Conceptual design of an automation-facilitating tomato greenhouse concept

F.F. Bunnik

TU Delft RIDDER

Conceptual design of an automation-facilitating tomato greenhouse concept

by

F.F. Bunnik

Master's thesis

In partial fulfilment of the requirements for the degree of

Master of Science
in Mechanical Engineering

At the Department Maritime and Transport Technology of Faculty Mechanical, Maritime and Materials Engineering of Delft University of Technology
to be defended publicly on Monday January 16, 2023 at 13:00.

Student number:	4492315	
MSc track:	Multi-Machine Engineering	
Project duration:	February 15, 2022 - January 16, 2023	
Thesis committee:	Dr. ir. D.L. Schott,	TU Delft, Faculty 3mE, Section TEL, Chair
	Dr. ir. Y. Pang,	TU Delft, Faculty 3ME, Section TEL, Committee member
	Dr. X. Lin,	TU Delft, Faculty CiTG, Section T&P, Committee member
	Dr. ir. A.N.M. De Koning,	Ridder Growing Solutions, Additional expert member
	ir. W. Oltheten,	Ridder Growing Solutions, Additional expert member
Report number:	2022.MME.8741	

This thesis is confidential and cannot be made public until January 16, 2026.

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

TU Delft RIDDER

Preface

This thesis investigates a new tomato greenhouse concept, which facilitates robotisation. In this new concept the tomato plants are transported to the robots, instead of sending the robots into the greenhouse environment. I believe in the concept and think that it can become the new standard for growing greenhouse tomatoes. It was exciting to be part of the development of that new concept.

This thesis is written during an internship at Ridder Growing Solutions. It was the last assignment to complete the master's study in Mechanical Engineering, track Multi-Machine Engineering, at Delft University of Technology. I liked applying the theories learned on a complex practical problem. Furthermore, it was fun to work on a project about tomato greenhouses, because my first summer job was also in a tomato greenhouse.

During my thesis, several people helped me by sharing their knowledge and experience. This contributed to the quality of this thesis. I want to thank these people. Firstly, my company supervisors, Ad de Koning and Wouter Oltheten. They were always available for questions and helped me through all the available information on the greenhouse tomatoes. They also helped me to gain insight into technical problems and how to treat them scientifically. Because I was added to the innovations team, I got insights into what was going on in the company, which is also valuable.

Furthermore, I want to thank Yusong Pang, my supervisor from the TU Delft. His critical view from a bit more distance helped me look differently at a problem, which led to better results. I also want to thank Dingena Schott, for being critical and positive during the milestone meetings.

I hope you will enjoy reading my thesis as much as I did writing it.

Floris Bunnik
Delft, November 2022

Summary

Implementing automation can boost the efficiency and consistency of greenhouse operations and reduce the labour cost. For this thesis the crop of interest is the tomato. Therefore, the purpose of this project is to contribute to achieving a higher level of automation in tomato greenhouses. The conventional tomato greenhouse environment is jungle-like, which makes it difficult to operate robots. Research and development on these robotic solutions, for over a decade, hardly delivered commercially operational automation solutions for performing crop-handling tasks. Greenhouse solutions where the plants are transported to a central crop-handling area, in which the crop manipulation tasks are performed appear at a higher level of automation. For the cultivation of lettuce, a concept involving mobile plants is commercially operational. In this concept, the plants are transported on movable gutters, something that can potentially be done for greenhouse tomatoes as well. For lettuce, the level of 'conditional automation' has been achieved. At this level all actions are performed automatically by robots, whereas the monitoring and fallback mechanisms are an interaction between the robots and human operators (Baratam et al., 2022). The aim is to achieve at least a similar level of automation for tomato greenhouse by applying a similar movable-gutter concept.

To make tomato plants feasible for transport, they should be smaller than the 15 [m] of stem lengths in the conventional concept. A limited-truss concept is proposed, in which the plants only reach a length of up to 1.1 [m]. This concept was thoroughly researched in the 1990s. The major difference between the new concept and the conventional concept is that for the new concept the cultivation cycle is much shorter, up to 10 weeks vs 45 weeks, and fewer trusses are harvested per plant (two or three vs. 35). This limited-truss concept is not commercially operational yet; however, it is claimed to be profitable due to the increased truss density and the higher number of successive cultivation cycles in a year.

In this project a generic conceptual desgin framework for the limited-truss tomato on a movable-gutter concept is researched. The scientific knowledge gap addressed in this study concerns the design and evaluation of the new concept. To cover the knowledge gap the main research question is defined as *How to develop a generic conceptual design framework for a tomato greenhouse with the limited-truss tomato on a movable-gutter concept?* The conceptual design is defined as a floorplan with an area allocation, the capacity of handling tasks and the capacity of the transport system. Generic means in the context of this research that it is applicable to multiple cases with very different climate conditions. The main challenge regarding climate conditions is the variations in temperature and light intensity. Temperature is dominant for the ripeness process (i.e. the lead time), while the photosynthetically active radiation (PAR) spectrum of light leads to fruit growth (i.e. the increase in fruit weight). To obtain trusses of constant weight, these two variables should be balanced.

To answer the main research question, a generic design framework is developed. This framework is schematically in Figure 1. There are three modules that together form one simulation model. The first module is based on the existing model of the De Koning (1994). In this module, the growth of the crops is modelled, which is used to calculate physical requirements lead time and required space of a plant. The second module is a mixed-integer linear programming (MILP) optimisation model that generates a conceptual design and sowing strategy. The third module simulates the generated design and corresponding strategy for a scenario of choice. The object-oriented discrete event simulation (OO-DES) protocol is used for this simulation. The implementation is done in Python, with making use of the Salabim package. The next step is evaluating the conceptual design and possibly improve it. This manual iterative evaluation process follows the protocol in Figure 2.

Implementation of the new concept led to a generic, all-encompassing design framework. It is proven that the developed framework is generic by employing the design framework on case studies with very different climate conditions, which are Rotterdam, Darwin, and Melbourne. The design

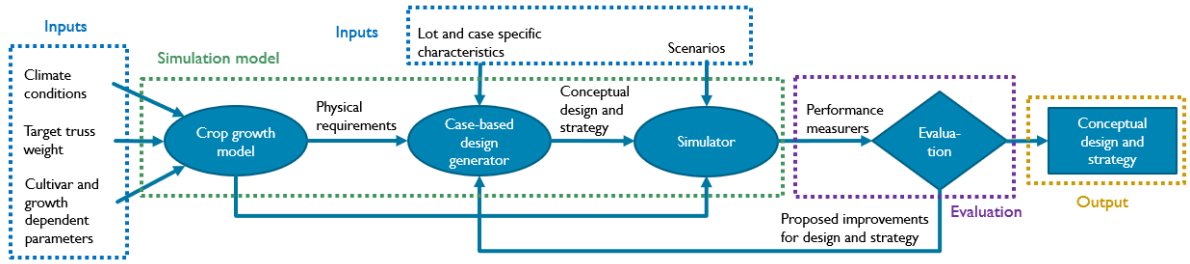


Figure 1: Developed design framework to obtain the conceptual design and corresponding sowing strategy

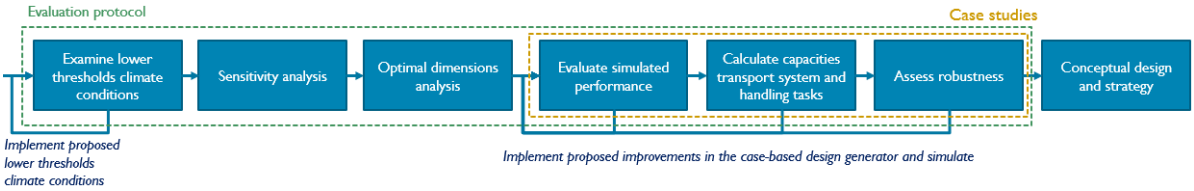


Figure 2: Protocol for the evaluation of a conceptual design and sowing strategy

framework performed well for all these cases, meaning it is generic. Therefore, the aim of filling the scientific knowledge gap – regarding the generation and evaluation of designs for the new tomato-greenhouse concept – is achieved. The applied model-based simulation in the framework is capable to explore different design solutions and to identify bad design choices before significant resources are spent. Moreover, proposed improvements of the design following from the evaluation protocol can directly be implemented, simulated and evaluated.

