit_11.setText(f'{self.weather.RH}')
        """
        self.dlg.lineEdit_8.setText(filtered_rows['TT'])
        self.dlg.lineEdit_8.setText(filtered_rows['FF'])
        self.dlg.lineEdit_8.setText(filtered_rows['dd'])
        self.dlg.lineEdit_8.setText(filtered_rows['Q'])
        self.dlg.lineEdit_8.setText(filtered_rows['Qdif'])
        self.dlg.lineEdit_8.setText(filtered_rows['sunalt'])
        self.dlg.lineEdit_8.setText(filtered_rows['RH'])

    def exportdata(self):

        self.spatial.ymax = float(self.dlg.lineEdit_7.text())
        self.spatial.xmax = float(self.dlg.lineEdit_6.text())  # east
        self.spatial.ymin = float(self.dlg.lineEdit_5.text())  # south
        self.spatial.xmin = float(self.dlg.lineEdit_4.text())  # west
        self.spatial.cellsize = float(self.dlg.lineEdit_17.text())  # south
        self.spatial.blocksize = float(self.dlg.lineEdit_16.text())  # west
        self.spatial.directory_in = self.dlg.lineEdit_3.text()
        self.spatial.directory_out = self.dlg.lineEdit_2.text()
        self.spatial.label = self.dlg.lineEdit_1.text()
        self.spatial.station = self.dlg.lineEdit_15.text()
        self.spatial.Resize()

        #self.weather = DynParameters()
        self.weather.TT = float(self.dlg.lineEdit_8.text())
```

```
        self.weather.FF = float(self.dlg.lineEdit_9.text())
        self.weather.dd = float(self.dlg.lineEdit_10.text())
        self.weather.wind, self.weather.WE, self.weather.winddir =
            wind_direction(self.weather.dd, self.weather.FF)
        self.weather.Q = float(self.dlg.lineEdit_12.text())
        self.weather.Qdif = float(self.dlg.lineEdit_13.text())
        self.weather.sunalt= float(self.dlg.lineEdit_14.text())
        self.weather.RH = float(self.dlg.lineEdit_11.text())

        with open(f'{self.spatial.directory_out}set.csv', 'wt') as f:

            f.write(f'directory_in,{self.spatial.directory_in}\n')
            f.write(f'directory_out,{self.spatial.directory_out}\n')
            f.write(f'label,{self.spatial.label}\n')
            f.write(f'station,{self.spatial.station}\n')
            f.write(f'ymax,{self.spatial.ymax:2.2f}\n')
            f.write(f'xmax,{self.spatial.xmax:2.2f}\n')
            f.write(f'ymin,{self.spatial.ymin:2.2f}\n')
            f.write(f'xmin,{self.spatial.xmin:2.2f}\n')
            f.write(f'cellsize,{self.spatial.cellsize:2.0f}\n')
            f.write(f'blocksize,{self.spatial.blocksize:2.0f}\n')
            f.write(f'TT,{self.weather.TT:2.2f}\n')
            f.write(f'FF,{self.weather.FF:2.2f}\n')
            f.write(f'dd,{self.weather.dd:2.2f}\n')
            f.write(f'Q,{self.weather.Q:2.2f}\n')
            f.write(f'Qdif,{self.weather.Qdif:2.2f}\n')
            f.write(f'sunalt,{self.weather.sunalt:2.2f}\n')
            f.write(f'RH,{self.weather.RH:2.2f}\n')
            f.write(f'wind,{self.weather.wind}\n')
            f.write(f'WE,{self.weather.WE}\n')
            f.write(f'winddir,{self.weather.winddir}\n')
            f.write(f'daynight,{self.weather.daynight}\n')
            f.write(f'diurnal,{self.weather.diurnal}\n')
            f.write(f'Tmin,{self.weather.Tmin}\n')
            f.write(f'Tmax,{self.weather.Tmax}\n')
            f.write(f'U,{self.weather.U}\n')
            # f.write(f'upwind,{self.weather.upwind}\n')
            # f.write(f'sidewind,{self.weather.sidewind}\n')
            # f.write(f'downwind,{self.weather.downwind}\n')
            # f.write(f'nowind,{self.weather.nowind}\n')
            # f.write(f'upveg,{self.weather.upveg}\n')
            # f.write(f'sideveg,{self.weather.sideveg}\n')
            # f.write(f'downveg,{self.weather.downveg}\n')
            # f.write(f'noveg,{self.weather.noveg}\n')

    def weatherknmi(self):
        self.importknmi()
        self.exportdata()

    def run(self):
        """Run method that performs all the real work"""

        # Create the dialog with elements (after translation) and keep
            reference
        # Only create GUI ONCE in callback, so that it will only load when the
            plugin is started
        if self.first_start == True:
            self.first_start = False
            self.dlg = PetUiDialog()

        self.dlg.lineEdit_3.setText(f'{self.spatial.directory_in}')
```

```

        self.testin = f'{self.spatial.directory_out}in.txt'
        f = open(self.testin, 'wt')
        f.close()
        self.testout = f'{self.spatial.directory_out}out.txt'
        f = open(self.testout, 'wt')
        f.close()

        self.dlg.label_18.setText('')
        self.dlg.label_13.setText('')
        self.dlg.label_14.setText('')
        self.dlg.label_15.setText('')
        self.dlg.label_16.setText('')
        self.dlg.label_17.setText('')

        # show the dialog
        self.dlg.show()

        self.dlg.pushButton1.clicked.connect(self.importdata)
        self.dlg.pushButton2.clicked.connect(self.clipping)
        #self.dlg.pushButton4.clicked.connect(self.weatherknmi)
        self.dlg.pushButton3.clicked.connect(self.calculate)


        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            a=1
```

C

Users manual

User Manual: Installation Requirements

The software required to run the PET simulator includes QGIS, Python, and the UMEP QGIS plugin. Additionally, Excel and Notepad are useful if the option to write text files from the generated in-between files and output files is checked.

1. Installation of QGIS on Windows
   (a) Visit the QGIS website and go to the download page. Preferably, choose the OSGEO4W Network Installer (64-bit) and start the installation.
   (b) To install the latest version (3.x), begin the installation and choose Express Desktop Install. Note that the plugin works on QGIS 3.30.1. Visit `www.qgis.org` for installation instructions on other operating systems.
2. Install the UMEP plugin
   (a) Go to: `Plugins -> Manage and Install Plugins...` in QGIS Desktop.
   (b) Under the All tab, search for UMEP. Click on UMEP and then click Install Plugin. We recommend clicking OK to the popup question below to avoid troubles later on.
3. Adding missing Python libraries and other OSGEO functionalities
   (a) Operating system and installation instructions
      i. Windows: As Windows does not include a Python installation, QGIS makes use of a separate Python installation added during QGIS installation on your PC. There are two options available:
         A. (Try this first) Run the `osgeo4w-setup-x86_64.exe` (or `osgeo4w-setup-x86_64.exe` depending on your system) executable. This can be found using the Windows search bar. Select `Advanced Install -> Install from Internet`. When prompted to select packages, search for the required package (e.g., pandas) and click on Skip until you see a version number of pandas. Finish the installation.
         B. Alternatively, use `pip` in the OSGeo4W shell provided with QGIS to install desired Python libraries.

      For other operating systems such as MAC OS X, Linux, or other platforms, refer to the UMEP documentation: `https://umep-docs.readthedocs.io/en/latest/Getting_Started.html`.
4. Installation of PyCharm
   (a) Download PyCharm
      - Go to the official PyCharm website: `https://www.jetbrains.com/pycharm/download/`

- Choose the edition (Community or Professional) and click on the corresponding download button for Windows.
- Once downloaded, locate the installer file (`.exe`) on your computer.

(b) Run the installer

- Double-click on the installer file to start the installation process. Windows may prompt you to allow changes to your system.
- Follow the setup wizard prompts to configure the installation, choosing installation location and additional components as needed.

(c) Complete the installation

- After configuring installation options, click "Install" to start the process. The installer will copy necessary files and configure PyCharm.

(d) Launch PyCharm

- Once installed, launch PyCharm either from the Start menu or desktop shortcut.

(e) Activate PyCharm (Professional Edition)

- If using the Professional Edition, activate it using a license key or JetBrains account credentials.

(f) Set up Python interpreter

- Upon first launch, configure a Python interpreter. Choose an existing installation or install Python from within PyCharm if needed.

(g) Start using PyCharm

- Explore PyCharm features and tools for Python development.

5. Downloading PET simulator from GitHub

- The PET simulator directory should be added to the file location of plugins in the directory of QGIS.
- Example location: `C:\Users\marie\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins`

Listing C.1: Tifs necessary for retrieving SVF files from knmi api

```
    files = [
    "37EZ2.tif",
    "37FZ1.tif",
    "37FZ2.tif",
    "37GN2.tif",
    "37HN1.tif",
    "37HN2.tif",
    ]
```

Figure **??**

Downloading the plugin

The open link to the PET simulator plugin is `https://github.com/mariekeve/pet_plugin`. Here you can find the pet_simulator directory which need to be placed in the directory of python plugins in QGIS. An example is `C:\Users\marie\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\pet_simulator`. Next to this you can find an example file of run10 which showcases the run directory. An example where to put the directory in order to let it run should be in `D: \project` see Figure C.1. If you are in the directory of a run for example there is a data directory containing the base maps see Figure C.1 . In each hour simulation Figure C.3 showcases the directories created like clip, input, output, scale and tif. Also a txt file is created for the computation time and a set.csv is there for the climate and static parameters.

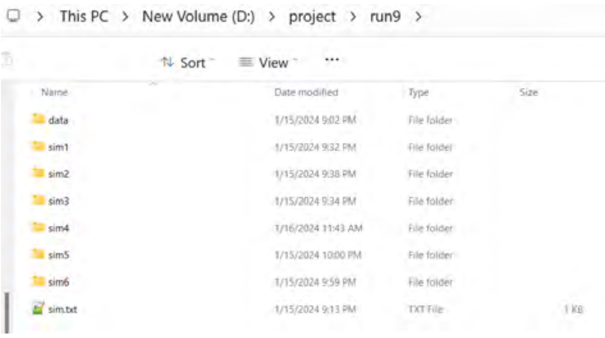


Figure C.1: Simulation overview of hours and base map data

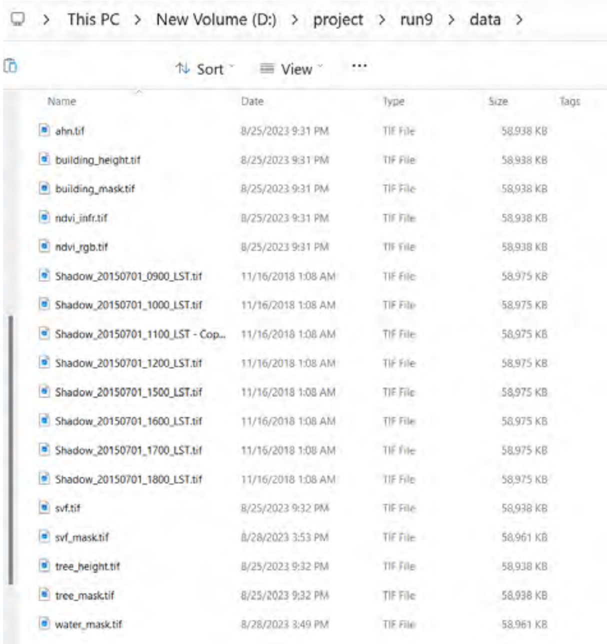


Figure C.2: Directory base maps in the map data

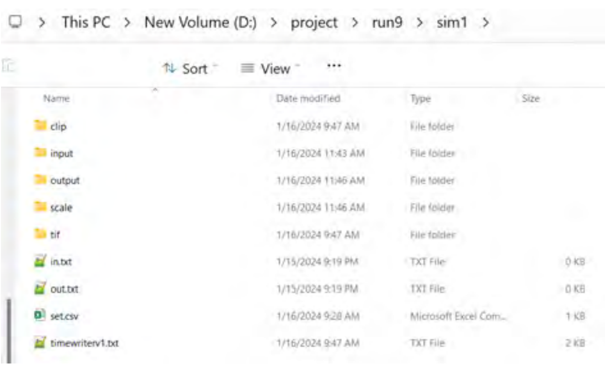


Figure C.3: Hour simulation directory of the run Rotterdam

PET simulator QGIS plugin

Go to: `Plugins -> Manage and Install Plugins...` in QGIS Desktop. Under the All tab, search for PET simulator. Click on PET simulator and then click Install Plugin. We recommend clicking OK to the popup question below to avoid troubles later on. The figures below show the outlook of the different screens of the plugin. Figure C.5 showcases the second screen of the plugin. This window needs the input directories of the base maps and the set.csv directory. Input is for example `D:\project \run10 \data \` and output is `D:\project \run10 \sim25 \` and the label is run10sim25.

Figure C.4: Qgis plugin PETs window 1 static parameters.

The following figure C.5 showcases the second screen of the plugin. All the climate parameters are visible.

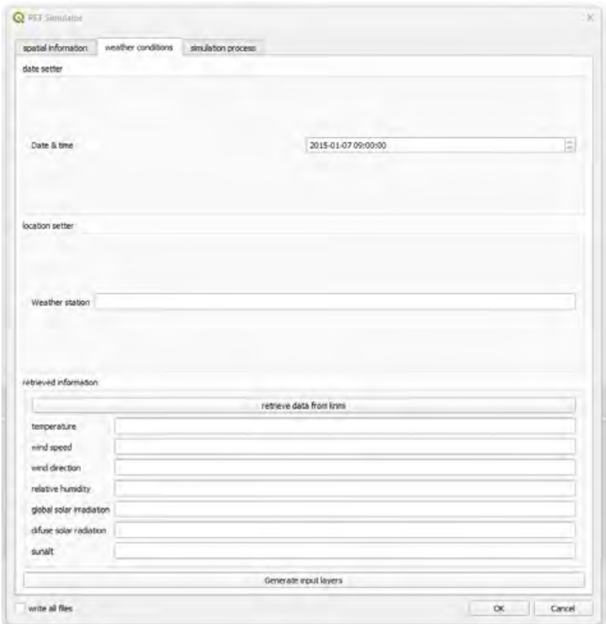


Figure C.5: Qgis plugin PETs window 2 dynamic parameters.

Figure C.6 showcases the third screen of the plugin. Here you can run the program. Check buttons will appear if one of the python calculation files are well excuted.

Figure C.6: Qgis plugin PETs window 3 calculation screen.

Eventually the results are stored as geotiffs in the directories clip, input, output, scale and tif. Input is the extended research area of the research area. Scale are the scaled wind, svf and fveg for the averaging windows. Output showcases the in-between results and endresults. Tif directory outputs tifs for report documentation.

Libraries required

The required installment of running the code are the packages gdal in python as well as in QGIS python environment. This can be installed with downloading a wheel. The wheel can be retrieved by `https://github.com/cgohlke/geospatial-wheels/releases/tag/v2023.7.16`

Listing C.2: Mean Squared Errors 1000x1000 m gebied

```
import pip

def install_whl(path):
pip.main(['install', path])