The output designs and corresponding strategies developed by employing the framework and its evaluation protocol for different cases show a high production compared to the conventional system. In addition to this competitive production, the new concept contains other elements which makes it interesting. The concept facilitates automation, but even if no automation is implemented the new concept will gain the operational efficiency. In addition, the multiple successive cycles per year reduce the severeness of illnesses. If a plant becomes ill, there is less loss in a short cultivation cycle than for a long cultivation cycle. An advantage of the new concept is also the year-round production, which also enables more continuous operations. Moreover, it makes very efficient use of resources, which makes the new concept more sustainable than the conventional one.

A first recommendation for a follow up study on the developed framework is to review the assumptions made in greater depth. An assumption is that the existing crop-growth model of De Koning (1994) still holds for the new cultivation process. The exact relationships for fruit growth and development with the climate conditions for different growing stages should be researched. Also, the definition of growing stages should also be substantiated in greater depth. Furthermore, the simplified parts, assuming them to be constant (e.g. LUE or PAR Ratio) or linear (e.g. crop development), can be replaced by time-series or fundamental exponential equations. The second recommendation is taking the necessary step of the preliminary and detailed design of subsystems as the transport system and components as the movable-gutter itself. The third recommendation is regarding the validation and real-world experiments. Since the system researched in this study does not exist, a thorough validation was impossible, let alone real-world experiments. The next step would be to develop a small-scale test setup to undertake the validation and experiments. Furthermore, the concept should be reviewed in a broader sense. A cost analysis should be conducted. In addition to that, a study on the quantification of the energy usage of the new concept could be conducted, which could be used to examine its sustainability quantitatively.

List of Figures

List of Tables

Contents

1

Introduction

1.1. Background

Automation can increase the efficiency and consistency of operations in horticultural greenhouses (Baratam et al., 2022). In addition, automation reduces the amount of human labour needed, leading to operational cost savings. Therefore, automation is a topic of high interest for the greenhouse sector. Ridder Growing Solutions is a company that offers technical solutions for greenhouses. The company aims for scalable automation solutions to save energy and improve efficiency. This research project was conducted in cooperation with Ridder Growing Solutions.

Although the tomato is the most cultivated crop in greenhouses in the Netherlands (CBS, 2017), the hypothesis that automation is lagging in respect to other crops is affirmed in a literature study (Delft University of Technology report number 2022.MME.8597). There are hardly automation solutions operational in tomato greenhouses. An important reason for this lack is the jungle-like environment. This environment is caused by the high-wire growing concept (Figure 1.1). In this concept, tomato crops are grown along a wire, grow partly horizontally and reach stem lengths of up to 15 [m]. Due to this concept, there are many stems and leafs at a small area (see Figure 1.2). Therefore, trusses are often (partly) behind a leaf or a stem, which makes them hard to identify by a robot without touching the leafs and stems. Even if the trusses were identified, it would be too hard for a robotic arm to approach them without damaging the plants. Therefore, in this project, the jungle-like environment is considered too challenging to achieve a level of automation higher than Level 2 (see Table 1.1), shortly.

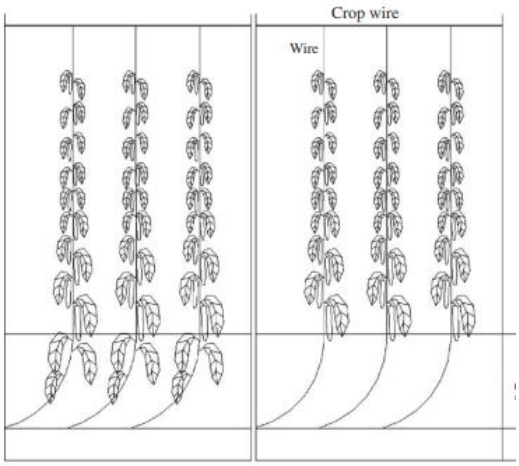


Figure 1.1: High-wire cultivation concept (Van Henten et al., 2006)

According to Baratam et al. (2022) there are four aspects regarding automation: 1) Logistics, 2) Crop Manipulation, 3) Crop Growing, and 4) Greenhouse Strategy. Automation is particularly lacking for logistics and crop manipulation. The specific requirements to achieve a level of automation for the individual aspects can be found in Baratam et al. (2022). To reach level 3 of automation for crop manipulation, the system must be capable of handling all operations, whereas the monitoring and fallback mechanism involve a combination of human operators and an automated system. Although different companies claim their solutions can achieve a high level of automation, the success rate in actual operations appears too low to achieve that level.

Table 1.1: Overview of the different stages of automation (Baratam et al., 2022)

Level of Automation	Name	Definition	Individual automation levels
0	Basic Greenhouse	Rudimentary	All aspects level 0
1	Technology Assisted Greenhouse	Grower receives assistance from AI/robots but is involved in operations all the time	At least two aspects are level 1
2	Partially Automated Greenhouse	Grower can take 'hands off' of operations in a specific (set of) greenhouse aspects	At least two aspects are level 2, others are level 1
3	Conditionally Automated Greenhouse	Grower can take 'eyes off' (i.e. need not actively monitor over a (set of) aspects)	At least two aspects are level 3, others are level 2
4	Highly Autonomous Greenhouse	Grower can take 'brains off' (i.e. expect AI to take over a [set of] aspects)	At least two aspects are level 4, others are level 3
5	Fully Autonomous Greenhouse	Grower is only involved in target-setting, and AI takes over the greenhouse operations	All aspects are automated to level 4

Higher levels of automation can be achieved for cropping systems optimised for robots, particularly movable-crop systems, in which crops are transported to a central crop-handling area. The central crop-handling area is fully conditioned to the needs of the robots, which enables them to run at maximum performance. Operational examples of this system include the cultivation of orchids (Ter Laak Orchids, 2019) and lettuce (FromBoer, 2019). At FromBoer lettuce is cultivated in gutters (Figure 1.3). This type of cropping system is considered most applicable to facilitate automation in greenhouse tomato growing. This is considered, because in the conventional system the crops are often grown in gutters as well. A boundary condition of this system is that the crops must have a limited length, otherwise they are not feasible to transport. Therefore, a new tomato-cultivation concept is proposed, called 'limited-truss' concept. This method of growing has already been thoroughly investigated, and it is claimed to be economical viable (Giacomelli et al., 1994 and Logendra et al., 2001). In this limited-truss concept, the crops reach a length of up to 1.1 [m], instead of 15 [m] in the conventional, high-wire, concept. Another important difference with high-wire concept is that for the limited-truss concept the cultivation cycle is short, up to 10 weeks, rather than 45 weeks, and fewer trusses are harvested per plant (two or three vs. 35). The crops in the limited-truss concept are cultivated with a high plant density and in multiple successive cycles in a single year to achieve a production which is competitive with the high-wire concept.

Figure 1.2: The jungle-like environment in greenhouses (Boulard et al., 2017)

1.2. Problem definition

In this research project, the combination of the limited-truss concept en the movable-crop concept is explored. This combination is called the limited-truss tomato on a movable-gutter concept. This new concept is marked as a concept that facilitates automation due to the characteristic of performing

crop-handling tasks in a central area. For this new concept, both how to design and how to evaluate a design are unknown. The generation and evaluation of designs is the scientific knowledge gap this study addresses. This research delivers a generic framework in which conceptual designs with corresponding sowing strategies can be generated and evaluated. This generic framework did not exist for this new cultivation concept. Moreover, for the greenhouse sector, no such framework was found. Logiqs (2021) has a pilot project under development for automated greenhouse design for crop growing on benches. However, that project focuses on structural greenhouse design rather than the system design, which this current project considers.

The main design challenge is the seasonal variation in temperature and light intensity. Temperature is dominant for the ripeness process (i.e. lead time), while the photosynthetically active radiation (PAR) spectrum of light leads to fruit growth (i.e. increase in fruit weight). To obtain trusses of constant weight, these two variables should be balanced. This balancing issue makes it complex to reach an optimum for design and strategy. The variation of temperature led to a varying lead time for Rotterdam of 52 to 75 days, while the inside light intensity of the PAR spectrum varies with a factor three. To cope with this challenge, model-based simulation is used for design. An advantage of model-based simulation, is that multiple design options can be explored and erroneous design choices can be eliminated before significant resources are spent.

Figure 1.3: Movable gutters for lettuce (GroenteKennisnet, n.d.)

1.3. Research questions

To cover the scientific knowledge gap defined in the previous section, the next research questions are formulated.

Main research question:

How to develop a generic conceptual design framework for a tomato greenhouse with the limited-truss tomato on a movable-gutter concept?

Sub-questions:

- SQ1: What could be a viable path towards automation in tomato greenhouses?
- SQ2: What modelling methodologies can be used to generate and simulate conceptual designs?
- SQ3: How should the new concept be implemented in a simulation model?
- SQ4: How can the conceptual design be evaluated, and the simulated performance of the conceptual design be improved?

1.4. Scope

The main output of this research project is a generic framework that generates and evaluates designs. Since the framework needs to be generic, the designs should be generated automatically. Therefore, a model was used to generate these designs. This study limits itself to the conceptual design phase. For this study, a conceptual design is defined as a floorplan with an area allocation, required capacities for the transport system, and a sample selection of equipment. The sowing strategy provides a sowing rate optimised for achieving a maximum turnover. In this conceptual design and sowing strategy, a set of aspects of greenhouse operations is incorporated. However, there are also aspects beyond the

scope of this project. A schematic overview is provided in Figure 1.4.

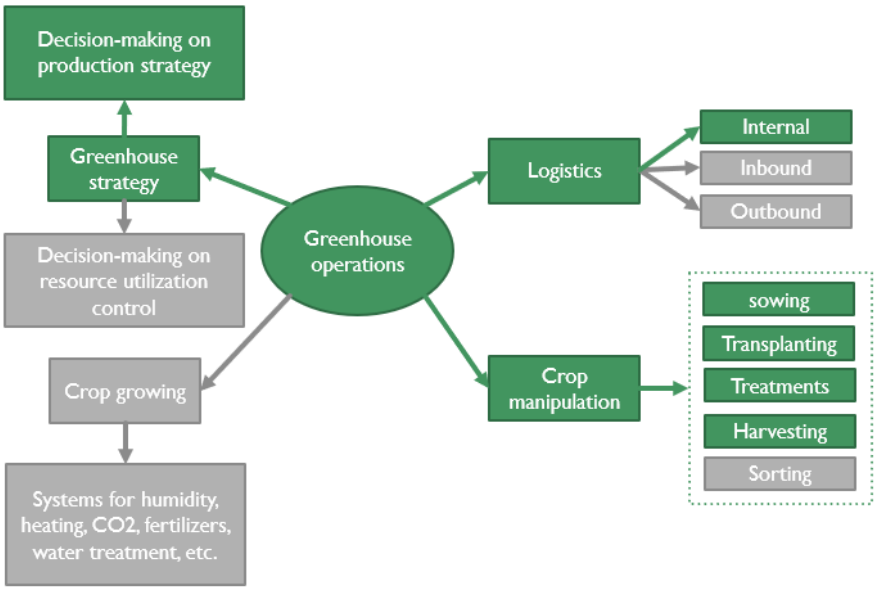


Figure 1.4: Aspects of greenhouse operations (In accordance with Baratam et al. (2022)): grey is out of scope, the green aspects are incorporated in the study

The framework developed contains a simulation model and an evaluation protocol. The simulation model was developed by using the spiral workflow. In this spiral workflow, the model starts simple, and complexities are added step by step. The simulation model should be all-encompassing. Therefore, it was accepted that assumptions and simplifications were made. Because the system is highly complex, the final delivered simulation model at the end of this study might still contain simplifications or assumptions. However, it was required that the simulation model is all-encompassing and delivers outputs that are reasonably accurate. These accurate outputs were needed for the evaluation step.

In the evaluation, first a sensitivity analysis and optimal dimension analysis was conducted to observe trends. For these two steps general statements on sensitivity and dimensions were made. For the next evaluation steps, three case studies was created. These case study serves as proof that the developed framework is generic. Moreover, they can be used as an example for how the next evaluation steps should be taken and how improvements to the conceptual design and sowing strategy can be made. Once these evaluation steps were passed, the conceptual design and strategy for the case study was presented.

The simulation model developed in this study was verified. The possibilities for validating the model were limited, as there are no real systems or test setups available. Therefore, a thorough validation and test experiments were beyond the scope of this research project. The developed framework was developed for the design phase. However, main parts apply to the operational phase as well. In the framework, for a fixed design, there could be simulated for different inputs or thresholds. Furthermore, this study focuses on the technical aspects. Therefore aspects as a cost analysis are beyond the scope of this project.

1.5. Structure

The sub-questions were answered chronologically. Answering these questions contributed to building the design framework and guided to an answer to the main research question. This led to the report structure in Figure 1.5.

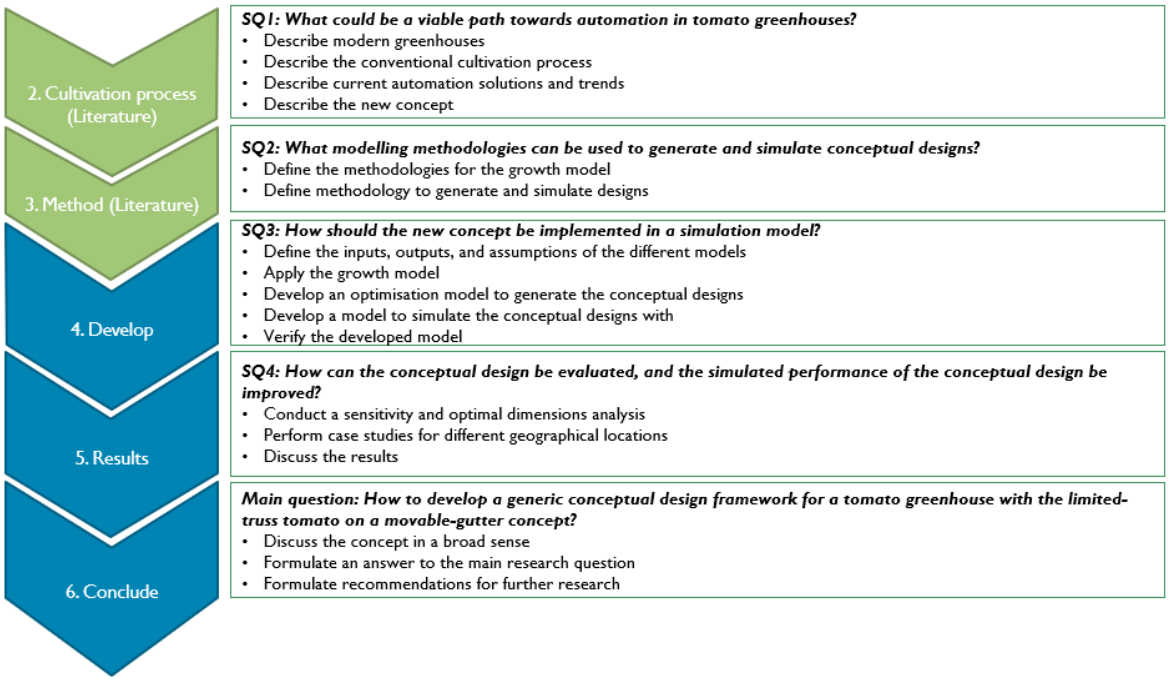


Figure 1.5: Structure of the study, parts in light green are mainly literature study

2

The viable path towards automation in tomato greenhouses

This Chapter starts with describing state-of-the-art technologies for the cultivation of greenhouse tomatoes. First a brief introduction to modern greenhouses (Section 2.1) is provided. The conventional cultivation process is then discussed (Section 2.2), followed by the trends in greenhouse automation (Section 2.3). Based on the existing state-of-the art solutions and trends, a path towards further automation in tomato greenhouses is proposed (Section 2.4). This proposed path helps to answer, in Section 2.5, the first sub-question: *What could be a viable path towards automation in tomato greenhouses?*

2.1. State-of-the-art greenhouses

Greenhouse designs have different shapes (see Figure 2.2). The even-span greenhouse concept is often the best design choice. An important characteristic of the conventional tomato greenhouse is the high-wire cultivation concept (explained later with Figure 2.5). These high-wire supporting systems keep the top of the plants typically 2.25 [m] above the cultivation medium. This does not mean the plant length is limited to this length; they can reach a length of 15 [m]. These high-wire systems come in rows, typically with a row width of 1.25 [m] and a row distance of 1.6 [m]. There is a heating tube between the different rows, on which equipment can travel (see Figure 2.1). (Heuvelink and Dorais, 2005)

Figure 2.1: Picture of a tomato greenhouse (Boulard et al., 2017)

Two important greenhouse design choices are the roof shape and the aspect ratio. Many studies have been conducted to select the optimum choices (e.g. Achour et al., 2021, Boulard et al., 2017 & Valera et al., 2017). There is a consensus that the optimal design depends on the evaluation criteria and the location. If, for example, heat is the dominant criterion, the design will differ from a case in which relative humidity is the principal consideration (Achour et al., 2021).