install_whl("path/to/gdal.whl")
```

D

Extended research area eastern wind Wageningen

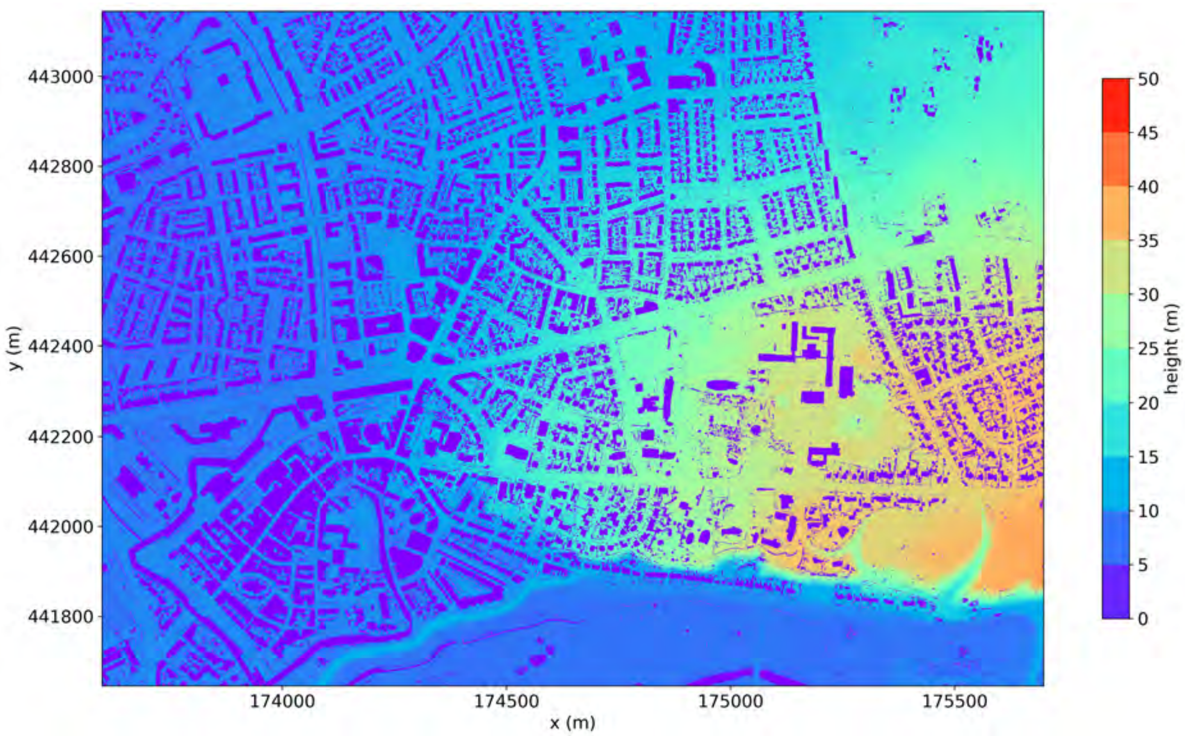


Figure D.1: DTM

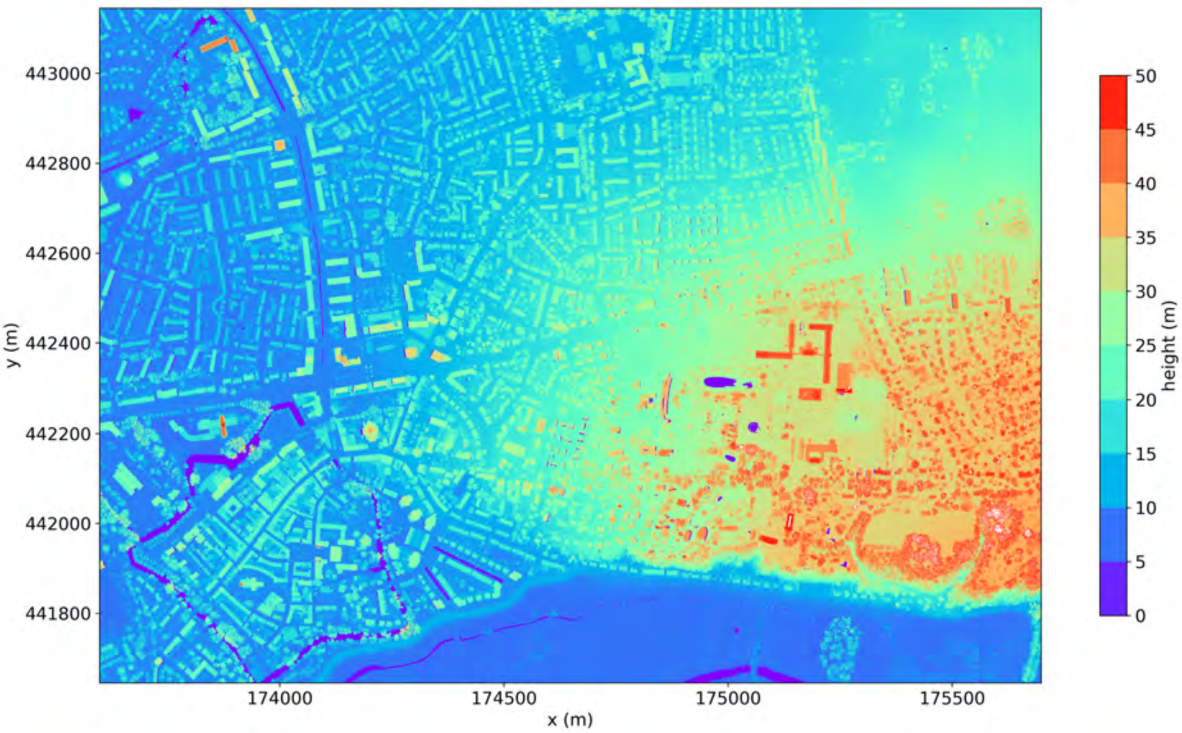


Figure D.2: DSM

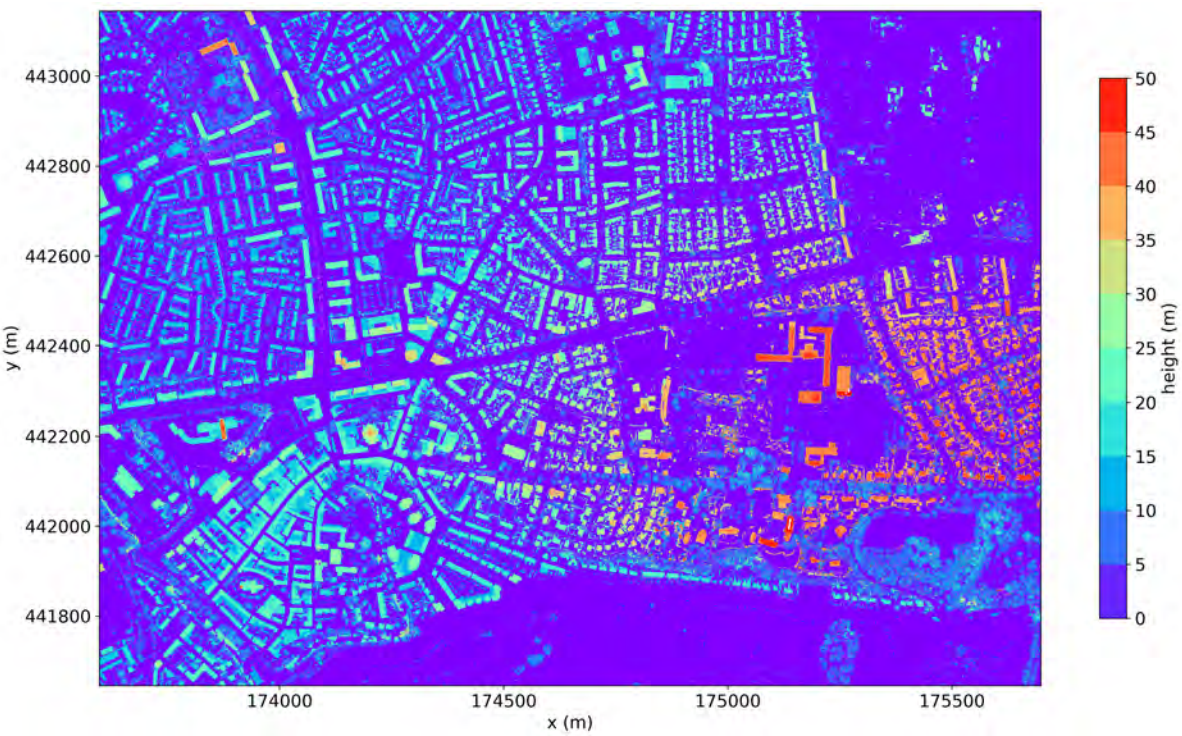


Figure D.3: DSM - DTM

Figure D.4: Building mask.

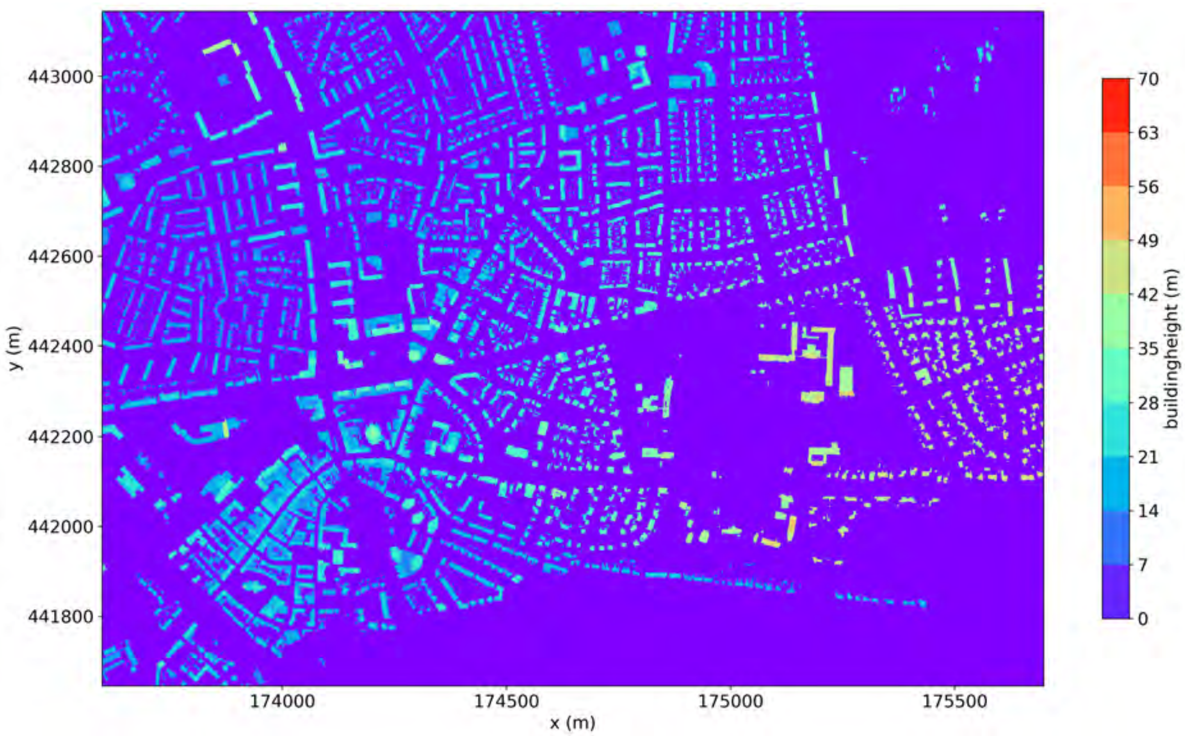


Figure D.5: Building height.

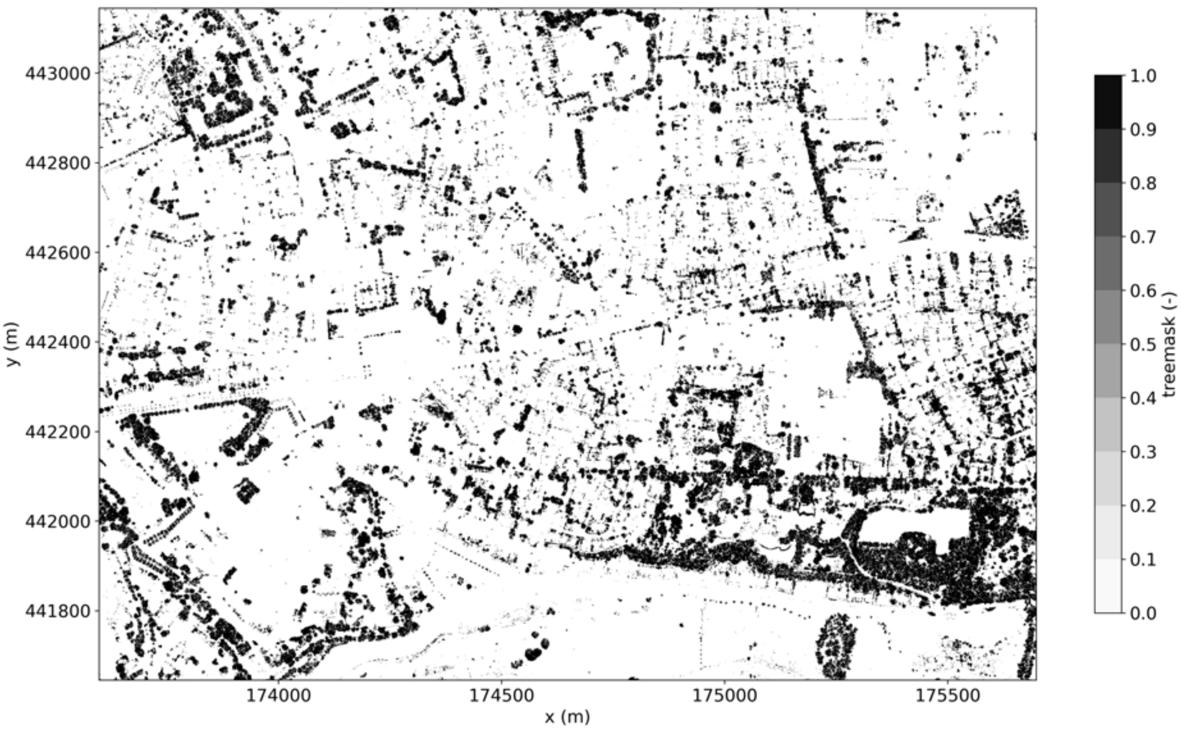


Figure D.6: Building mask.

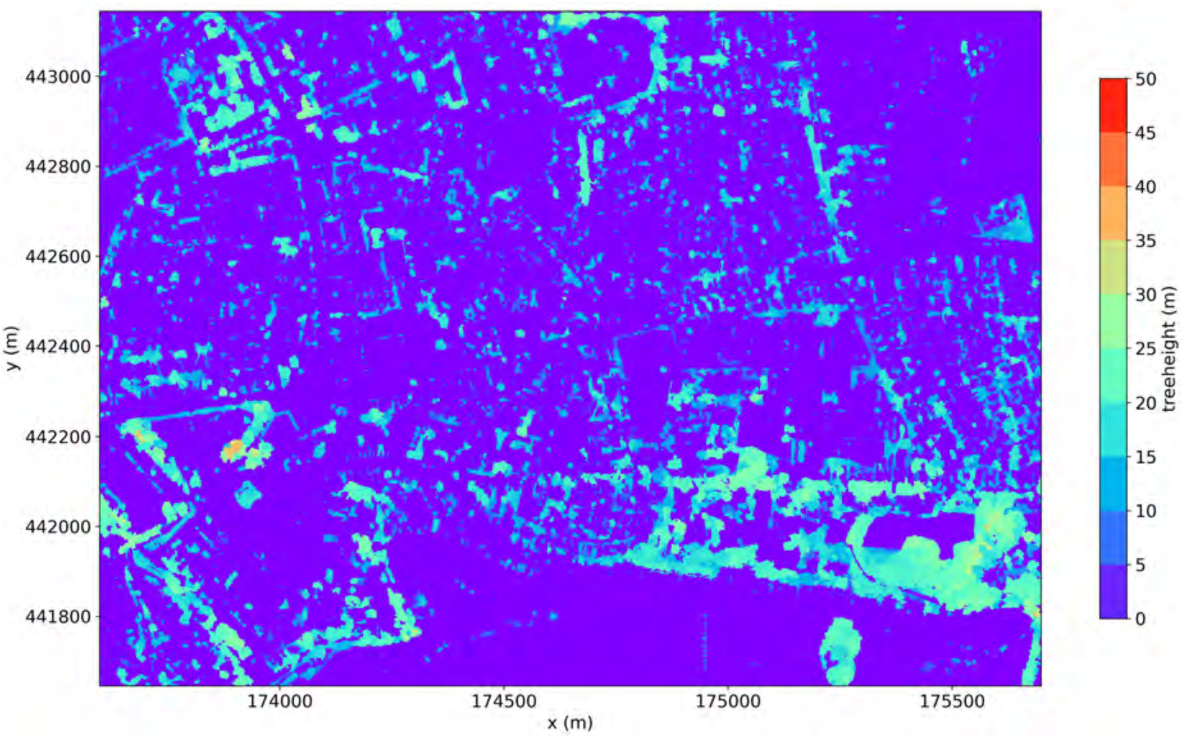


Figure D.7: Tree mask.

Figure D.8: Sky view factor.

Figure D.9: Sky view factor mask.

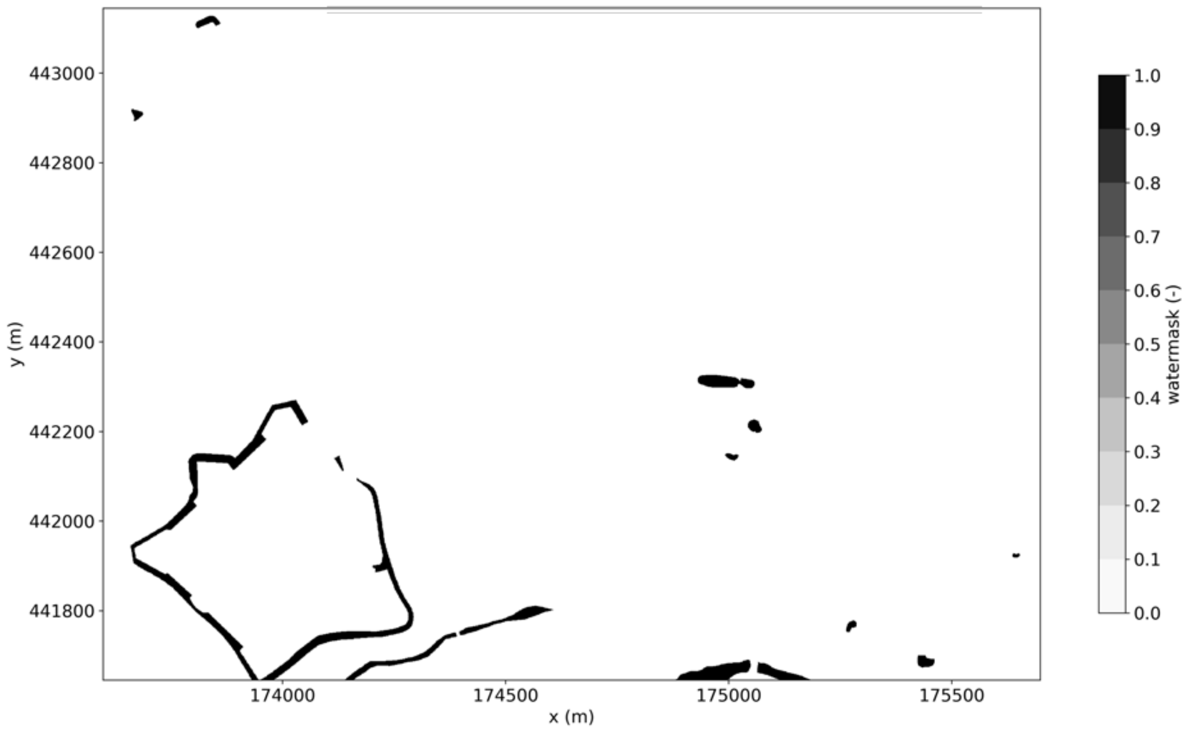


Figure D.10: Water mask.

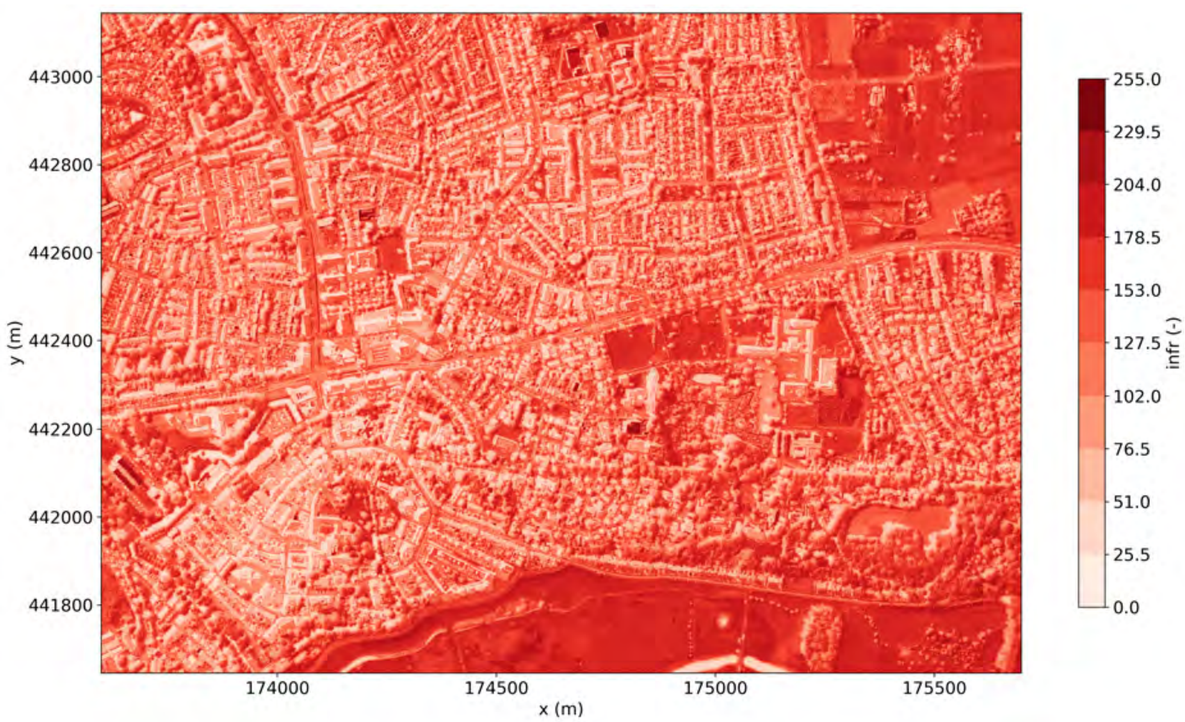


Figure D.11: NDVI near infrared.

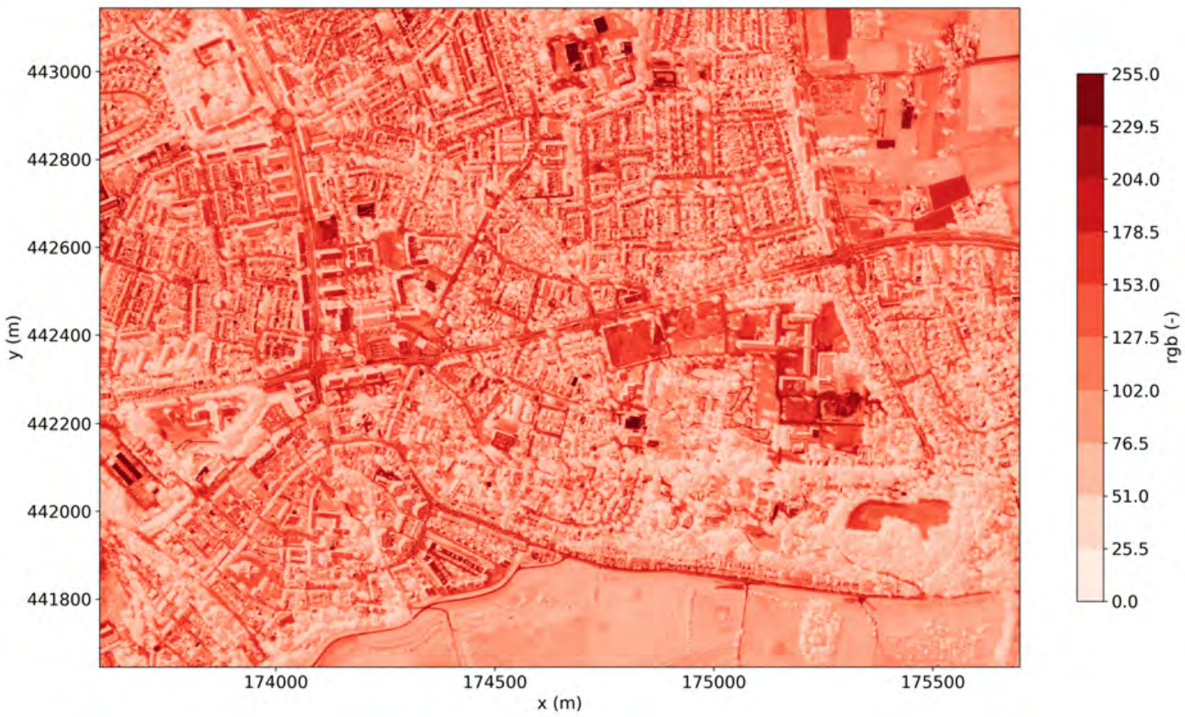


Figure D.12: NDVI red green blue.

Figure D.13: Shadow 1200 LST.

E

Extended research area eastern wind Rotterdam

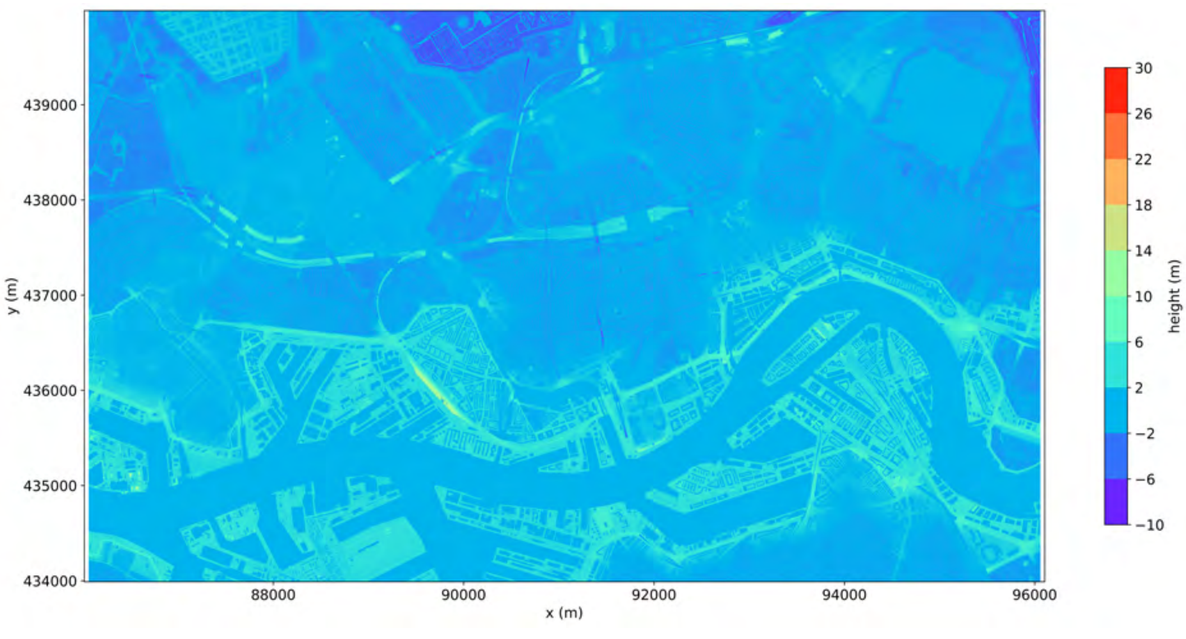


Figure E.1: DTM

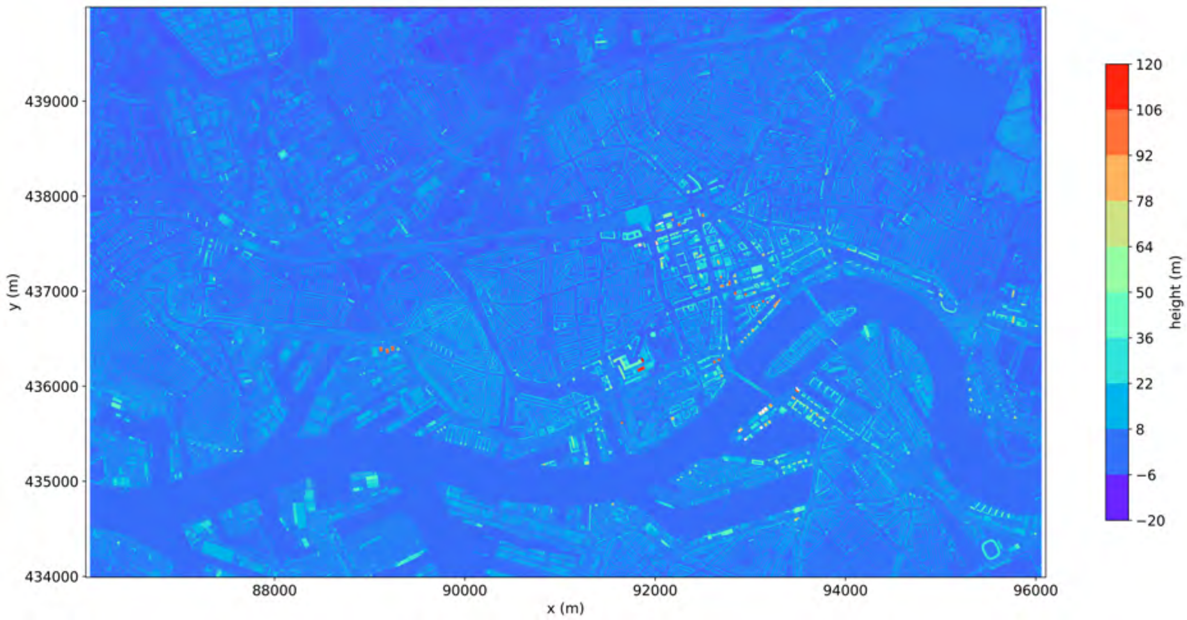


Figure E.2: DSM