As heat is crucial in greenhouses, this aspect has been investigated thoroughly. Mostafavi and Resaei (2019) investigated all five types of greenhouses (Figure 2.2). The main conclusion of their research is that the even-span concept is thermally often the most efficient choice. Therefore, the even-span, Venlo-type, greenhouse is used as starting point for this study. Furthermore, the thermal

load depends highly on the greenhouse covers. For example, a single layer of polycarbonate requires a 65% higher thermal load than two-layer glass (Mostafavi and Resaei, 2019). Smart choices regarding the material can reduce the total cost of a greenhouse operation. There is usually a trade-off between investment cost and operational cost. The exact design will always depend on the exact application, location, and desires of the investor (Vanthoor et al., 2011). The optimal greenhouse for a specific project remains the challenge of the designer.

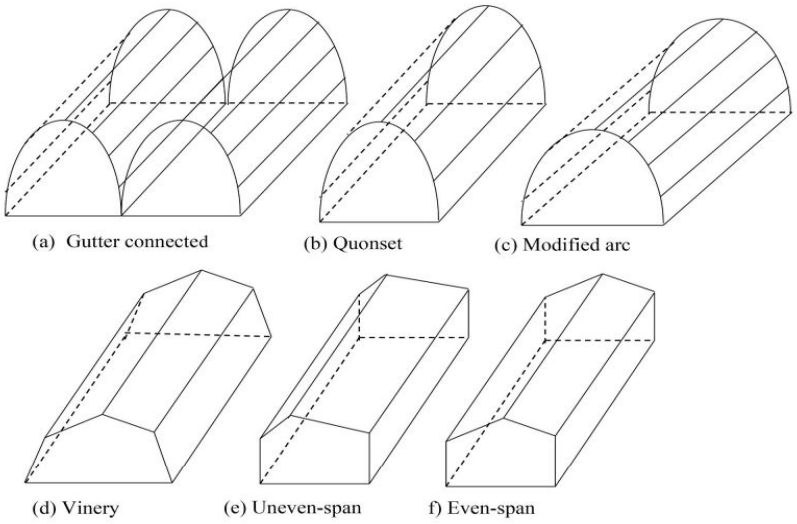
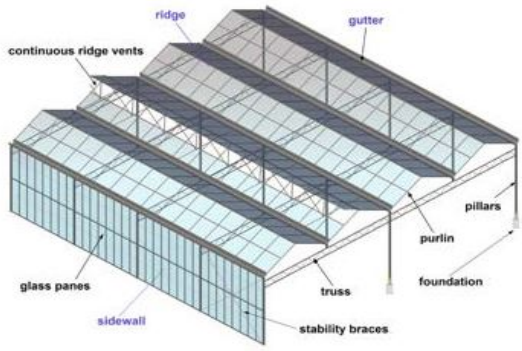
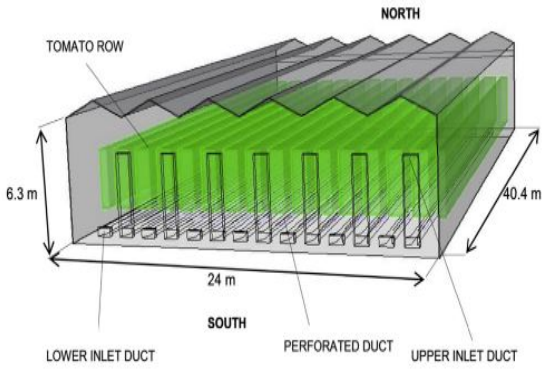


Figure 2.2: Drawing of different greenhouse shapes (Achour et al., 2021)



(a) Drawing of an even-span greenhouse (Valera et al., 2017)



(b) Model of an even-span greenhouse for tomatoes. Dimensions are just an example (Boulard et al., 2017)

Figure 2.3: Figures of an even-span greenhouses

2.2. Conventional cultivation process

The full conventional cultivation process is elaborately described in Dieleman et al. (2009), Van Os et al. (2020), and Heuvelink and Dorais (2005). A schematic overview of the process is provided in Figure 2.5. In this section, the cultivation process is treated by reviewing all the tasks that must be performed in a conventional tomato greenhouse. The beginning of the cultivation process is sowing. The cultivation medium for the beginning of this process is Rockwool in a tray. The seeds are placed in a tray with a high plant density, where they remain during the first stage of the growing process. One advantage of cultivation in Rockwool compared with bare soil is that nutrients and water do not flow away into the ground.

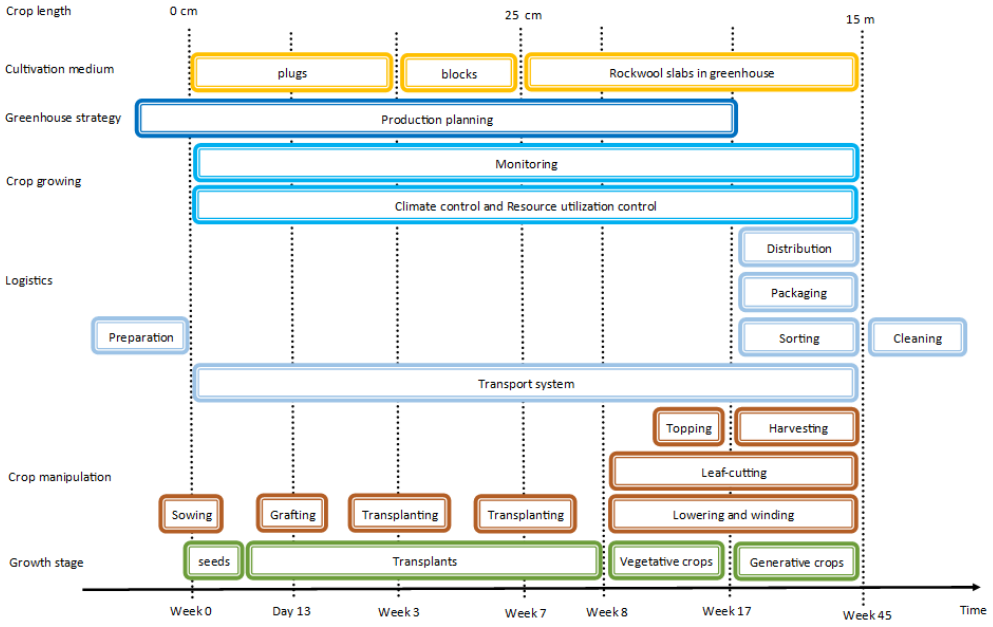


Figure 2.4: Schematic of the conventional cultivation process, based on Dieleman et al. (2009)

After sowing, when the stem reaches a thickness of 2 [mm], on about Day 13, the grafting task is performed. The principle of grafting is taking the top part of a stem of one plant and placing it on the bottom part of another plant. This process combines the favourable properties of a specific cultivar with the strong root system of another cultivar. The small plants need to be transplanted. The first transplanting task is from a high plant-density tray with plugs to blocks with lower plant density. After seven weeks, the small tomato plants of approximately 25 [cm] are placed in the greenhouse in slabs of Rockwool. (Dieleman et al., 2009)

The small plants need to be transplanted. The first transplanting task is from a high plant density tray with plugs to blocks with lower plant density. After seven weeks, the small tomato plants of approximately 25 [cm] are placed in the greenhouse in slabs of Rockwool. Now, the vegetative growth process begins. The flowers are pollinated, and the plant is led along the wires in a process called winding. The wire is wound around the top of the plant. This concept is called the 'high-wire' concept. The other associated task here is lowering, also called layering. Lowering is performed for high-wire crops, such as the greenhouse tomato. If the crop reaches the top of the wire, the crop is lowered circa 25 [cm] by unwinding with spare wire. This spare wire is connected to the crop wire by a coil that hangs on the crop wire. In addition, the crop wire (Figure 2.5) is moved to the right, so the crop continues growing without reaching an extreme height. The plants can reach a length of up to 15 [m], whereas the distance from the cultivation medium to the crop wire is about 2.25 [m]. (Heuvelink and Dorais, 2005)