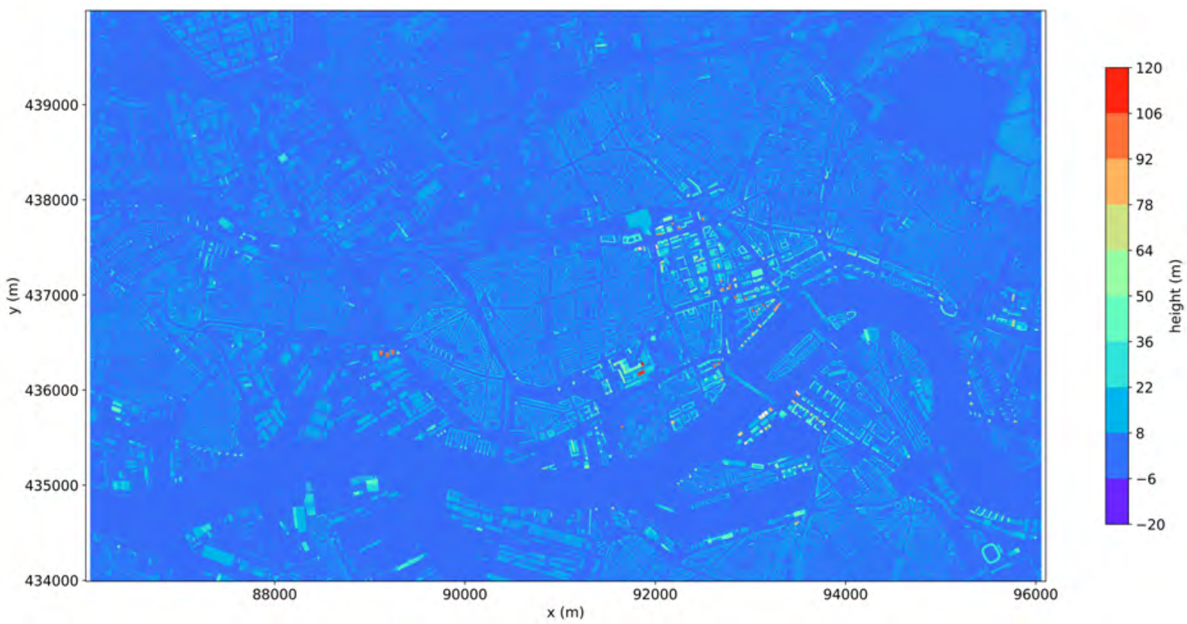


Figure E.3: DSM - DTM

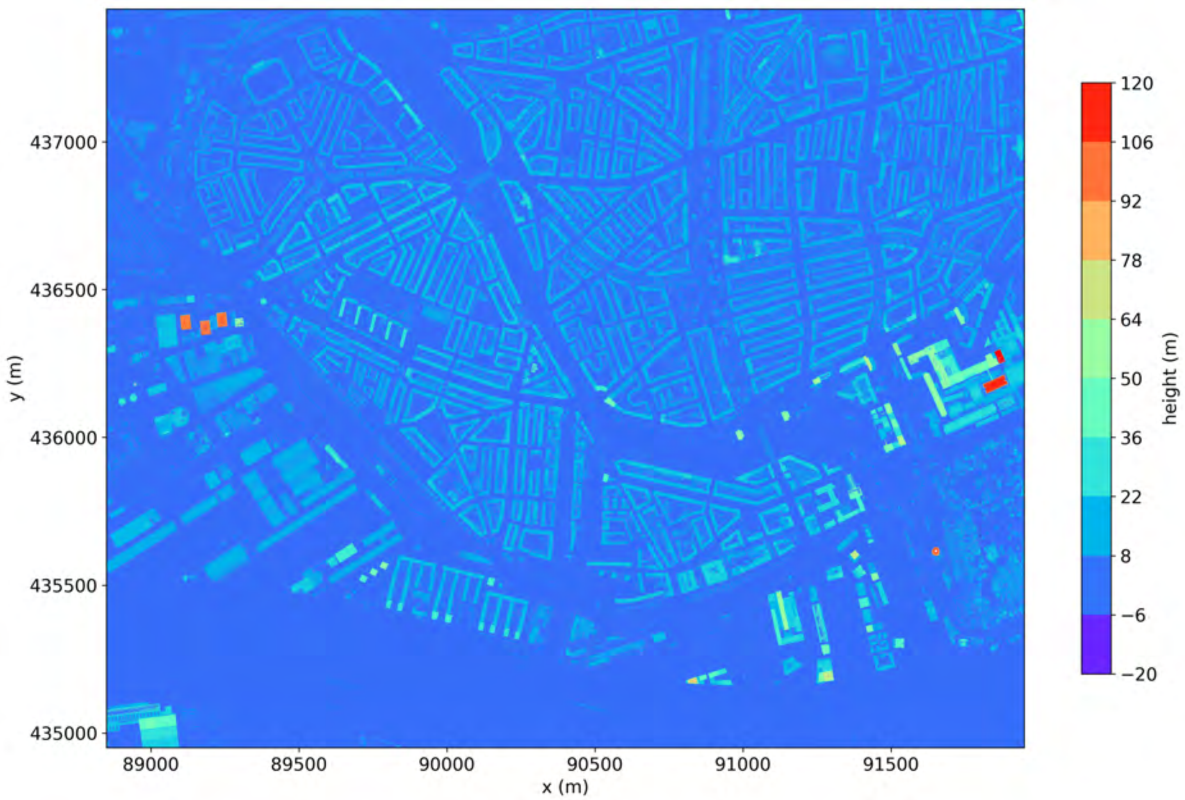


Figure E.4: DSM - DTM

Figure E.5: Building mask.

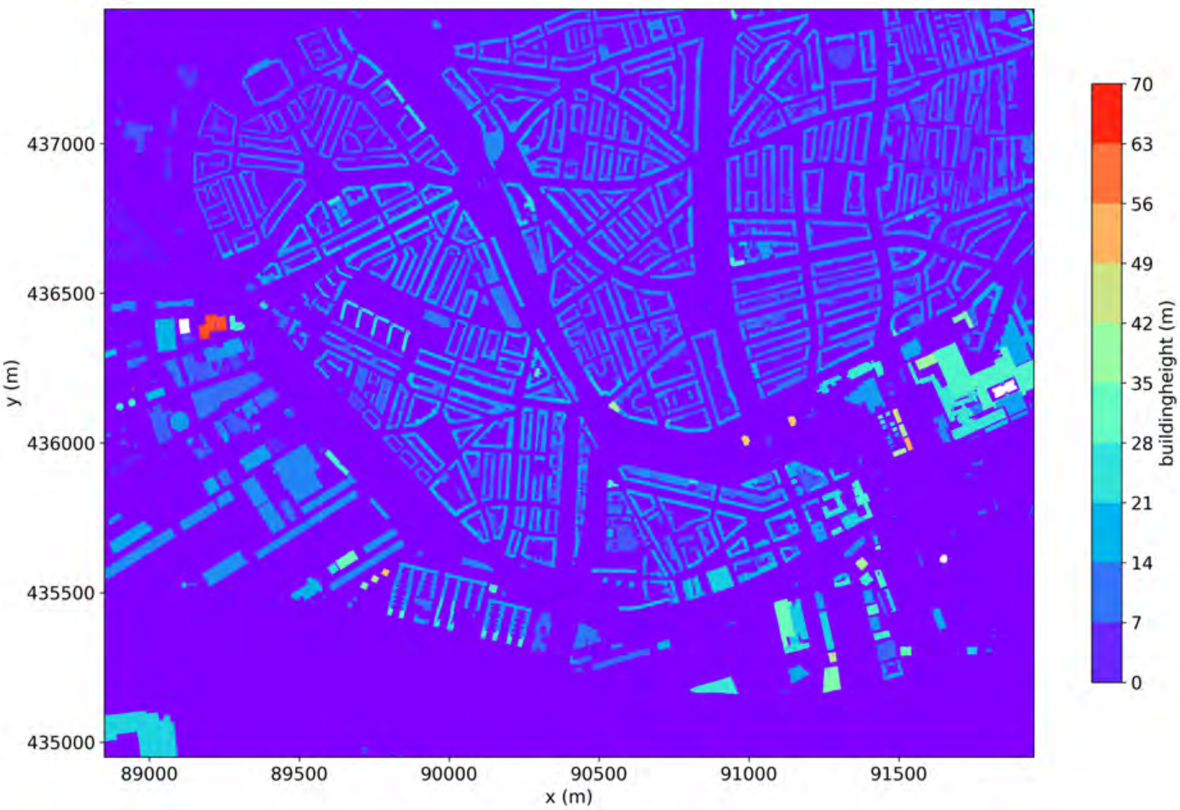


Figure E.6: Building height.

Figure E.7: Building mask.

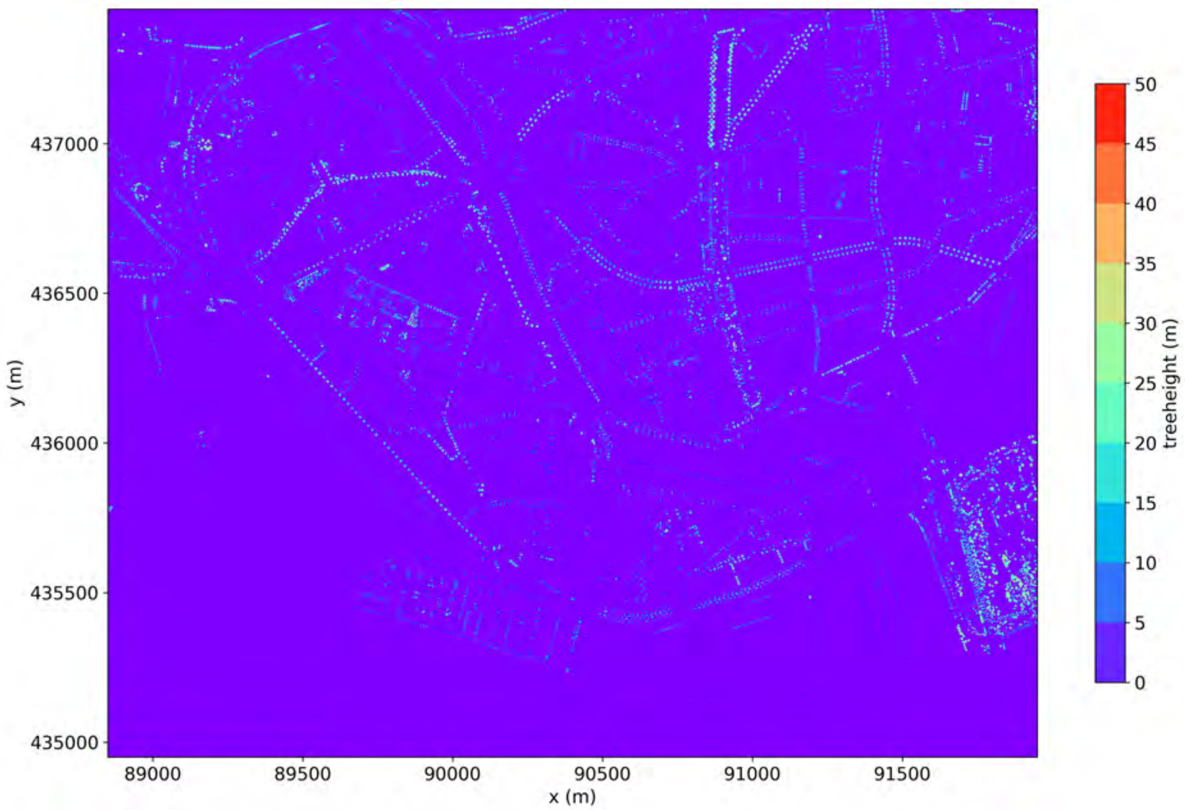


Figure E.8: Tree mask.

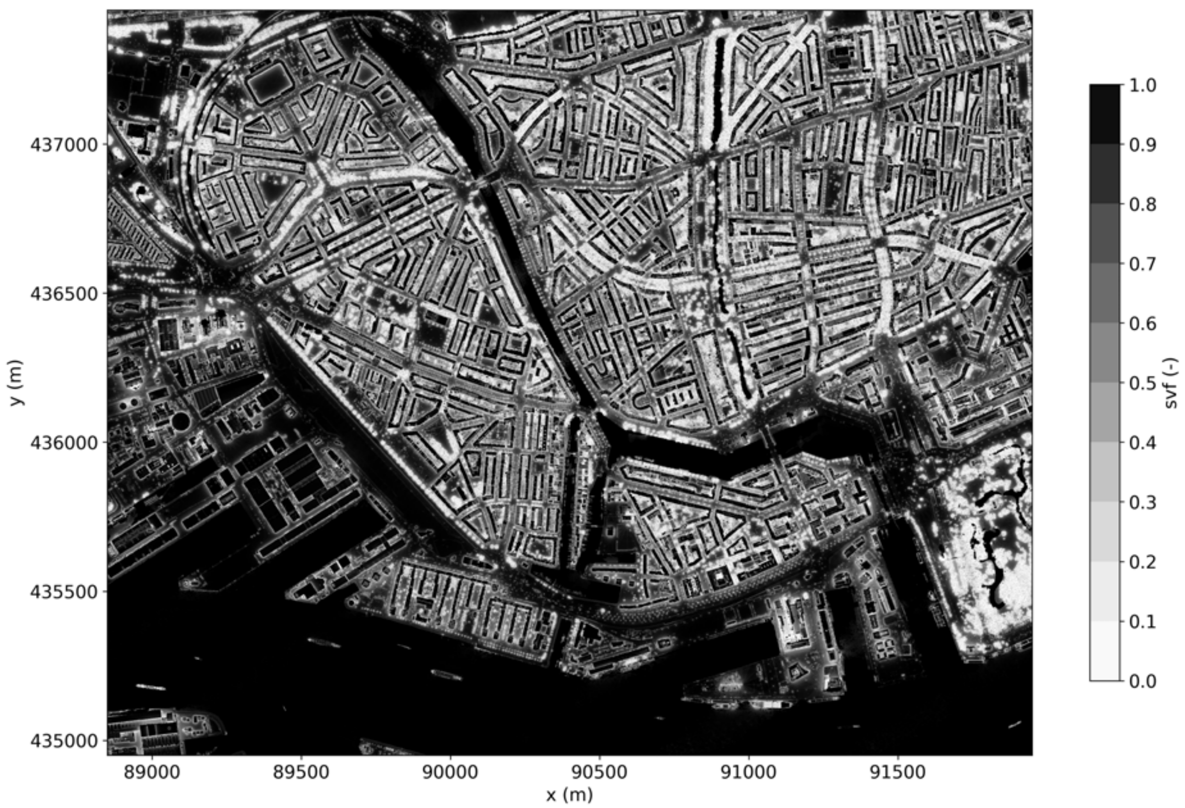


Figure E.9: Sky view factor.

Figure E.10: Sky view factor mask.

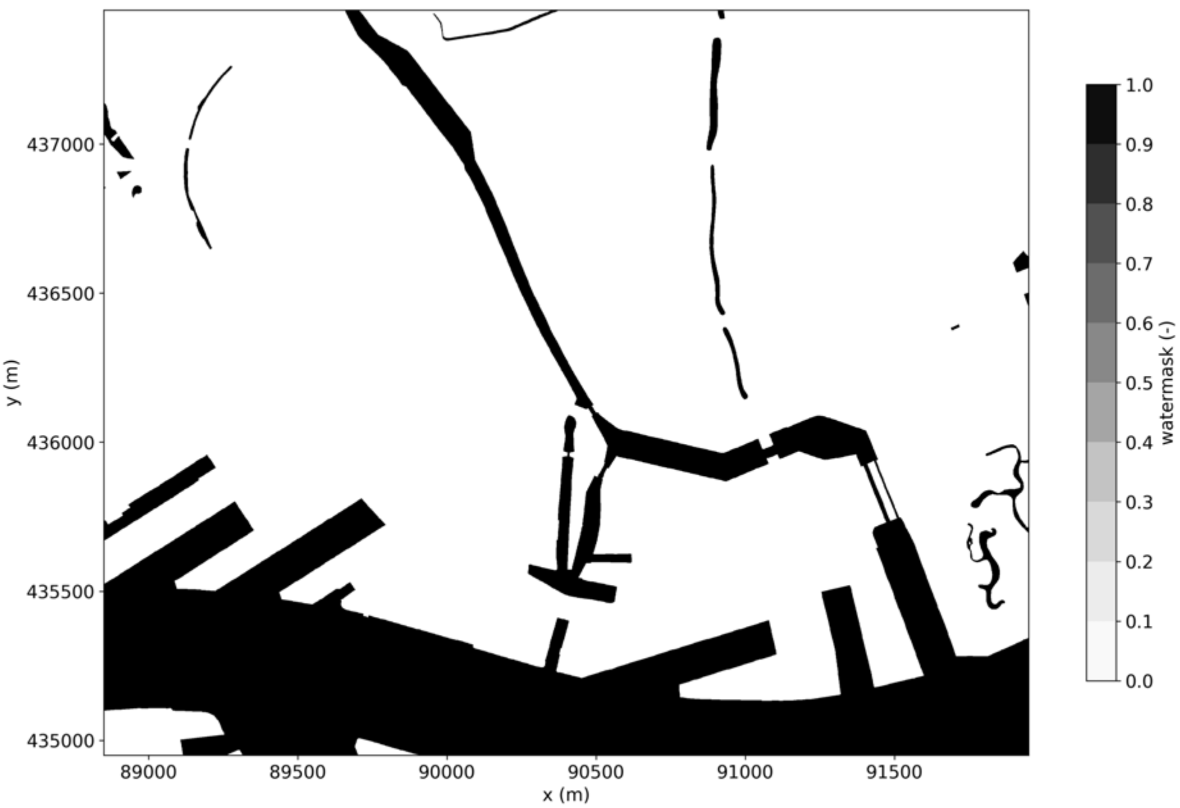


Figure E.11: Water mask.

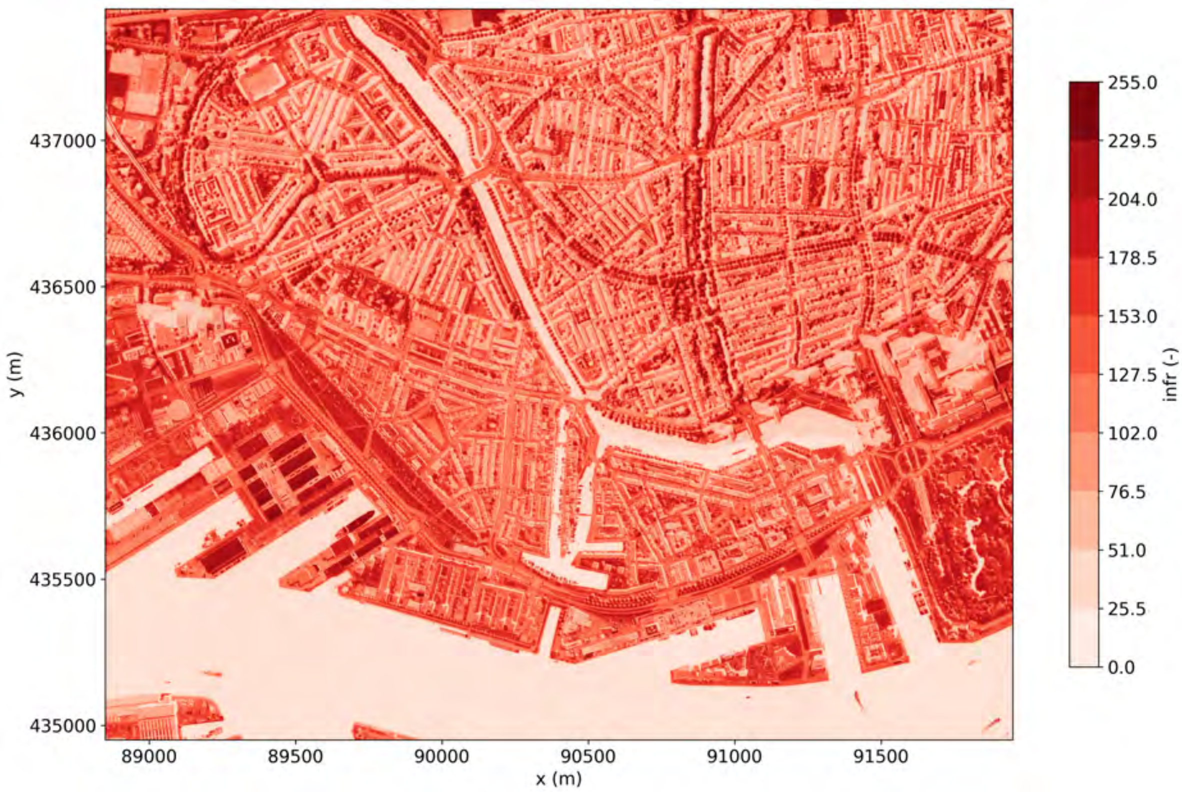


Figure E.12: NDVI near infrared.

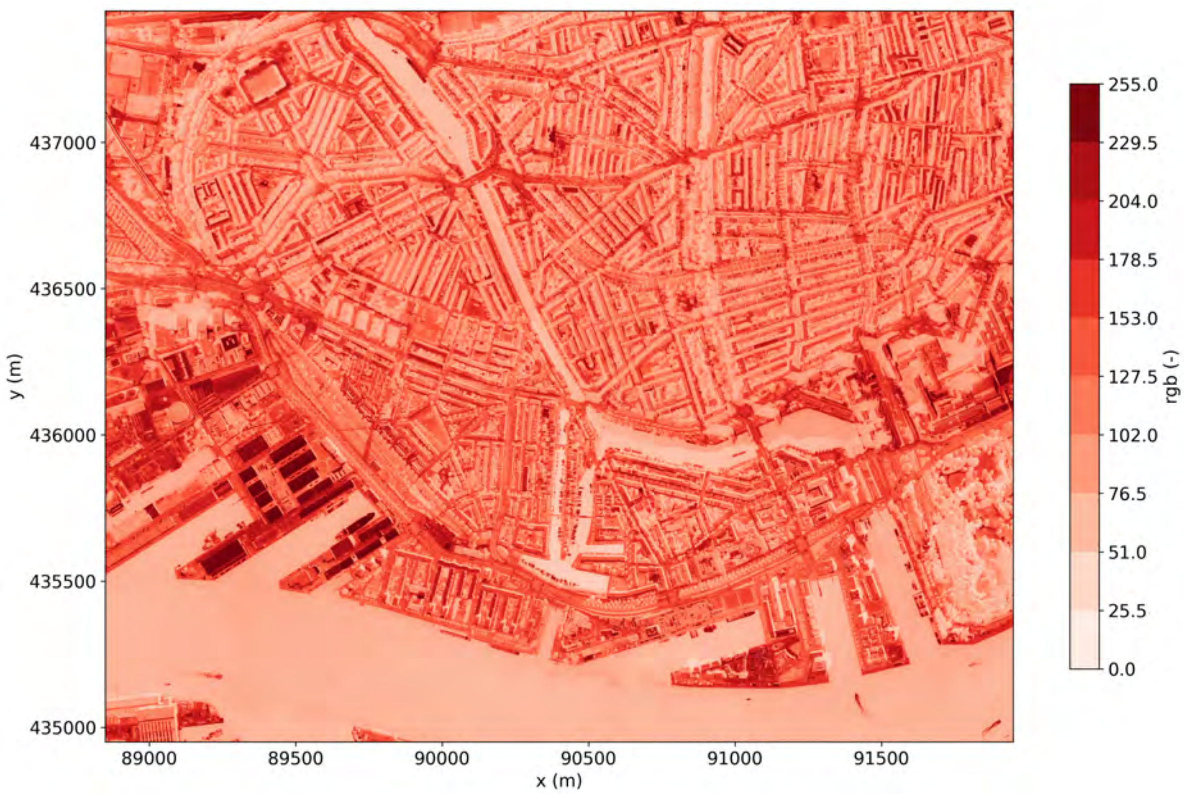


Figure E.13: NDVI red green blue.

Figure E.14: Shadow 1200 LST.

F

Diurnal table

	17-Jun	17-May	18-May	19-May	19-Apr	20-Apr	20-Mar
0	0.748	0.782	0.807	0.91	0.9	1	1
1	0.667	0.64	0.704	0.78	0.757	0.888	0.866
2	0.602	0.573	0.617	0.675	0.71	0.728	0.69
3	0.525	0.49	0.533	0.59	0.543	0.609	0.56
4	0.449	0.355	0.435	0.49	0.413	0.49	0.38
5	0.281	0.15	0.227	0.32	0.15	0.256	0.107
6	0.127	0.078	0.095	0.12	0.057	0.079	0.015
7	0.063	0.025	0.032	0.04	0	0.007	-0.02
8	0.019	-0.013	-0.009	-0.005	-0.02	-0.02	-0.007
9	-0.015	-0.02	-0.02	-0.02	-0.005	0.006	0.007
10	-0.02	-0.001	-0.003	-0.004	0.013	0.01	0.029
11	0	0.025	0.02	0.016	0.037	0.033	0.05
12	0.03	0.056	0.048	0.042	0.063	0.056	0.074
13	0.065	0.09	0.08	0.071	0.09	0.082	0.108
14	0.117	0.165	0.136	0.111	0.15	0.128	0.161
15	0.205	0.27	0.215	0.176	0.222	0.184	0.228
16	0.335	0.413	0.325	0.27	0.318	0.27	0.312
17	0.532	0.6	0.485	0.386	0.45	0.366	0.424
18	0.747	0.803	0.662	0.546	0.6	0.506	0.556
19	0.906	0.92	0.849	0.716	0.762	0.651	0.695
20	0.975	0.978	0.932	0.877	0.89	0.803	0.838
21	1	1	0.979	0.941	0.95	0.901	0.911
22	0.931	0.925	1	0.981	0.982	0.958	0.964
23	0.849	0.83	0.918	1	1	0.983	0.984

G

Additional concept figures

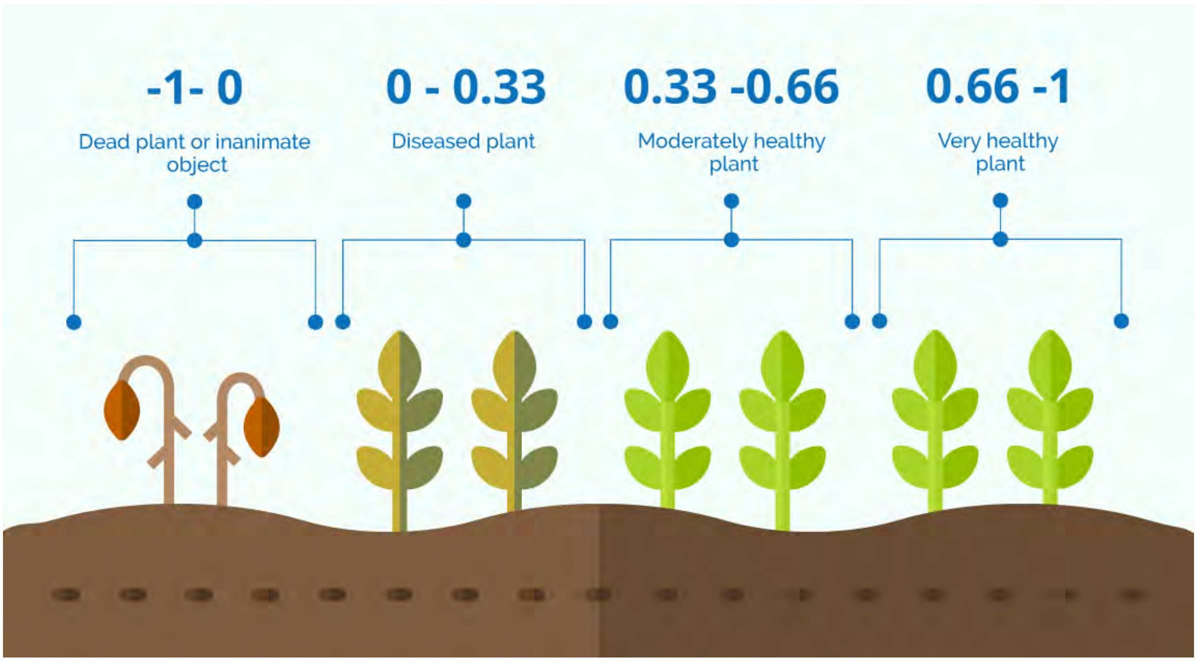


Figure G.1: NDVI values retrieved from [eesa, 2024]

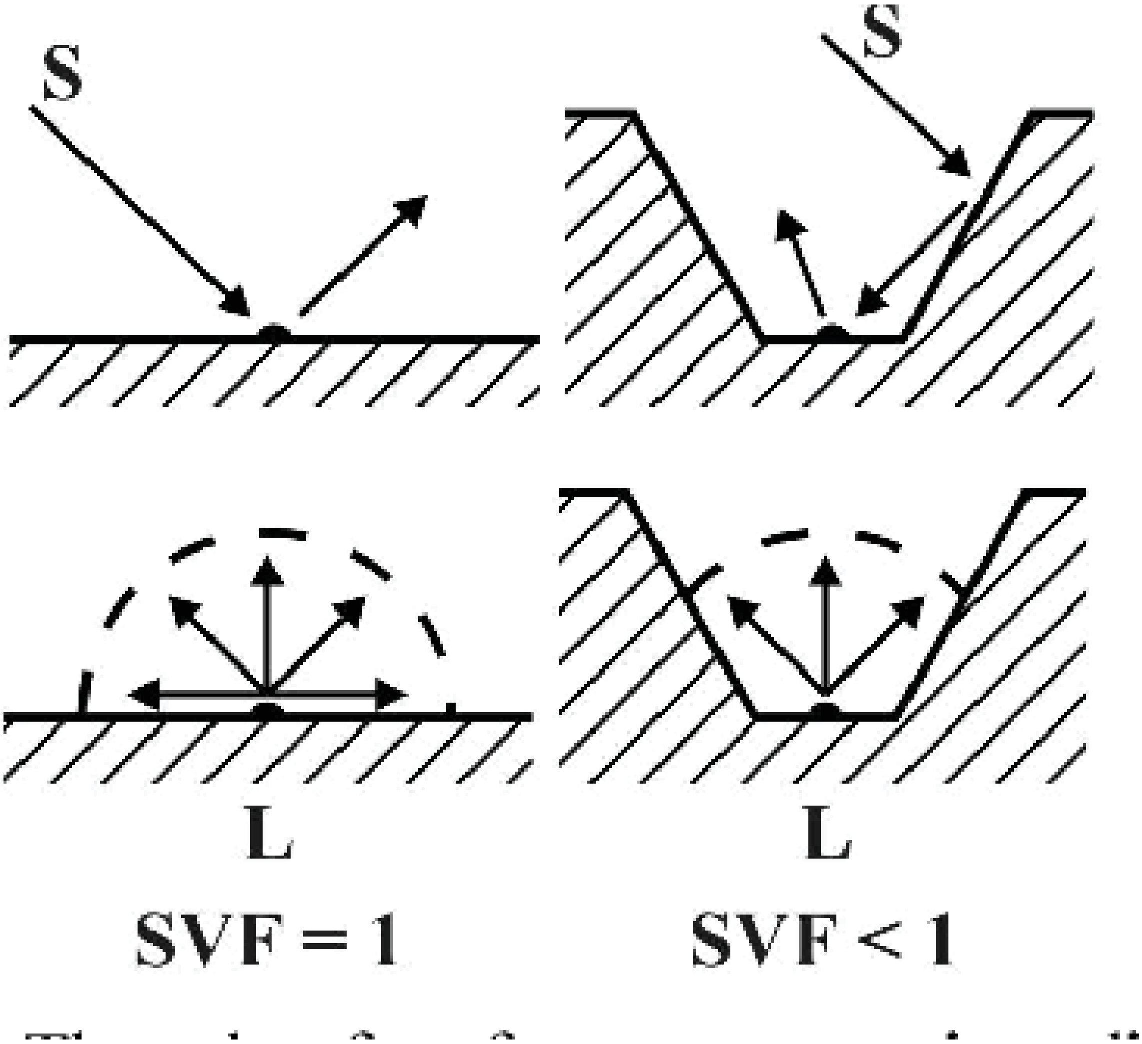


Figure G.2: Sky view factor[Hämmerle et al., 2011]

H

Original python code

H.1. sytse/fraction_area_buildings_treeregr.py