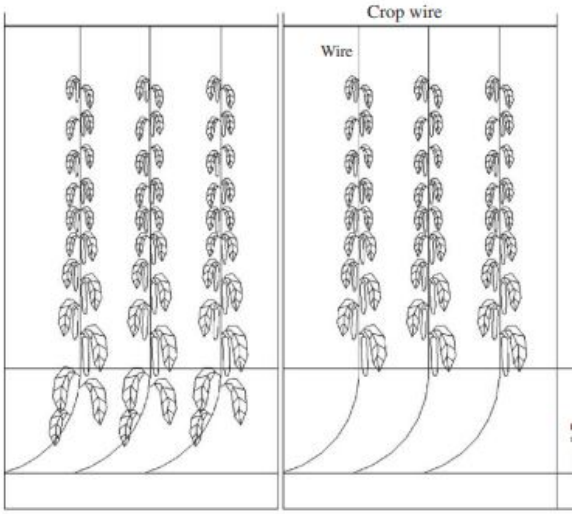


Figure 2.5: Illustration of the high-wire concept (Van Henten et al., 2006)

The dominant pruning task that is required for the greenhouse tomato is de-leafing. The leaves are cut to maximise the amount of light that reaches the fruits to ripen them. The other pruning task is topping. When the daylength is decreasing, the number of stems must be reduced by removing the additional side shoots.

From Week 17 on, the harvest season starts, which can last 28 weeks. In the harvest season, the ripe fruits are collected from the tomato plants. After the harvest, the products are sorted, packaged, and distributed to the buyers. When the harvest season ends, the plants are removed, and the greenhouse is emptied. This process leads to substantial waste (Figure 2.6). Nutrients remain in the substrate, and are therefore lost. After the greenhouse is cleaned, it is prepared for the next cultivation cycle.

In Northwest Europe and North America, the cultivation process is relatively efficient. To achieve a high efficiency, highly skilled human operators are required. This type of labour is scarce in these areas. The skills required mainly involve monitoring and climate and resource utilisation control. In particular, the levels of of artificial light and temperature are of strong influence on the growing process (Verheul et al., 2022). Furthermore, the management of the resources water, fertilisers and nutrients affecting the quality and quantity of the tomatoes produced.

Figure 2.6: Waste slabs of Rockwool. Picture by De Koning, A.N.M.

Due to the favourable conditions in the Western greenhouses, over 35 trusses can be harvested per plant. Thus productivity can exceed 500 t/ha, and even go up to 700 t/ha (Heuvelink and Dorais, 2005). The actual production level depends on the cultivar and the (supplementary artificial) light intensity, with the boundary condition that there is enough heat, nutrients, water and CO2. Verheul et al. (2012) investigated possible production for Norwegian greenhouses. The total yield potential even increased to 1250 t/ha by adding (much) artificial light. The two leading variables that should be optimised for a high production are light and heat. However, light and heat should be balanced and are not independent from other climatological variables. For example, light use efficiency (LUE: the accomplished growth in grams for an MJ of PAR light) depends not only on the light intensity, but also on humidity and CO2 concentration (Heuvelink and Dorais, 2005).

2.3. Market trends in greenhouse operations

The first observed trend is that there is growing demand for locally produced food, a trend accelerated by the COVID-19 crisis (Pedersen and Hansson, 2021). Undoubtedly, this trend will be strengthened due to the cut in wheat supply in 2022 because of the war in Ukraine (Carriquiry et al., 2022). Cultivation in greenhouses can play an important role in producing food locally. Different places worldwide have different demands regarding greenhouse operations. Emerging countries often lack in-depth knowledge about crop growing, as highly skilled crop growers are in great demand. According to Ridder Growing Solutions, the output of a greenhouse is highly sensitive to the quality of the grower. Automation can support the grower and take over some tasks, which reduces this sensitivity to the quality of the grower. Therefore, automation can lead to improved harvest results in places where knowledge about efficient greenhouse crop-growing is lacking.

In the Netherlands, as an example of a developed country, there is great knowledge about greenhouses and crop growing. Highly experienced growers can compete with the results of automation technology (e.g. Hemming et al., 2019 and Hemming et al., 2020). However, technology can still help to enhance production and efficiency. Sensors and smart systems can, for example, be a great help in determining the most efficient opening of windows. In the Netherlands, contemporary, the main

function of automation technology is enhancing the efficiency of crop growing and enabling growers to manage larger greenhouses than would be possible without the technology.

One challenge in the Netherlands is that lower-skilled human labour is becoming scarcer and more expensive (Groentennieuws.nl, 2021). The lack of this labour is a threat for the consistency of the greenhouse operations. More automation by robots performing crop-handling task contribute to ensuring the consistency of the process. Ensuring consistency means fewer risks for investors in greenhouses. This is important, because more investors without knowledge about greenhouses are entering the market (Gersdorf, 2022). The entry into the market of these investors implies that the higher investment costs of highly automated systems are less a problem if they lead to lower total costs in the longer term.

One risk for investors is the gas crisis that has greatly impacted North-West Europe, including the Netherlands. Greenhouses require light and heat, and in the colder months, artificial lighting and supplementary heat are needed. Energy has become very costly, but to produce locally, extra energy must be supplied to greenhouses. However, there is no consensus whether exploding energy costs in North-West Europe will be temporary or remain locally it is safe to say that the most profitable future greenhouses will be the most energy efficient greenhouses. This statement is supported by the increasing demand for sustainable production. Automation technologies will ensure energy is used as efficiently as possible. Therefore, there is an increasing drive towards greater automation.

2.4. The new concept

The previous three sections emphasised that greenhouse cultivation is a high-tech area and automation is a topic of high-interest. Many types of solutions have entered the market. These solutions mainly focus on the crop growing aspect and increasing the yield. Although the list of processes that are optimised and automated is expanding, there remains hardly any automation operational in crop manipulation and logistics. The crop-handling tasks require most human labour, while human labour is the largest share of the variable costs (Testa et al., 2014). Therefore, automating these processes will lead to the largest cost savings.

In the literature study (Delft University of Technology report number 2022.MME.8597), there are two common paths identified towards further automation in the greenhouse sector. The first path is designing robotic solutions that can operate in conventional modern greenhouse concepts. The second approach is creating an environment that fully caters to robots. Both paths have seen investment in recent decades.

A result of the first path is the de-leafing robot of Priva b.v. (The Netherlands). Zeelen submitted a patent for this robot in 2008. This robot is the state-of-the-art pruning robot for high-wire crops. Priva b.v. claims the machine can cut leaves with an accuracy of 85% up to 1 ha per week in 24/7 operation. This is a great advance in automation, substantially reducing the human labour needed. However, significant post-treatment is still needed by human labour. Moreover, pruning the last 15% of leafs requires substantially more than 15% of the total work for the task. Priva b.v. expects that, in the future, all the tasks in the greenhouse will be performed by a mesh of robots. At that stage, at least Automation Level 3 should be reached, with hardly any post-treatment needed, and the monitoring and fallback will be a combination of human operators and the system. The grower will be able to take their ‘eyes off’ (Baratam et al., 2022). Nevertheless, although 14 years have been spent working on the robot, it is still not commercially operational. Moreover, with the current specifications, the robot is at Automation Level 2 due to the required post-treatment by a human operator. Therefore, it is not expected that the desired mesh of operating robots will be achieved soon.

The other approach towards automation is to transport the crops to the robots via an automated transport system. This movable-crop concept facilitates automation, as the working environment can be adjusted to robots when performing them in a central crop-handling area. Tasks in a central crop handling area appear at a higher level of automation than solutions that operate in the conventional

greenhouse. Therefore it is expected that the movable-crop concept is a viable path towards a higher level of automation. (Delft University of Technology report number 2022.MME.8597)

This concept was patented in 2015 by Deforche and Deforche. A similar concept is operational for the cultivation of lettuce (FromBoer, 2019). At FromBoer lettuce is cultivated in gutters (Figure 1.3). A similar cropping system is considered most applicable to facilitate the automation of greenhouse tomato growing. A boundary condition of this system is that the crops remain at a limited length, otherwise they are not feasible to transport. Therefore, the recommendation from the literature (Delft University of Technology report number 2022.MME.8597) is to investigate a movable-crop concept with smaller tomato plants. This cultivation approach is called the limited-truss concept, which has already been thoroughly investigated and is claimed to be economically viable (Logendra et al., 2001, Giacomelli et al., 1994). In this concept, crops can grow up to a length of 1.1 [m]. These crops are cultivated with a high plant density and in multiple successive cycles in a year to achieve competitive production. For the limited-truss tomato, a new gutter can be designed to fit existing movable-gutter systems. Such systems have proven in practice that they can reach up to Automation Level 3 for other crops.