```
from IPython import get_ipython
get_ipython().magic('reset -sf')

import numpy as np
from PIL import Image
#from osgeo import gdal

#ds = gdal.Open('D:/DGRW/UHImax95_denhaag_zoetermeer.tif')
#channel = np.array(ds.GetRasterBand(1).ReadAsArray())

#im = Image.open('D:/DGRW/denhaag/CID/vegfraction_denhaag_zoetermeer_2040green.
    tif')
#im = Image.open('D:/DGRW/denhaag/CID/larger/
    vegfraction_denhaag_zoetermeer_2040_lp.tif')
im = Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/buildings_meanheight_2.tif')
im2 = Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/buildings_mask_mean_2.tif')
#im3b = Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043
    /NL_heatmap/Wageningen/output/trees/treegrid.tif')
im3= Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/trees/trees_ahn.tif')
im4= Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/trees/tree_mask.tif')




bheights = np.array(im)
trees = np.array(im3)*0.9
#trees_ahn=np.array(im3b)*0.9
mask_tree=np.array(im4)
mask = np.array(im2)
w=np.shape(im)[1]
h=np.shape(im)[0]

#print tree_height
#np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/wind/tree_effect/base/tree_height.csv',trees,
    delimiter=',',fmt='%10.5f')
#np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/output/wind/tree_effect/ahn/tree_height_ahn.csv',
```

```
    trees_ahn,delimiter=',',fmt='%10.5f')

#
latarray=np.zeros(shape=(h,w))
lonarray=np.zeros(shape=(h,w))
ymin=172075
ymax=176425
xmin=440675
xmax=444815
latmin=xmin+(xmax-xmin)/(2*h)
latmax=xmax-(xmax-xmin)/(2*h)
lonmin=ymin+(ymax-ymin)/(2*w)
lonmax=ymax-(ymax-ymin)/(2*w)
##cells=32*48
##create lat and lons
for i in enumerate(lonarray[0]):
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
#print('lonarray',lonarray)
for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)
#print('latarray',latarray)
#    for j in enumerate(latarray[i[0]]):
#        print(i[0],j[0])
#    for j in enumerate(latarray[i]):
#        print(i,j)

#vegfra_array=np.zeros(shape=(h/4,(w+1)/4,3))
#urban_2d=np.zeros(shape=(cells,3))
height_2d=np.zeros(shape=(0,3))
area_2d=np.zeros(shape=(0,3))
building_tree_2d=np.zeros(shape=(0,4))
lambda_2d=np.zeros(shape=(0,3))
front_2d=np.zeros(shape=(0,3))
front_tree_2d=np.zeros(shape=(0,3))
wind_2d=np.zeros(shape=(0,3))
wind_notree_2d=np.zeros(shape=(0,3))
wind_tree_2d=np.zeros(shape=(0,3))
mean_area_2d=np.zeros(shape=(0,3))
tree_area_2d=np.zeros(shape=(0,3))
##urban_new=[[]]
#for i in range(50,len(heights)-50,10):
#    for j in range(50,len(heights[0])-50,10):
#        item=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(
    heights[i-50:i+50,j-50:j+50])]
#        area=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(mask
    [i-50:i+50,j-50:j+50])]

z0_grass=0.03
k=0.4

refwind=1/0.63501
red_grass=np.round(refwind*np.log(1.2/z0_grass)/np.log(10/z0_grass),2)
red_60_10=np.log(10/z0_grass)/np.log(60/z0_grass)
#trees
#CS=0.003 from Raupach 1994.
#CR=0.3

winddir=True    # True is winddirection, False is no wind direction
WE=True         #WE= True means West or east, False, north or south
verspringend=False
unbc=140 #positive is east or south, negative is west or north
```

```
width=140
length=280
#height_thres=10

cellsize=1
if winddir:
    if WE:
        horc=length
        verc=width
        unbwc=unbc
        unbnc=0
    else:
        horc=width
        verc=length
        unbnc=unbc
        unbwc=0
else:
    horc=175
    verc=175
    unbnc=35
    unbwc=35
#     unbc=0

#outsize=1
unbw=int(unbwc/cellsize/2)
unbn=int(unbnc/cellsize/2)
hor=int(horc/cellsize/2)
ver=int(verc/cellsize/2)
out=abs(int(unbc/cellsize/4))
#for i in range(945,1050,out):
total_area=hor*2*ver*2
buildingfactor=0.6
treefactor=0.5*0.6

for i in range(ver-unbn,len(bheights)-ver-unbn,out):
#for i in range(ver-unbn,350,out):
#for i in range(2000,2900,out):
    print(i)
    for j in range(hor-unbw,len(bheights[0])-hor-unbw,out):
#     for j in range(hor-unbw,350,out):
#     for j in range(1500,2400,out):
#         print(j)
#     for j in range(hor-unb,len(heights[0])-hor-unb,int(unb/2)):
#         area=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.sum(mask[i-ver:i+ver,j-hor+unb:j+hor+unb])]
        switch=False
        mean_area=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(mask[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])]
        tree_area=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(mask_tree[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])]
        if mean_area[2]>0:
            building_height=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(bheights[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])/mean_area[2]]
            switch=True
        else:
            building_height=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]
        if tree_area[2]>0:
            tree_height=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.mean(trees[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])/
```

```
                tree_area[2]]
            tree_height_regr=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.max(7.721*tree_height[2]**0.5,0)]
            switch=True
        else:
            tree_height=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]
            tree_height_regr=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]
        if switch:
            #weigh heights from trees en buildings
#            height_com_pre=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),max((building_height[2]*mean_area[2]+tree_height[2]*tree_area[2]*treefactor/buildingfactor)/(mean_area[2]+tree_area[2]*treefactor/buildingfactor),4)]
            height_com_pre=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),max((building_height[2]*mean_area[2]+tree_height_regr[2]*tree_area[2]*treefactor/buildingfactor)/(mean_area[2]+tree_area[2]*treefactor/buildingfactor),4)]
#            height_com=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),max(max(height[2],tree_height[2]),4)]

        front=0
        tree=0
        building=0
        #easterly wind
#        if wind:
        if winddir:
            if WE:
                for m in range(i-ver+unbn,i+ver+unbn-1,1):
                    for n in range(j-hor+unbw,j+hor+unbw-1,2):
                        if bheights[m,n+2]-bheights[m,n]!=0:
                            front+=abs(bheights[m,n+2]-bheights[m,n])*0.5*buildingfactor
                            building+=abs(bheights[m,n+2]-bheights[m,n])*0.5*buildingfactor
#                        elif trees[m,n+2]-trees[m,n]!=0:
#                            front+=abs(trees[m,n+2]-trees[m,n])*0.5*treefactor
#                            tree+=abs(trees[m,n+2]-trees[m,n])*0.5*treefactor
                        elif trees[m,n+4]-trees[m,n]!=0:
                            front+=abs(trees[m,n+4]-trees[m,n])*0.5*treefactor
                            tree+=abs(trees[m,n+4]-trees[m,n])*0.5*treefactor
            else:
#                j=1085

                for n in range(j-hor+unbw,j+hor+unbw-1,1):
                    for m in range(i-ver+unbn,i+ver+unbn-1,2):
                        if bheights[m+2,n]-bheights[m,n]!=0:
                            front+=abs(bheights[m+2,n]-bheights[m,n])*0.5*buildingfactor
                            building+=abs(bheights[m+2,n]-bheights[m,n])*0.5*buildingfactor
                        elif trees[m+2,n]-trees[m,n]!=0:
                            front+=abs(trees[m+2,n]-trees[m,n])*0.5*treefactor
                            tree+=abs(trees[m+2,n]-trees[m,n])*0.5*treefactor
#                            print(m,n,abs(heights[m+1,n]-heights[m,n]))
        else:
#           print('no wind')
           for m in range(i-ver+unbn,i+ver+unbn-1,1):
                for n in range(j-hor+unbw,j+hor+unbw-1,2):
                    if bheights[m,n+2]-bheights[m,n]!=0:
```

```
                        front+=abs(bheights[m,n+2]-bheights[m,n])*0.25*buildingfactor
                        building+=abs(bheights[m+2,n]-bheights[m,n])*0.25*buildingfactor
                    elif trees[m,n+2]-trees[m,n]!=0:
                        front+=abs(trees[m,n+2]-trees[m,n])*0.25*treefactor
                        tree+=abs(trees[m,n+2]-trees[m,n])*0.25*treefactor
           for n in range(j-hor+unbw,j+hor+unbw-1,1):
                for m in range(i-ver+unbn,i+ver+unbn-1,2):
                    if bheights[m+2,n]-bheights[m,n]!=0:
                        front+=abs(bheights[m+2,n]-bheights[m,n])*0.25*buildingfactor
                        building+=abs(bheights[m+2,n]-bheights[m,n])*0.25*buildingfactor
                    elif trees[m+2,n]-trees[m,n]!=0:
                        front+=abs(trees[m+2,n]-trees[m,n])*0.25*treefactor
                        tree+=abs(trees[m+2,n]-trees[m,n])*0.25*treefactor


#        print("")
#        print(i,j,front/total_area)
#        print(" ")

        #fit for ahn tree to treefile (bomenbestand)
        tree_regr= 45.45*(tree**0.5)

        front_regr= building + tree_regr
        building_tree=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),building/total_area,tree/total_area]

        if front_regr> 25 and switch == True: # bij hele kleine oppervlakten gewoon op 0 laten, moet hoogte hebben zit ook in BW script
#            lambda1_pre=front/total_area
            height_com=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),max(height_com_pre[2],4)]
            lambda1=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),min(front_regr/total_area+0.015,0.33)]
            front1=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),front_regr]
            front1_tree=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),tree_regr]
            lambda_tree=min(tree/total_area,0.33)
            if switch is False:
                raise "fix front height issue"
#            height2=np.array([8,16,24])
            if lambda1[2] < 0.08:
                if verspringend:
                    z0=0.075*height_com[2]
                    d=0.078*height_com[2]
                    zw=2*height_com[2]
                    A=-0.41*height_com[2]
                    B=0.59
                else:
                    z0=0.048*height_com[2]
                    d=0.066*height_com[2]
                    zw=2*height_com[2]
                    A=-0.35*height_com[2]
                    B=0.56

            elif lambda1[2] <0.135:
                if verspringend:
```

```
                    z0=0.140*height_com[2]
                    d=0.26*height_com[2]
                    zw=1.9*height_com[2]
                    A=-0.45*height_com[2]
                    B=0.58
                else:
                    z0=0.071*height_com[2]
                    d=0.26*height_com[2]
                    zw=2.5*height_com[2]
                    A=-0.35*height_com[2]
                    B=0.50

            elif lambda1[2] <0.18:
                if verspringend:
                    z0=0.150*height_com[2]
                    d=0.32*height_com[2]
                    zw=1.4*height_com[2]
                    A=-0.73*height_com[2]
                    B=0.83
                else:
                    z0=0.084*height_com[2]
                    d=0.32*height_com[2]
                    zw=2.7*height_com[2]
                    A=-0.34*height_com[2]
                    B=0.48
            elif lambda1[2] <0.265:
                if verspringend:
                    z0=0.140*height_com[2]
                    d=0.47*height_com[2]
                    zw=1.3*height_com[2]
                    A=-0.82*height_com[2]
                    B=0.88
                else:
                    z0=0.08*height_com[2]
                    d=0.42*height_com[2]
                    zw=1.5*height_com[2]
                    A=-0.56*height_com[2]
                    B=0.66
            else:
                if verspringend:
                    z0=0.084*height_com[2]
                    d=0.65*height_com[2]
                    zw=1.3*height_com[2]
                    A=-0.62*height_com[2]
                    B=0.68
                else:
                    z0=0.077*height_com[2]
                    d=0.57*height_com[2]
                    zw=1.2*height_com[2]
                    A=-0.85*height_com[2]
                    B=0.92