2.4.1. The limited-truss concept

A schematic of the proposed limited-truss concept is in Figure 2.7. The new process was investigated by Giacomelli et al. (1994) and Logendra et al. (2001). The five growing stages a crop undergoes are named, and the duration is indicated for a greenhouse in the Netherlands. The plant density is also indicated, as well as which tasks must be performed when. Not all crop-manipulation tasks need to be performed for the limited-truss concept. First, lowering and winding are no longer needed. The grafting task is also superfluous, as the advantage of using another root system is less relevant due to the short cycle of this concept. In addition, hardly any pruning is necessary, which makes the limited-truss concept less complex regarding crop-handling tasks.

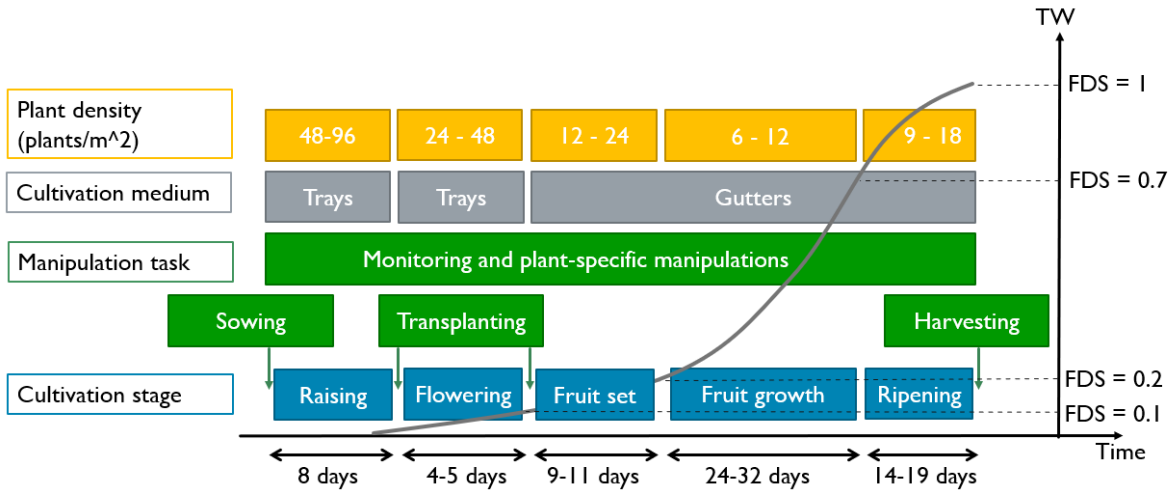


Figure 2.7: Conceptual representation of the new cultivation proposal. Note that the plant density is only an indication. TW stand for Truss Weight, FDS means Fruit Development State.

The varying lead times per growing stage are a major challenge for cultivation planning, as it is desirable to maximise the occupancy rate in the greenhouse. The stages with the largest variations are those stages with the lowest plant density. A plant contains a number of trusses. Because the energy (light and heat) is converted to truss growth and development of the fruits, the relevant variable for calculations for growth stage three to five is the truss density, not the plant density. Changing a plant density is one of the variables that available to adjust the truss density. However, there is a plant density is indicated in Figure 2.7, because in the first two growing stages there are no trusses yet.

The variation in cycle duration concerns the variations in climate conditions, mainly global radiation and temperature. For this research, it is assumed that the influence of variables other than temperature and light is negligible. This assumption is true for the condition that variables as humidity and CO2 concentration are within a desired interval. In a greenhouse, the temperature and the light intensity can be manipulated. Normally, however, no constant numbers for temperature and light are maintained. During summer, nature provides a high temperature and light intensity. To achieve the same conditions during winter leads to unfeasibly high cost. Only for the growth stages with short lead times and high plant densities is it could be feasible to maintain constant conditions.

The main characteristics of the new proposed concept vs. the conventional concept are compared in Table 2.1. The most striking aspect regarding the cultivation process is the cycle duration. The current commercial tomato-cultivation cycle is about 45 weeks. In these 45 weeks, 35 trusses are harvested per plant. The new proposed cultivation process only has up to three trusses per plant. In the study of Logendra et al. (2001), they experimented with one to three trusses per crop, depending on the use case for how many trusses are optimal. More trusses result in bigger plants and longer harvest times. Two or three trusses seem efficient numbers.

Table 2.1: Main characteristics of the new and conventional tomato greenhouse concepts

Feature	Conventional concept	New concept
Cycle time (for the Netherlands)	45 weeks	49-74 days
Maximum length of crops	15 [m] (high-wire concept)	1.1 [m] (limited-truss concept)
Number of growing stages present in greenhouse	1	5
Number of compartments with different climate conditions	1	5
Number of different cultivars present	1	Free choice (i.e. different cultivars in parallel batches)
Demand or price driven strategy	No	By a continuous flow of parallel batches
Greenhouse type	Even-span Venlo-type greenhouse	Even-span Venlo type greenhouse
Cultivation medium	Rockwool in fixed gutters	Hydroponics movable gutters
Level of automation according to standard framework (Baratam et al., 2022)	Up to Level 2	Potentially Level 3

2.4.2. The movable-gutter concept

In this subsection, the concept of a movable-gutter concept for tomato greenhouses is discussed. Starting with the global drawing of the greenhouse and its crop-handling area in Figure 2.8, Number 3 indicates the crop-handling area, whereas the greenhouse is indicated by Number 2. Number 23 indicates the rows of gutters or trays in the greenhouse. Whether the crop is in a tray or a gutter depends on the growing stage, as schematically illustrated in Figure 2.7. The greenhouse is subdivided into circa five compartments for the different growing stages. A schematic drawing of the floor plan of the greenhouse is in Figure 2.9. There are rows of trays and gutters, indicated by 123, that enter a row at 105. The trays and gutters leave the rows at the top of the figure, to be transported to a central crop-handling area (103) conditioned to the tasks. This process facilitates an optimal working environment for both robots and human labour. This factor is the most important gain in efficiency for human labour, and an opportunity for a next step in tomato-greenhouse automation. The underlined numbers distinguish the different growing compartments. Stage 1 is at 106, 2 at 107, 3 at 108, 4 at 109, and 5 at 110. The climate conditions differ per compartment. The required space per stage has large variations.

The key element of the movable-gutter system is the gutter itself. The proposed gutter has a width

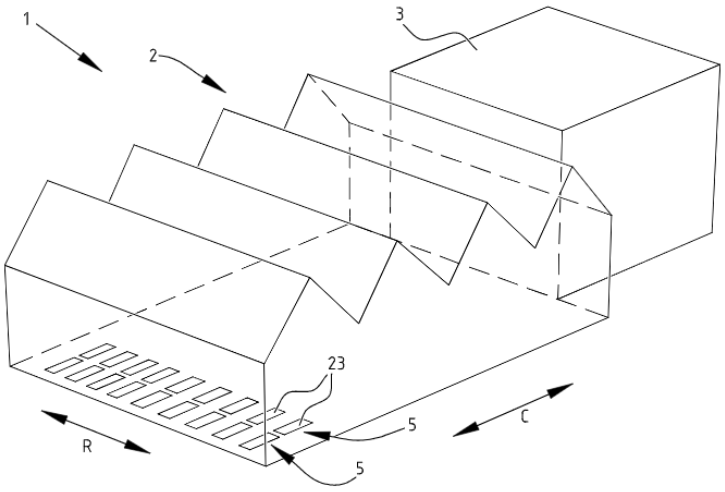


Figure 2.8: Schematic drawing of a greenhouse with a crop handling area. © Ridder Growing Solutions

of 8 [m]. The water, possibly with nutrients, is provided at 129 and flows into the gutter at 132 (Figure 2.10). The gutter slightly bends due to its own weight and the weight of the crops, despite the diagonal wires or cables of 130 being added. Therefore, water flows to the middle and is drained at 135 and 136. The gutter has supports at 121. These supports are above the centre of gravity to ensure the stability of the gutters. The support system for the crops is indicated at 131. The crops (124) reach a length of 1.1 [m] at their largest.

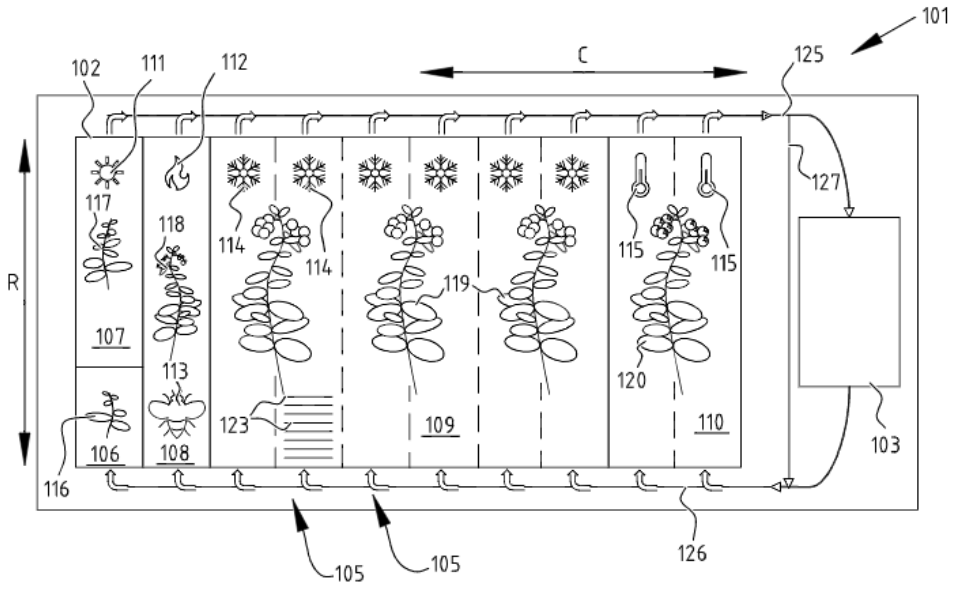


Figure 2.9: Schematic top view drawing of the different compartments of the greenhouse. Note that this image is not necessarily to scale © Ridder Growing Solutions

Figure 2.11 illustrates how the plant density varies. In that figure, the plant density is varied by changing the distance between gutters. Moreover, the number of plants per gutter can be varied. Note

that the distance variation is discrete in steps, as the watering system should be connected to the gutters at fixed places. In the gutters, the crops are cultivated at a fixed distance from each other for a batch. At the onset of growth in gutters, it is possible to choice another distance between plants by changing the number of plants per gutter. It is assumed that, on the gutter, every 25 [cm] there is a hole in which a plant can grow. Depending on the desired plant density, it is possible to choose how many of these holes will be used.

For the first two growing stages, no gutters will be used, only trays. Due to the fixed dimensions of the trays and gutters and the definition that a new compartment may only start at a new row, the area allocation per compartment can only be done in discrete steps.

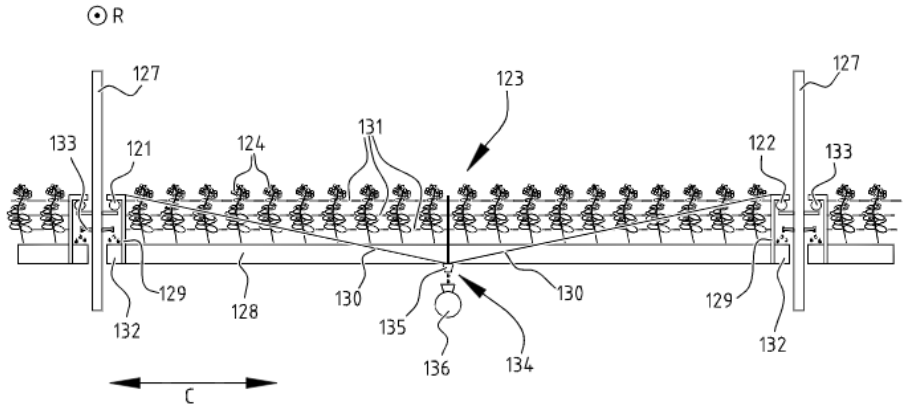


Figure 2.10: Drawing of front view of a gutter with plants at the latest growing stage. © Ridder Growing Solutions

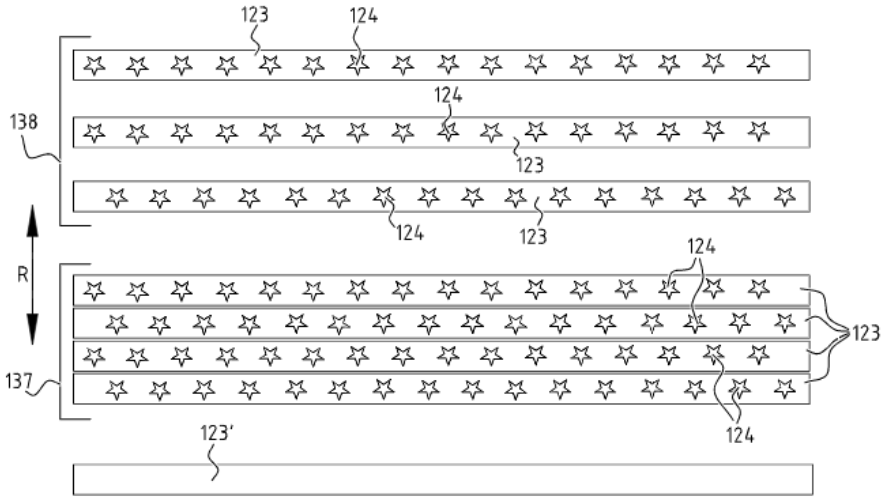


Figure 2.11: Top view of a row of gutters. At the stars crops can be grown. © Ridder Growing Solutions

2.4.3. Balancing the growth process

Regarding the cultivation process, it is important to define two terms: growth and development. The balance between these terms is important to achieve a target fruit or truss weight at harvest. Generally, a consistent fruit weight over the year is desired. For this study, a truss of 500 grams, with six fruits per truss is the target weight. For constant climate conditions, a constant fruit weight can be produced over the year. However, in a greenhouse, the inside light and temperature will vary seasonally.

Growth is defined as an increase in truss weight (TW in figures and tables). The second term is fruit development, which starts with the fruit set and ends when the truss is harvest ripe. Fruit development state (FDS) is defined to be 0 at fruit set and 1 at harvest ripe. The daily progress in FDS is expressed as the fruit development rate (FDR).

The truss growth is driven by light, whereas the FDR is driven by temperature. It is essential that the FDS is in balance with the truss weight: if not, too big or too small tomatoes are the result. The required balance of truss weight and FDS means the crop growth and FDR should also be in balance, which implies that temperature and light intensity should be balanced as well. This balance is schematically illustrated in Figure 2.12.

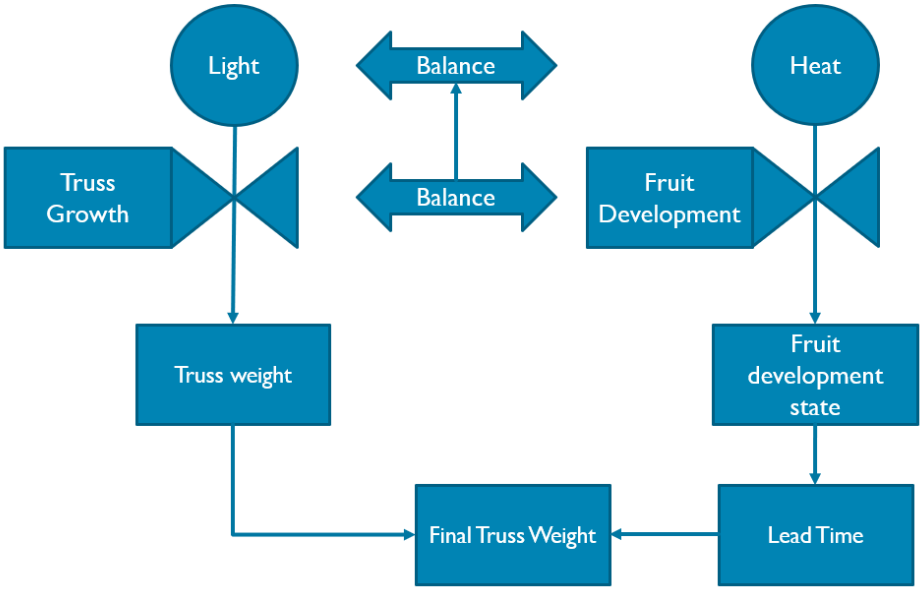


Figure 2.12: Schematic of balancing the fruit development and truss growth

Balancing light and temperature is complex. To determine this balance, the existing crop-growth model of De Koning (1994) is used. The relationships of the variables and parameters and how they lead to a harvested truss are in Figure 2.13. The circled ”x” is a multiplication, the circled ”+” is a summation, while the blank circles are a combination of manipulations. The variables are in the light green boxes, while the darker green boxes indicate the resulting variable that are used in the calculations in the model. One can observe that there is a large set of variables that will influence the final truss weight.