#             if height_com > height_thres:
            ustar=refwind/red_60_10*k/np.log((60-d)/z0)
            #uzw= refwind/red_60_10*np.log((zw-d)/z0)/np.log((60-d)/z0) #uh ~= uzw otherwise uh is too low. In reality use 17.8 and fill zw in z.
            uzw= ustar/k*np.log((zw-d)/z0) # same as previous statement
            uh=uzw-ustar/B*np.log((A+B*zw)/(A+B*height_com[2]))
```

```
#            wind=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),min(uh*np.exp(9.6*lambda1[2]*(1.2/height_com[2]-1)),red_grass)] #redundant but safe meausure, reduntant because uh cannot be larger than red_grass
#            wind=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),min(uh*np.exp(9.6*lambda1[2]*(1.2/height_com[2]-1)),red_grass-lambda_tree/treefactor)]
            wind=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),min(uh*np.exp(9.6*lambda1[2]*(1.2/height_com[2]-1)),red_grass)]
            wind_notree=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),min(uh*np.exp(9.6*building/total_area*(1.2/height_com[2]-1)),red_grass)]
            wind_tree=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),uh*np.exp(9.6*tree/total_area*(1.2/height_com[2]-1))]
            wind_notree_2d=np.append(wind_notree_2d,[wind_notree],axis=0)
#            if lambda1_pre/0.0025 < height_com_pre[2]:
#                print(np.round(latarray[i,j],4),np.round(lonarray[i,j],4),lambda1_pre/0.0025,uh,wind[2])
##                if wind[2]==1:
##                    stop
            if tree_regr > 25:
                wind_tree_2d=np.append(wind_tree_2d,[wind_tree],axis=0)


#            else:
        else:
            wind=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),red_grass]
            height_com=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]

            lambda1=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]
            front1=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]
            front1_tree=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),0]

        wind_2d=np.append(wind_2d,[wind],axis=0)
        wind_notree_2d=np.append(wind_notree_2d,[wind_notree],axis=0)
        wind_tree_2d=np.append(wind_tree_2d,[wind_tree],axis=0)
        height_2d=np.append(height_2d,[height_com],axis=0) #note the [] around item, this ensures that dimensions are the same
#        area_2d=np.append(area_2d,[area],axis=0)
        building_tree_2d=np.append(building_tree_2d,[building_tree],axis=0)
        lambda_2d=np.append(lambda_2d,[lambda1],axis=0)
        front_2d=np.append(front_2d,[front1],axis=0)
        front_tree_2d=np.append(front_tree_2d,[front1_tree],axis=0)
        mean_area_2d=np.append(mean_area_2d,[mean_area],axis=0)
        tree_area_2d=np.append(tree_area_2d,[tree_area],axis=0)

if winddir:
    if WE:
        if unbc >0:
            np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/wind_E.csv',wind_2d,delimiter=',',fmt='%10.5f')
            #output for research, not necessary for creation PET heat map
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/front.csv',front_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/front_tree.csv',front_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/H_E.csv',
```

```
height_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/lambda_E.csv',lambda_2d,delimiter=',',fmt='%10.5f')
#
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/wind_E_notree.csv',wind_notree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/wind_E_tree.csv',wind_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/building_tree_E.csv',building_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/plan_area_fraction_E.csv',mean_area_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_effect/ahn4regr/tree_area_fraction_E.csv',tree_area_2d,delimiter=',',fmt='%10.5f')
#       else:
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/H_W.csv',height_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/lambda_W.csv',lambda_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_W.csv',wind_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_W.csv',wind_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_W_notree.csv',wind_notree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_W_tree.csv',wind_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/building_tree.csv',building_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/plan_area_fraction.csv',mean_area_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/tree_area_fraction.csv',tree_area_2d,delimiter=',',fmt='%10.5f')
#   else:
#       if unbc >0:
#            np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/wind_S.csv',wind_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/H_S.csv',height_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/lambda_S.csv',lambda_2d,delimiter=',',fmt='%10.5f')

#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/wind_S_notree.csv',wind_notree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/wind_S_tree.csv',wind_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/building_tree_S.csv',building_tree_2d,delimiter=',',fmt='%10.5f')
```

```
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/plan_area_fraction_S.csv',mean_area_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/test/tree_area_fraction_S.csv',tree_area_2d,delimiter=',',fmt='%10.5f')
#
#        else:
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/H_N.csv',height_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/lambda_N.csv',lambda_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/wind_N.csv',wind_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/wind_N_notree.csv',wind_notree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/wind_N_tree.csv',wind_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/building_tree_N.csv',building_tree_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/plan_area_fraction_N.csv',mean_area_2d,delimiter=',',fmt='%10.5f')
#           np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/output/wind/tree_area_fraction_N.csv',tree_area_2d,delimiter=',',fmt='%10.5f')
#else:
#
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/H_C.csv',height_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/lambda_C.csv',lambda_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_C.csv',wind_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_C_notree.csv',wind_notree_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_C_tree.csv',wind_tree_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/building_tree_C.csv',building_tree_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/plan_area_fraction_C.csv',mean_area_2d,delimiter=',',fmt='%10.5f')
#    np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/UserData/koopm043/NL_heatmap/Wageningen/output/wind/tree_area_fraction_C.csv',tree_area_2d,delimiter=',',fmt='%10.5f')

#
##get boundaries
```

```
##xmin= lonmin+(lonmax-lonmin)/(w-1)*(10-2)
##xmax= lonmin+(lonmax-lonmin)/(w-1)*(934+2)
##ymin= latmax-(latmax-latmin)/(h-1)*(10-2)
##ymax= latmax-(latmax-latmin)/(h-1)*(610+2)
##xspace= (lonmax-lonmin)/(w-1)*4
##yspace= (latmax-latmin)/(h-1)*4
#
```

H.2. sytse/ndvi_infr_large.py

```
from IPython import get_ipython
get_ipython().magic('reset -sf')

import numpy as np
from PIL import Image
im0_rgb = Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/
    koopm043/NL_heatmap/Wageningen/vegfra/ndvi_large/ndvi_rgb_merge.tif')
im0_infr= Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/
    koopm043/NL_heatmap/Wageningen/vegfra/ndvi_large/ndvi_infr_merge.tif')

im=im0_rgb
w=np.shape(im)[1]
h=np.shape(im)[0]
#
latarray=np.zeros(shape=(h,w))
lonarray=np.zeros(shape=(h,w))

ymin=171223
ymax=177323
#ymax=176223
#ymin=ymax-1990
xmin=439783
xmax=445683

#xmax=444657
#xmin=xmax-2000

latmin=xmin+0.5
latmax=xmax-0.5
lonmin=ymin+0.5
lonmax=ymax-0.5
out=1
##cells=32*48
##create lat and lons
for i in enumerate(lonarray[0]):
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
#    print('lonarray',lonarray)
for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)

lufo_rgb=np.array(im0_rgb)
lufo_infr=np.array(im0_infr)
ndvi_img=np.array(im0_infr)

r=lufo_rgb[:,:,0].astype(int)
g=lufo_rgb[:,:,1].astype(int)
b=lufo_rgb[:,:,2].astype(int)
infr=lufo_infr[:,:,0].astype(int)
#ndvi=g/(r+g+b)
ndvi_infr=(infr-r)/(infr+r)
ndvi_infr[ndvi_infr<0]=0
#vari=(g-r)/(g+r-b)
#vari[vari<0]=0
#tgi=(g-0.39*r-0.61*b)/g
#tgi[tgi<0]=0

#lufo[:,:,1]=255
#img = Image.fromarray(lufo)
#ndvi=0.55
```

```
#red=(1-ndvi**0.5)*255
#green=ndvi**0.5*255

ndvi_img[:,:,0]=infr
ndvi_img[:,:,1]=0
ndvi_img[:,:,2]=0

#ndvi_2d_temp=[np.ravel(latarray[:]),np.ravel(lonarray[:]),np.ravel(ndvi[:])]
#ndvi_2d=np.array(ndvi_2d_temp).transpose()
ndvi_infr_temp=[np.ravel(latarray[:]),np.ravel(lonarray[:]),np.ravel(ndvi_infr
    [:])]
ndvi_infr_2d=np.array(ndvi_infr_temp).transpose()

img = Image.fromarray(ndvi_img)
np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/vegfra/ndvi_large/ndvi_infr_merge.csv',ndvi_infr_2d,
    delimiter=',',fmt='%5.3f')
#img.save('E:/NL_heatmap/Wageningen/vegfra/ndvi/ndvi_infr_0.tif')
```

H.3. sytse/vegetation_footprints.py

```
from IPython import get_ipython
get_ipython().magic('reset -sf')

import numpy as np
from PIL import Image
#from osgeo import gdal

day=False
wind=False  # True is winddirection, False is no wind direction
WE=True     #WE= True means West or east, False, north or south
unbc=-900   #positive is east or south, negative is west or north
width=500
length=1100

if day:
    im = Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/
        vegfraction_water_cropland_day_28992_Wageningen_begroeidbgt.tif')
else:
    im = Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/
        vegfraction_water_cropland_28992_Wageningen_begroeidbgt.tif')
vegfra = np.array(im)
w=np.shape(im)[1]
h=np.shape(im)[0]
#
latarray=np.zeros(shape=(h,w))
lonarray=np.zeros(shape=(h,w))
ymin=171322
ymax=177291
xmin=439813
xmax=445583
latmin=xmin+(xmax-xmin)/(2*h)
latmax=xmax-(xmax-xmin)/(2*h)
lonmin=ymin+(ymax-ymin)/(2*w)
lonmax=ymax-(ymax-ymin)/(2*w)
##cells=32*48
##create lat and lons
for i in enumerate(lonarray[0]):
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
#print('lonarray',lonarray)
for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)
vegfra_2d=np.zeros(shape=(0,3))
area_2d=np.zeros(shape=(0,3))
lambda_2d=np.zeros(shape=(0,3))
cellsize=25
outsize=25
if wind:
    if WE:
        horc=length
        verc=width
        unbwc=unbc
        unbnc=0
    else:
        horc=width
        verc=length
        unbnc=unbc
        unbwc=0
    unbw=int(unbwc/cellsize/2)
    unbn=int(unbnc/cellsize/2)
```

```
else:
    horc=700
    verc=700
    unbw=0
    unbn=0

hor=int(horc/cellsize/2)
ver=int(verc/cellsize/2)
out=int(outsize/cellsize)
for i in range(ver-unbn,len(vegfra)-ver-unbn,out):
    for j in range(hor-unbw,len(vegfra[0])-hor-unbw,out):
        mean_vegfra=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.sum(vegfra[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])/np.sum(vegfra[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw]>0)]
#        print(i,j)
#        print(hor, unbw, j-hor+unbw)
        vegfra_2d=np.append(vegfra_2d,[mean_vegfra],axis=0) #note the [] around item, this ensures that dimensions are the same

if wind:
    if WE:
        if unbc >0:
            if day:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25E_day.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
            else:
                 np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25E_night.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
        else:
            if day:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25W_day2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
            else:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25W_night2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
    else:
        if unbc >0:
            if day:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25S_day.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
            else:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25S_night.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
        else:
            if day:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25N_day2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
            else:
                np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25N_night2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
else:
    if day:
        np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25_calm_day2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
    else:
        np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/vegfra/vegfra25_calm_night2.csv',vegfra_2d,delimiter=',',fmt='%10.5f')
#np.savetxt('E:/NL_heatmap/Wageningen/output/wind/Ad.csv',area_2d,delimiter=',',fmt='%10.5f')
#np.savetxt('E:/NL_heatmap/Wageningen/output/wind/lambda.csv',lambda_2d,delimiter=',',fmt='%10.5f')
```

```
#np.savetxt('E:/NL_heatmap/Wageningen/output/wind/wind.csv',wind_2d,delimiter
    =',',fmt='%10.5f')

#get boundaries
#xmin= lonmin+(lonmax-lonmin)/(w-1)*(10-2)
#xmax= lonmin+(lonmax-lonmin)/(w-1)*(934+2)
#ymin= latmax-(latmax-latmin)/(h-1)*(10-2)
#ymax= latmax-(latmax-latmin)/(h-1)*(610+2)
#xspace= (lonmax-lonmin)/(w-1)*4
#yspace= (latmax-latmin)/(h-1)*4
```

H.4. sytse/skyview_footprints.py

```
from IPython import get_ipython
get_ipython().magic('reset -sf')

import numpy as np
from PIL import Image

im = Image.open('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/
    NL_heatmap/Wageningen/Nynke/urban_morphology/SVF_Wageningen_mean25.tif')
svf = np.array(im)
w=np.shape(im)[1]
h=np.shape(im)[0]
#
latarray=np.zeros(shape=(h,w))
lonarray=np.zeros(shape=(h,w))
ymin=171322
ymax=177291
xmin=439813
xmax=445583
latmin=xmin+(xmax-xmin)/(2*h)
latmax=xmax-(xmax-xmin)/(2*h)
lonmin=ymin+(ymax-ymin)/(2*w)
lonmax=ymax-(ymax-ymin)/(2*w)
##cells=32*48
##create lat and lons
for i in enumerate(lonarray[0]):
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
#print('lonarray',lonarray)
for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)

svf_2d=np.zeros(shape=(0,3))
area_2d=np.zeros(shape=(0,3))
lambda_2d=np.zeros(shape=(0,3))
wind_2d=np.zeros(shape=(0,3))

wind=True # True is winddirection, False is no wind direction
WE=True  #WE= True means West or east, False, north or south
unbc=900  #positive is east or south, negative is west or north
width=500
length=1100
cellsize=25
outsize=25
if wind:
    if WE:
        horc=length
        verc=width
        unbwc=unbc
        unbnc=0
    else:
        horc=width
        verc=length
        unbnc=unbc
        unbwc=0
    unbw=int(unbwc/cellsize/2)
    unbn=int(unbnc/cellsize/2)
else:
    horc=700
    verc=700
    unbc=0
```

```
    unbw=0
    unbn=0
hor=int(horc/cellsize/2)
ver=int(verc/cellsize/2)
out=int(outsize/cellsize)
for i in range(ver-unbn,len(svf)-ver-unbn,out):
#    print(i)
    for j in range(hor-unbw,len(svf[0])-hor-unbw,out):
        perc= np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw]>0)/np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw]>-1)
        if perc >= 0.2:
            mean_svf=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])/np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw]>0)] #
        elif perc >= 0.1: #linearize between svf=1 for 0.1 and svf as executed above
#            print('elif',i,j)
            mean_pre_svf=np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw])/np.sum(svf[i-ver+unbn:i+ver+unbn,j-hor+unbw:j+hor+unbw]>0)
            mean_svf=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),((perc-0.1)/0.1)*mean_pre_svf+(1-(perc-0.1)/0.1)*1]
#            print(perc,mean_pre_svf,mean_svf[2])
        else:
#            print('else',i,j)
            mean_svf=[np.round(latarray[i,j],4),np.round(lonarray[i,j],4),1]
        svf_2d=np.append(svf_2d,[mean_svf],axis=0) #note the [] around item, this ensures that dimensions are the same

if wind:
    if WE:
        if unbc >0:
            np.savetxt('C:/Users/koopm043/OneDrive - WageningenUR/Userdata/koopm043/NL_heatmap/Wageningen/Nynke/urban_morphology/svf25E.csv',svf_2d,delimiter=',',fmt='%10.5f')
#        else:
#            np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/svf/svf25W.csv',svf_2d,delimiter=',',fmt='%10.5f')
#    else:
#        if unbc >0:
#            np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/svf/svf25S.csv',svf_2d,delimiter=',',fmt='%10.5f')
#        else:
#            np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/svf/svf25N.csv',svf_2d,delimiter=',',fmt='%10.5f')
#else:
#    np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/svf/svf25C.csv',svf_2d,delimiter=',',fmt='%10.5f')
```

H.5. sytse/pet_calculate.py

```
from IPython import get_ipython
get_ipython().magic('reset -sf')

import pandas as pd
import numpy as np
import gdal
from PIL import Image

scenario="def"
Nynke=True

#get meteofile and put in panda table
obs_headernames=['YYYYMMDD','month','decade','hour','TT','FF','dd','Q','Qdif','
    sunalt','rh','diurn','UHImax']
FDATA = pd.read_table('D:/Ddrive/koopm043/NL_heatmap/Wageningen/Herwijnen/
    Herwijnen_1juli2015_10_16UTC_%s.csv' %(scenario),sep =",", skiprows=1,
    names=obs_headernames, engine='python')
#FDATA = pd.read_table('D:/Ddrive/koopm043/NL_heatmap/Wageningen/Herwijnen/
    Herwijnen_2-3aug2013_4_4UTC_%s.csv' %(scenario),sep =",", skiprows=1, names
    =obs_headernames, engine='python')

#get GIS static data Wageningen
im4= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/svf/svf_1m_allign.tif
    ')
im5= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/svf/
    svf_1m_mask_allign.tif')
im6 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/vegfra/ndvi/
    ndvi_infr_mask0.16_allign.tif')
im7= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/vegfra/ndvi/
    trees_2m_allign.tif')

firsttime=True
Bveg=0.4
Bnoveg=3
stef=5.67*10**-8
svf = np.array(im4)
svf_mask=np.array(im5)
mask_vegfra=np.array(im6)
trees_2m=np.array(im7)

w=np.shape(im4)[1]
h=np.shape(im4)[0]
#
latarray=np.zeros(shape=(h,w))
lonarray=np.zeros(shape=(h,w))

ymin=172323
ymax=176223
xmin=440883
xmax=444583

latmin=xmin+(xmax-xmin)/(2*h)
latmax=xmax-(xmax-xmin)/(2*h)
lonmin=ymin+(ymax-ymin)/(2*w)
lonmax=ymax-(ymax-ymin)/(2*w)
out=1

##create lat and lons
for i in enumerate(lonarray[0]):
```

```
    lonarray[:,i[0]] = lonmin + (lonmax - lonmin) * i[0]/(w-1)
#    print('lonarray',lonarray)
for i in enumerate(latarray[:,0]):
    latarray[i[0]] = latmax - (latmax - latmin) * i[0]/(h-1)

PET_2d=np.zeros(shape=(0,3))


PETshade=np.zeros(shape=(len(latarray),len(latarray[0])))
PETveg=np.zeros(shape=(len(latarray),len(latarray[0])))
PETnoveg=np.zeros(shape=(len(latarray),len(latarray[0])))

#run through timeseries and get time dependent GIS/meteofields like shadow/sun, wind and urban morphology (winddependent UHI equation Natalie Theeuwes)
for t in range(2,3,1):
#for t in range(0,len(FDATA)):

    month= FDATA['month'].iloc[t]
    decade= FDATA['decade'].iloc[t]
    hour= FDATA['hour'].iloc[t]
    sunalt= FDATA['sunalt'].iloc[t]
    T=FDATA['TT'].iloc[t]
    print(t,hour)
    if sunalt > 0:
        if month == 7:
            if decade == 1:
                if hour ==6:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_0600_LST.tif')
                elif hour == 7:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_0700_LST.tif')
                elif hour == 8:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_0800_LST.tif')
                elif hour == 9:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_0900_LST.tif')
                elif hour == 10:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1000_LST.tif')
                elif hour == 11:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1100_LST.tif')
                elif hour == 12:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1200_LST.tif')
                elif hour == 13:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1300_LST.tif')
                elif hour == 14:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1400_LST.tif')
                elif hour == 15:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1500_LST.tif')
                elif hour == 16:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1600_LST.tif')
                elif hour == 17:
```

```
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1700_LST.tif')
                elif hour == 18:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1800_LST.tif')
                elif hour == 19:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/julyhour/shadow_20140706_1900_LST.tif')
        elif month == 8:
            if decade == 1:
                if hour ==5:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_0500_LST.tif')
                elif hour == 6:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_0600_LST.tif')
                elif hour == 7:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_0700_LST.tif')
                elif hour == 8:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_0800_LST.tif')
                elif hour == 9:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_0900_LST.tif')
                elif hour == 10:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1000_LST.tif')
                elif hour == 11:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1100_LST.tif')
                elif hour == 12:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1200_LST.tif')
                elif hour == 13:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1300_LST.tif')
                elif hour == 14:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1400_LST.tif')
                elif hour == 15:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1500_LST.tif')
                elif hour == 16:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1600_LST.tif')
                elif hour == 17:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1700_LST.tif')
                elif hour == 18:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1800_LST.tif')
                elif hour == 19:
                    im = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/radiation/augusthour/shadow_20140806_1900_LST.tif')
#                im = Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/radiation/august/shadow_20140826_1800_LST.tif')
    FF= FDATA['FF'].iloc[t]
    dd= FDATA['dd'].iloc[t]
    sunalt= FDATA['sunalt'].iloc[t]
    print(dd)
```

```
    if FF >= 1.5: #0-1bft
        if dd <=45:
            if sunalt > 0:
                im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_N_allign.tif') #to do
            else:
                im2= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_N_allign.tif')
                im3= Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_N.tif')
        elif dd<135:
            if sunalt > 0:
                if Nynke:
                    im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/Nynke/urban_morphology/urban_morphology_25m_day_E_allign.tif')
                    print('Nynke')
                else:
                    im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_E_allign.tif') #to do
            else:
                    im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_E_allign.tif') #to do
            print('E ',hour)
            im3= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/wind/wind_E.tif')
        elif dd<225:
            if sunalt > 0:
                im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_S_allign.tif') #to do
            else:
                im2= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_S_allign.tif')
            im3= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/wind/wind_S.tif')
            print('S ',hour)
        elif dd<315:
            if sunalt > 0:
                im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_W_allign.tif') #to do
            else:
                im2= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_W_allign.tif')
            im3= Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind/wind_N.tif')
        else:
            if sunalt > 0:
                im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_N_allign.tif') #to do
            else:
                im2= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_N_allign.tif')
#                im3= Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind_N.tif')
    else:
        if sunalt > 0:
                im2 = Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_day_C_allign.tif') #to do
        else:
                im2= Image.open('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output/urban_morphology_25m_night_C_allign.tif')
```

```
        im3= Image.open('D:/UserData/koopm043/NL_heatmap/Wageningen/output/wind
            /wind_C.tif')

    urban=np.array(im2)
    Ta=urban[:]*FDATA['UHImax'].iloc[t]*FDATA['diurn'].iloc[t]+T

    Qgl= FDATA['Q'].iloc[t]
    Qdif= FDATA['Qdif'].iloc[t]
    sunalt= FDATA['sunalt'].iloc[t]

    rh=FDATA['rh'].iloc[t]

    Tw=T*np.arctan(0.15198*(rh+8.3137)**0.5)+np.arctan(T+rh)-np.arctan(rh
        -1.676)+0.0039184*rh**1.5*np.arctan(0.023101*rh)-4.686 #use station T

    wind = ((np.array(im3)-0.125)*0.5829+0.125)*FF #substitutie S.13 en S.14
    wind[wind<0.5]=0.5 #minimum wind speed is 0.5 m/s
    wind_temp=np.ravel(latarray[:]),np.ravel(lonarray[:]),np.ravel(wind)
    wind_res=np.array(wind_temp).transpose()
#         np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/verification/
    wind_2aug2013_%s_0.5.csv' %(hour),wind_res,delimiter=',',fmt='%6.2f')

        #1=sun 0=shadow
    # PETsun does not exist at nighttime and a simpler routine is followed in
        the night, i.e in the night PETshade is calculated everywhere
    if Qgl > 0 and sunalt > 0: #QDir < 120W is shadow #beam
        sun_temp = np.array(im)
        sun=sun_temp[74:-6,90:]*(1-trees_2m)
#         np.savetxt('D:/UserData/koopm043/NL_heatmap/Wageningen/radiation/sun.
    tif,ndvi_infr_2d,delimiter=',',fmt='%5.3f')

        PETshade=(latarray[:],lonarray[:],-12.14+1.25*Ta[:]-1.47*np.log(wind
            [:])+0.060*Tw+0.015*svf[:]*Qdif+0.0060*(1-svf[:])*stef*(Ta
            [:]+273.15)**4)*(1-sun[:])*svf_mask[:]
        PETveg=(latarray[:],lonarray[:],-13.26+1.25*Ta[:]+0.011*Qgl-3.37*np.log
            (wind[:])+0.078*Tw+0.0055*Qgl*np.log(wind[:])+5.56*np.sin(sunalt
            /360*2*np.pi)-0.0103*Qgl*np.log(wind[:])*np.sin(sunalt/360*2*np.pi)
            +0.546*Bveg+1.94*svf[:])*mask_vegfra[:]*sun[:]*svf_mask[:]
        PETnoveg=(latarray[:],lonarray[:],-13.26+1.25*Ta[:]+0.011*Qgl-3.37*np.
            log(wind[:])+0.078*Tw+0.0055*Qgl*np.log(wind[:])+5.56*np.sin(sunalt
            /360*2*np.pi)-0.0103*Qgl*np.log(wind[:])*np.sin(sunalt/360*2*np.pi)
            +0.546*Bnoveg+1.94*svf[:])*(1-mask_vegfra[:])*sun[:]*svf_mask[:]

        PET_tiff=PETshade[2]+PETnoveg[2]+PETveg[2]
        [cols,rows]=[np.shape(PET_tiff)[0],np.shape(PET_tiff)[1]]


    else:
        PETshade=(latarray[:],lonarray[:],-12.14+1.25*Ta[:]-1.47*np.log(wind
            [:])+0.060*Tw+0.015*svf[:]*Qdif+0.0060*(1-svf[:])*stef*(Ta
            [:]+273.15)**4)*svf_mask[:]

        PET_tiff=PETshade[2]
        [cols,rows]=[np.shape(PET_tiff)[0],np.shape(PET_tiff)[1]]

#create georeferenced Tiff
    im= gdal.Open('C:/Users/koopm043/NL_heatmap/avgPET_1july2015_Herw.tif') #
        pas op deze link is anders dan D:/Drive, dit bestand is verstuurd onder
         onder Imme/Ddrive/koopm043/NL_heatmap
    obj=im.GetRasterBand(1)
    obj_array=obj.ReadAsArray()
```

```
    driver = gdal.GetDriverByName("GTiff")
#     outdata = driver.Create('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output2/
    PET_2aug_tiff_%s_test_Imme.tif' %(hour), rows, cols, 1, gdal.GDT_UInt16)
#     outdata = driver.Create('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output2/
    PET_1July2015_Herw_12UTC.tif' %(hour), rows, cols, 1, gdal.GDT_UInt16)
    outdata = driver.Create('D:/Ddrive/koopm043/NL_heatmap/Wageningen/output2/
        PET_1July2015_Herw_12UTC.tif' %(hour), rows, cols, 1, gdal.GDT_Float32)
    outdata.SetGeoTransform(im.GetGeoTransform())##sets same geotransform as
        input
    im= None
    outdata.GetRasterBand(1).WriteArray(PET_tiff)
    outdata.FlushCache()
```

I

MSE wind old

Listing I.1: MSE between blocksize 1 and blocksize 5 100x100 area

```
R^2 = 0.6411
```

Comparing the blocksize between 5 and 25 there was a high correlation with the r2 score of 0.973.

Listing I.2: MSE between blocksize 5 and blocksize 25 100x100 area

```
R^2 = 0.973
```

But the accuracy of the data declines by comparing the blocksize between 1 and 25 there was a low correlation with the r2 score of -0.04.

Listing I.3: MSE between blocksize 1 and blocksize 25 100x100 area

```
R^2 = 0.5923
```

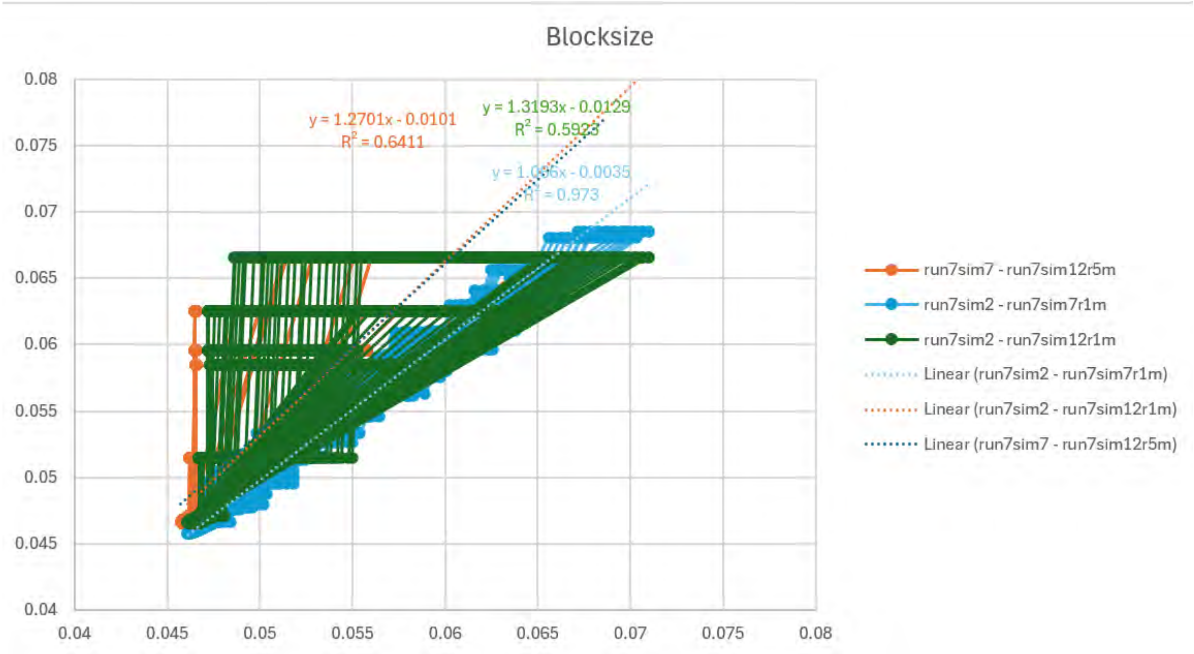


Figure I.1: Trendline time data block size 5m

J

Dates 2023 Rotterdam

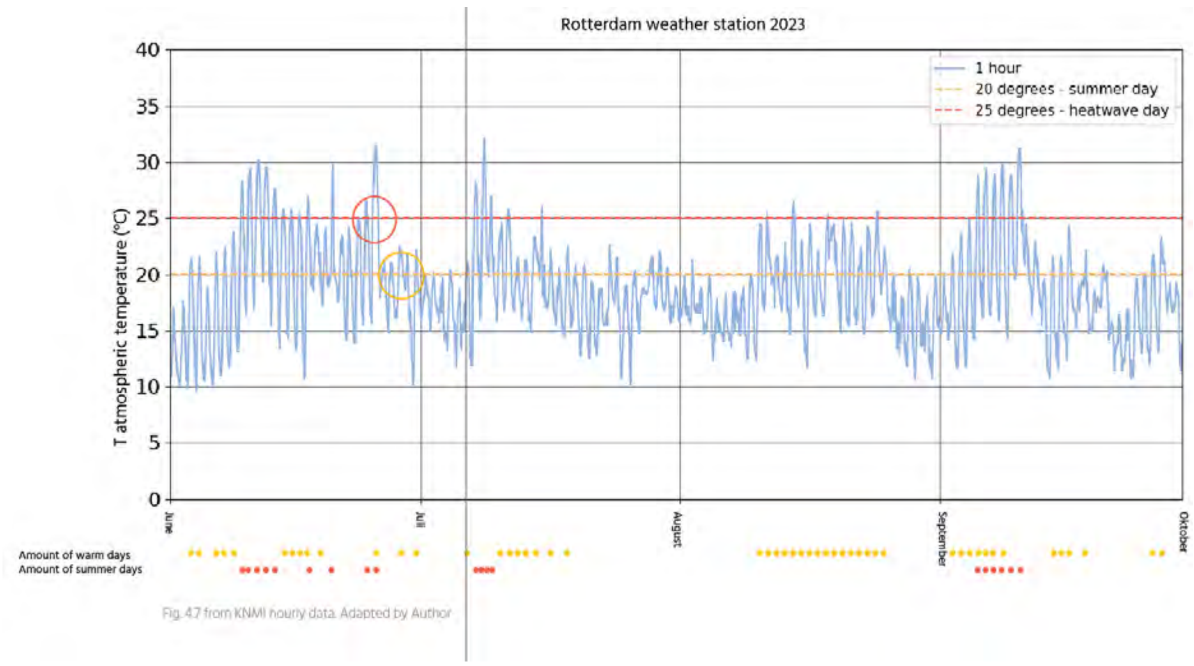
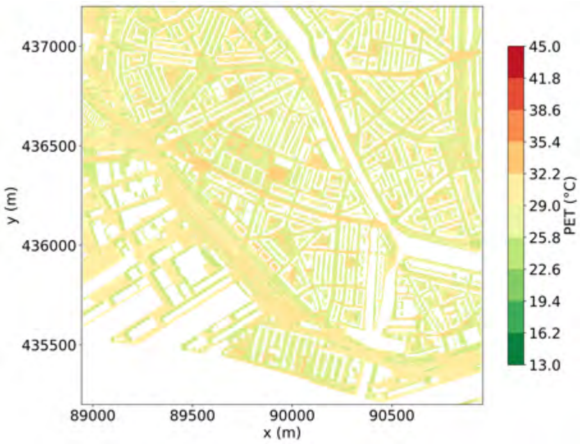


Figure J.1: Fig. T atmospheric temperature for Rotterdam in the months june till september 2023 (Data retrieved from KNMI [0000] postprocessed by author)

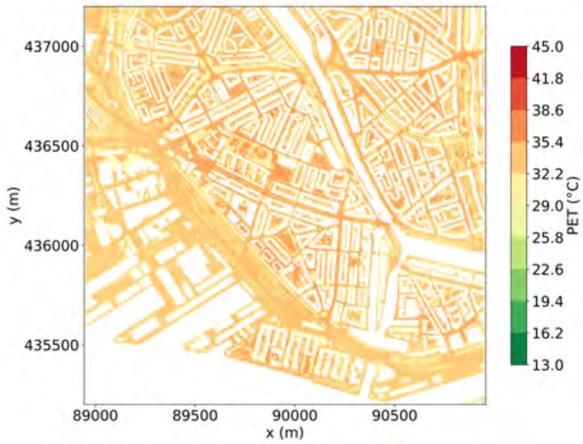
YYYYMMDD	HH	T	FF	DD	Q	Qdif	sunalt	RH	wind	WE	winddir	daynight	diurnal	Tmax	Tmin	U
6/25/2023	9	27.8	3	90	234	148.1063	47.48214	34	TRUE	TRUE	E	day	0.007	31.5	17.9	4.608696
6/25/2023	10	28.4	4	90	271	156.9444	54.84529	32	TRUE	TRUE	E	day	0.029	31.5	17.9	4.608696
6/25/2023	11	29.5	3	90	294	165.8333	59.74484	34	TRUE	TRUE	E	day	0.05	31.5	17.9	4.608696
6/25/2023	12	30.3	5	90	303	170.5556	60.74826	30	TRUE	TRUE	E	day	0.074	31.5	17.9	4.608696
6/25/2023	13	30.6	6	90	311	166.9444	57.46653	33	TRUE	TRUE	E	day	0.108	31.5	17.9	4.608696
6/25/2023	14	31.4	5	90	290	151.3889	51.03978	33	TRUE	TRUE	E	day	0.161	31.5	17.9	4.608696
6/25/2023	15	31.5	6	90	255	136.3031	42.88002	33	TRUE	TRUE	E	day	0.228	31.5	17.9	4.608696
6/25/2023	16	31.4	5	90	210	138.7116	33.94235	32	TRUE	TRUE	E	day	0.312	31.5	17.9	4.608696
6/25/2023	17	30.9	5	90	154	131.1178	24.81072	32	TRUE	TRUE	E	day	0.424	31.5	17.9	4.608696
6/25/2023	18	30.5	5	90	99	102.4217	15.89229	35	TRUE	TRUE	E	day	0.556	31.5	17.9	4.608696

YYYYMMDD	H	T	FF	DD	Q	Qdif	sunalt	RH	wind	WE	winddir	nightday	diurnal	Tmin	Tmax	U
6/28/2023	9	19.9	4	230	89	273.6111	47.48214	80	TRUE	TRUE	W	day	0.007	18.6	22.5	3.652174
6/28/2023	10	20.8	4	230	108	276.3889	54.84529	74	TRUE	TRUE	W	day	0.029	18.6	22.5	3.652174
6/28/2023	11	21.5	4	250	91	320.8333	59.74484	72	TRUE	TRUE	W	day	0.05	18.6	22.5	3.652174
6/28/2023	12	22.4	5	240	140	379.533	60.74826	68	TRUE	TRUE	W	day	0.074	18.6	22.5	3.652174
6/28/2023	13	22.5	5	260	178	355.946	57.46653	68	TRUE	TRUE	W	day	0.108	18.6	22.5	3.652174
6/28/2023	14	21.6	3	260	92	229.1667	51.03978	74	TRUE	TRUE	W	day	0.161	18.6	22.5	3.652174
6/28/2023	15	22	5	270	73	238.8889	42.88002	69	TRUE	TRUE	W	day	0.228	18.6	22.5	3.652174
6/28/2023	16	22	5	260	99	218.0556	33.94235	64	TRUE	TRUE	W	day	0.312	18.6	22.5	3.652174
6/28/2023	17	21.9	4	240	58	127.7778	24.81072	65	TRUE	TRUE	W	day	0.424	18.6	22.5	3.652174
6/28/2023	18	21.5	3	260	34	73.61111	15.89229	66	TRUE	TRUE	W	day	0.556	18.6	22.5	3.652174

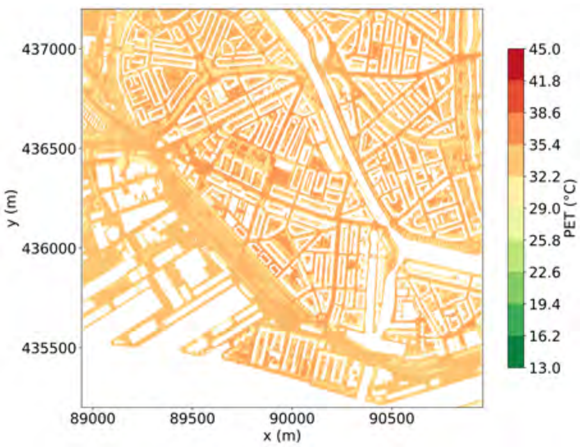
Figure J.2: The two dates for 2023



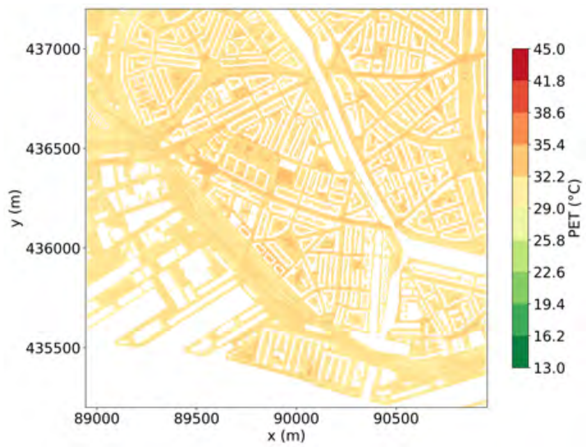
(a) 9:00



(b) 12:00

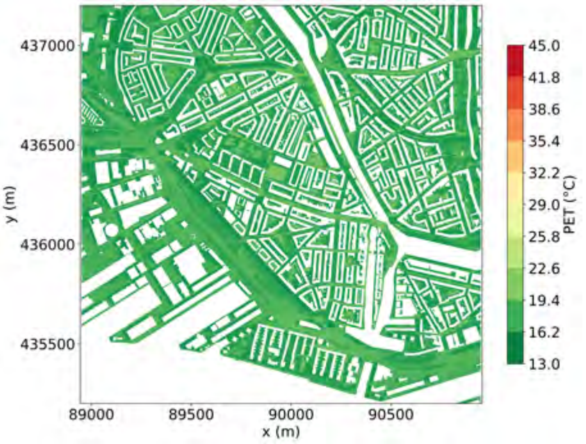


(c) 15:00

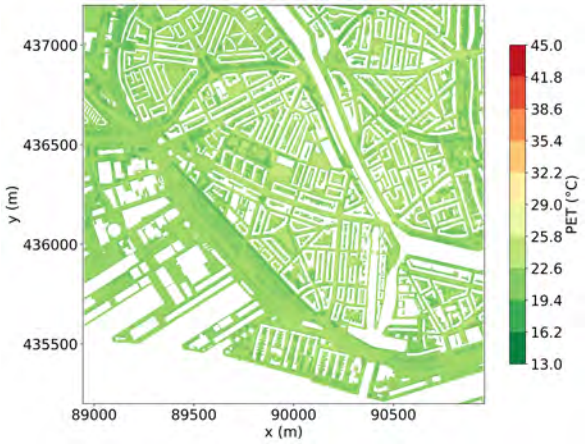


(d) 18:00

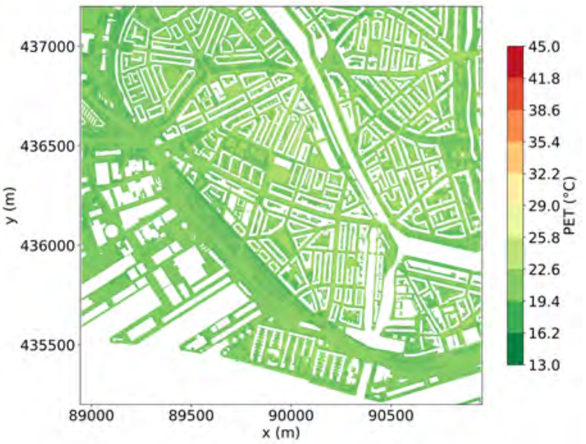
Figure J.3: Output files on research area 25th of Junest 2023.



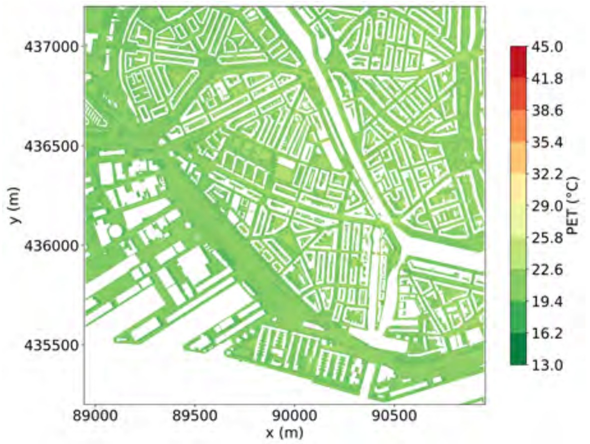
(a) 9:00



(b) 12:00



(c) 15:00



(d) 18:00

Figure J.4: Output files on research area 28th of Junest 2023.

K

Walkability analysis

Application network betweeness

To analyze the most frequently used routes, we will use angular choice analysis on the street network of Bospolder Tussendijken, which was generated by the tool developed by [Stavroulaki et al., 2019]. Angular choice analysis is a method used to identify the most commonly used paths based on their geometry. First, we need to normalize the data to highlight the importance of different routes and the urgency of using those paths. We will consider a distance of 500 meters as a neighborhood distance, which represents the distance an elderly person can walk within 15 minutes. For a regular person, a distance of 1000 meters will be considered, and for a biking distance of 15 minutes, a distance of 2500 meters will be used.

```
[language=SQL, caption={SQL statement for angular choice}, label=lst:case]
CASE
when ( "ac_500_norm" > "ac_1000_norm" AND "ac_500_norm" > "ac_2500_norm" ) then 'local'
when ( "ac_2500_norm" > "ac_500_norm" AND "ac_2500_norm" > "ac_1000_norm" ) then 'city'
when ( abs("ac_1000_norm" - "ac_2500_norm") <= 0.02 ) then 'intermediate'
when ("ac_500_norm" < 0.1 AND "ac_1000_norm" < 0.1 AND "ac_2500_norm" < 0.1) then 'irrelevantlocal'
else 'overig'
END
```

Determining the orientation of the streets

For determining the orientation of the streets the TOPNL [Kadaster, 2024] will be used. Next to this an excel table is linked to the names of the streets by a join by field attribute:

Listing K.1: SQL statement for orientation streets

```
CASE
WHEN "mainangle" >= 337.5 OR "mainangle" < 22.5 THEN 'North-South'
WHEN "mainangle" >= 22.5 AND "mainangle" < 67.5 THEN 'Northeast-Southwest'
WHEN "mainangle" >= 67.5 AND "mainangle" < 112.5 THEN 'East-West'
WHEN "mainangle" >= 112.5 AND "mainangle" < 157.5 THEN 'Northwest-Southeast'
WHEN "mainangle" >= 157.5 AND "mainangle" < 202.5 THEN 'North-South'
WHEN "mainangle" >= 202.5 AND "mainangle" < 247.5 THEN 'Northeast-Southwest'
WHEN "mainangle" >= 247.5 AND "mainangle" < 292.5 THEN 'East-West'
WHEN "mainangle" >= 292.5 AND "mainangle" < 337.5 THEN 'Northwest-Southeast'
ELSE NULL
END
```

By adding an additional table with the Height Width ratios of the streets there could be a determination if the solution ought to be sought in the public space or could be transformed by the architecture of buildings.

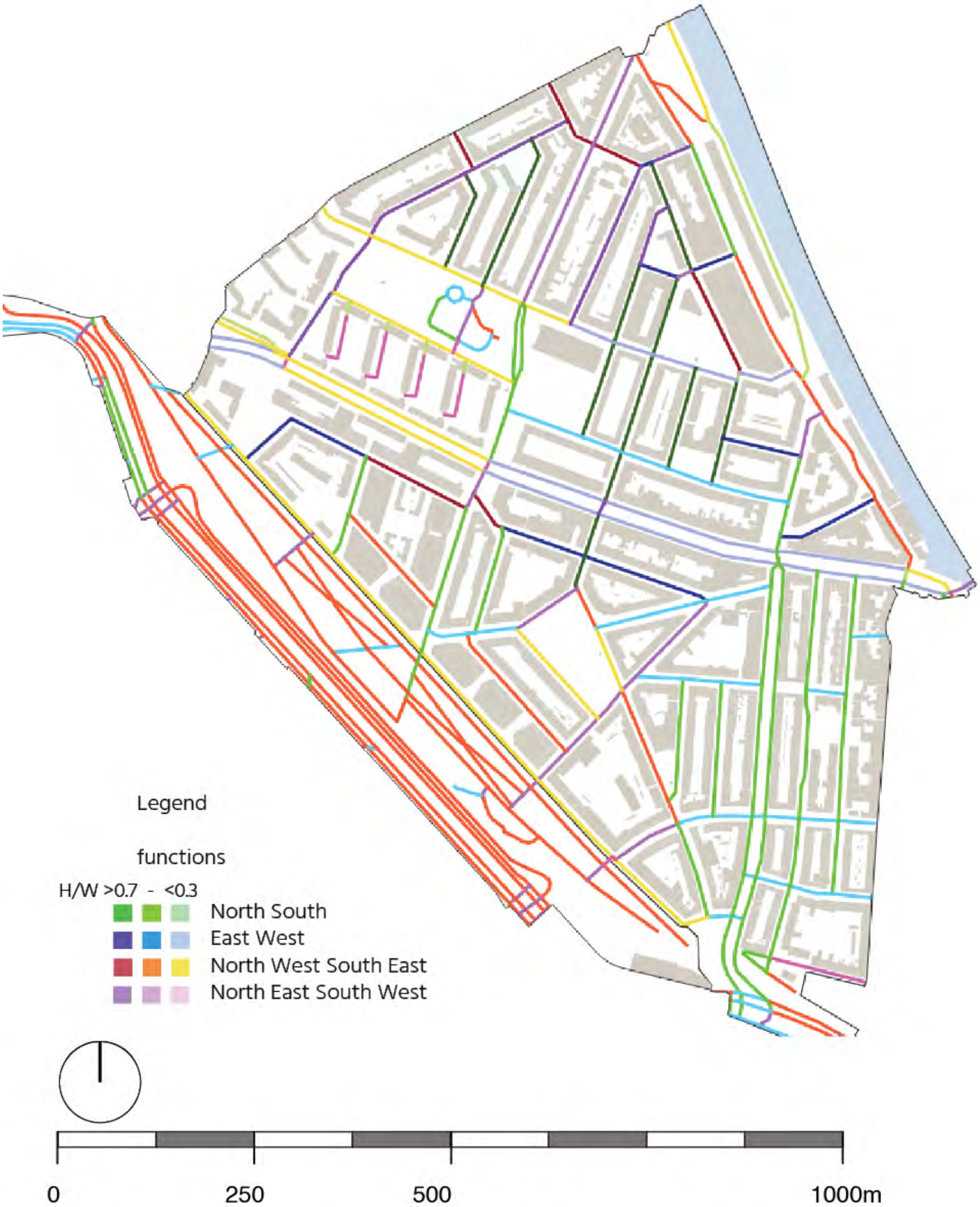


Figure K.1: Orientation map and H/W ratio buildings

Determining the attraction betweeness of certain locations in order to determine the streets to interfere in

As a guiding tool which routes are used the most based on dwellings and their destination points, the following procedure is set up to count the amount of shortest paths on line segments. The line segments network are from Dataset: Basic Topography Registration (BRT) TOPNL [Kadaster, 2024] . The set-up is as follows:

```
1 Bag dwellings create centroid points
2 QGIS network analysis shortest path for all the dwellings towards the
   preferred location
A. Explode lines
A1. Clean from A the multiple geometries > buffer 1m with 0.1 tolerance (
   buffer hull)
- Bufferhul create new attribute buffered \$id
3 Then A \& A1 intersect by location
- Virtual layer with bufferid from A1
- Virtual layer with buffer count how many times A is intersected in A1
- In python a table is created with how many times A is in A1 matching
   bufferid with count
4 Then link buffercount and bufferid to geometry A1.
```

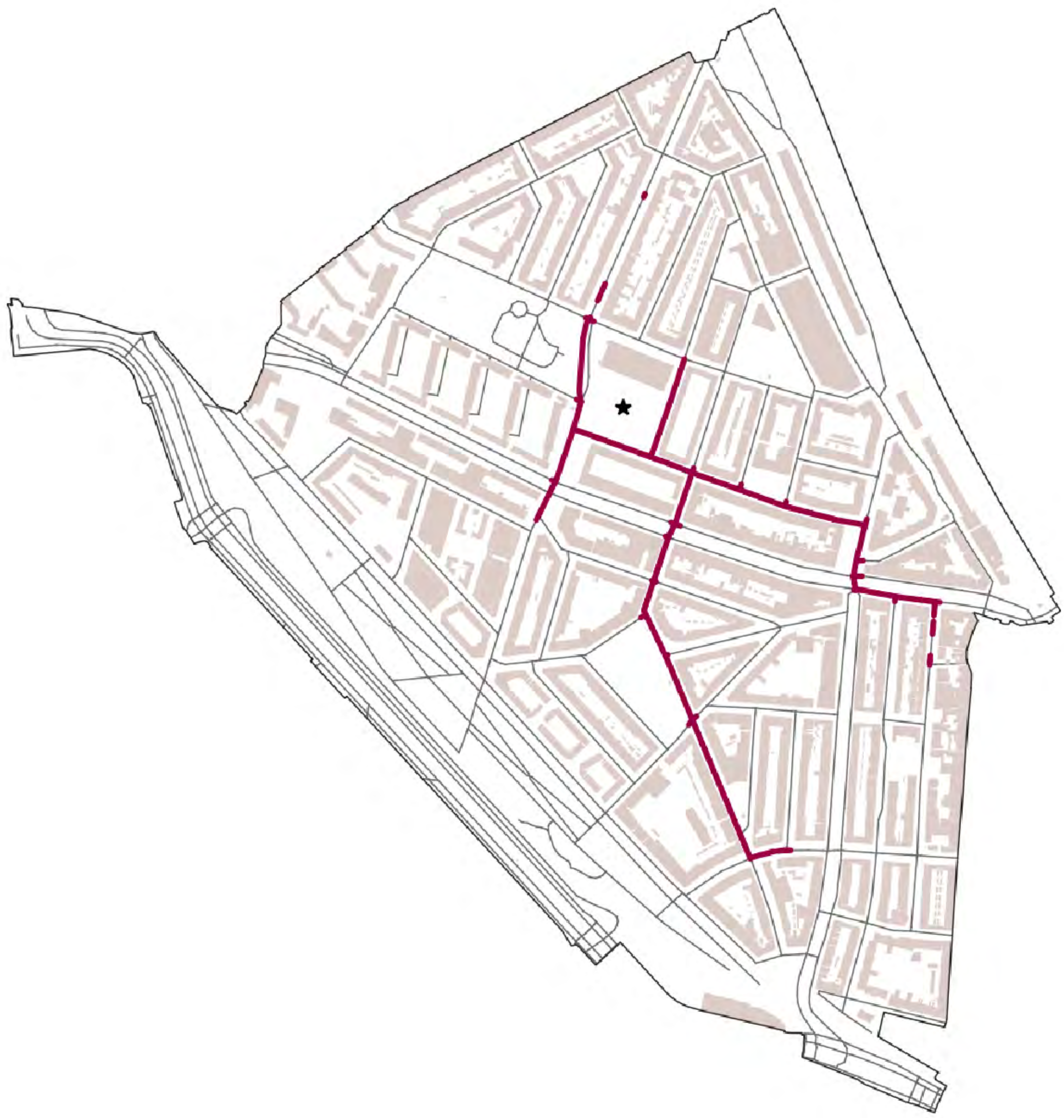


Figure K.2: Attraction betweeness market containing line segment pieces with more than 1000 dwellings as shortest path route