To review the input of the model, all the variables are mentioned in this section individually. The first set is the light intensity. The models used in this study should be able to process any realistic input light intensity. As starting, the monthly global radiance for Rotterdam, the Netherlands (Vermeulen, 2016) was taken and interpolated, with the curve illustrated in Figure 2.14. Rotterdam is chosen, because it is in one of the regions with the largest tomato production, data is freely available, and in the region of the cooperating company for this research. The total outside daily radiation curve is

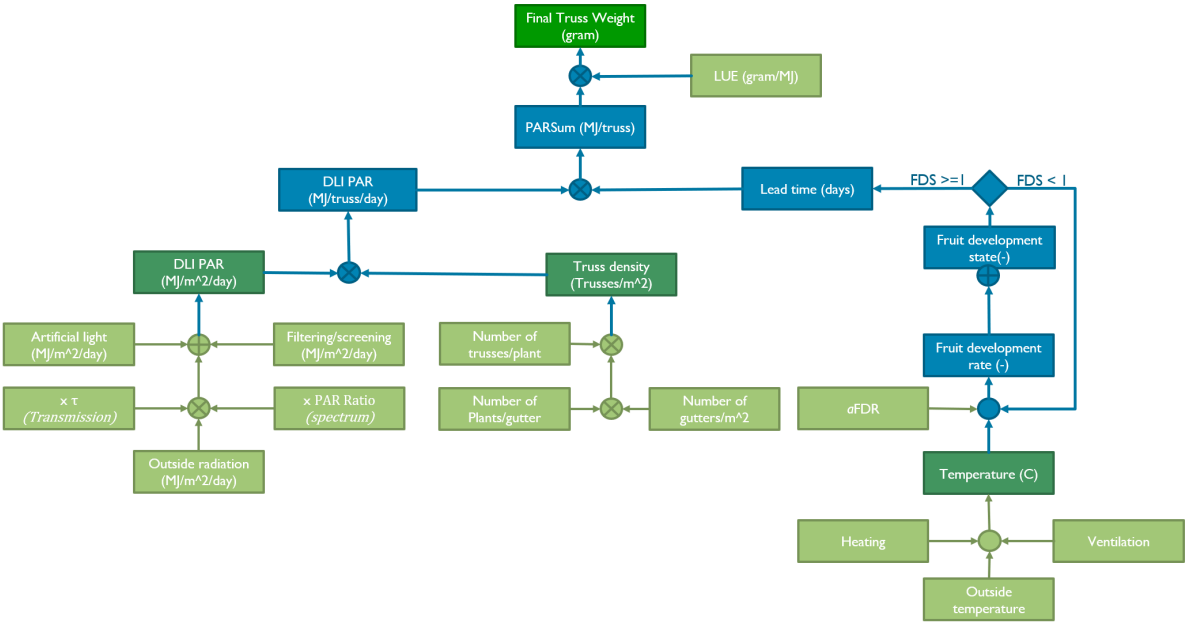


Figure 2.13: Variables and relationships in the cultivation process that lead to a harvested tomato truss of a certain truss weight

converted to a curve with the light sum of the relevant light (i.e. wavelengths in the PAR spectrum) for crop growth inside the greenhouse. This conversion is performed by multiplying the transmission rate of the greenhouse glass and the ratio of the relevant spectrum of the total for growth. The transmission rate varies according to the angle of incidence with the glass of the greenhouse (Mills, 2014). This angle varies over the day. As the outside radiation sum is a number per day, the transmission is taken as an average for the day. Additionally, this average transmission varies over the year, as the position of the sun in respect to a fixed location on earth changes; thus, the angle of incidence changes as well. However, it is assumed this difference is negligible.

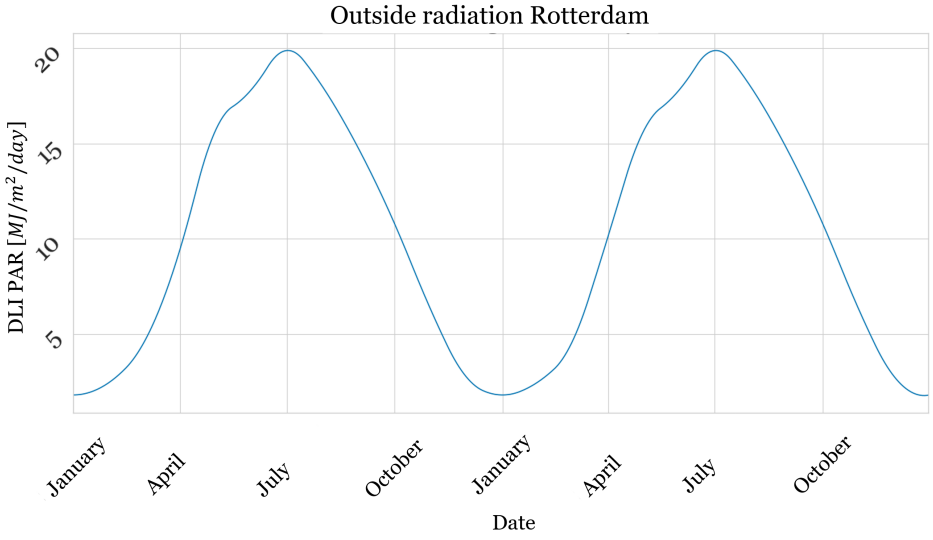


Figure 2.14: Outside radiation for Rotterdam (retrieved from Vermeulen, 2016)

Most important for the calculation of the transmission rate is the material type and its corresponding hemispherical transmittance coefficient. A smart material choice can increase the efficiency in light usages. Timmermans et al. (2020) investigated smart advanced materials to control the sunlight, and Timmermans et al. (2019) investigated ultra-thin glass. This ultra-thin glass can have transmission

coefficients up to 86%, whereas polycarbonate sheets transmit only 64% of the incoming light beam. These transmission coefficients are averages for the entire spectrum, as they can vary (Timmermans et al., 2020). In addition to the transmissive material, metal is used for construction, which does not transmit light. Therefore, the overall transmission rate is lower than that of the cover. A transmission rate of 0.80 is used as the average for most modern greenhouses (Vermeulen, 2016), and therefore used for this study as well.

The Photosynthetically active radiatio (PAR) Ratio is the amount of energy in the 400-700 nm spectrum as a fraction of the energy of the entire incoming beam (Timmermans et al., 2020). This ratio is the fraction of the incoming light in the relevant spectrum for crop growth. However, the PAR ratio varies seasonally by a changing composition of the incoming light beam, due to a variation in the average angle of incidence. A yearly average number of 0.50 is taken, based on Nederhoff and Marcelis (2010). By multiplying the initial curve by the transmission rate and the PAR ratio, the base curve for the daily light integral (DLI) is created (Figure 2.15).

Subsequently, artificial light and filtering are added to the inside DLI curve. The amount of artificial light and filtered light by screening is kept zero for now. In Section 4.1, the importance of artificial light is emphasised, with some examples to determine the thresholds of filtering and artificial lighting. Artificial light during the winter is needed because, for Dutch conditions, the crop will not grow without it. During summer, screening can be used to prevent excessive growth and unfeasible high temperatures or even sunburn. However, filtering should be minimised, as this means a loss of energy supplied by the sun.

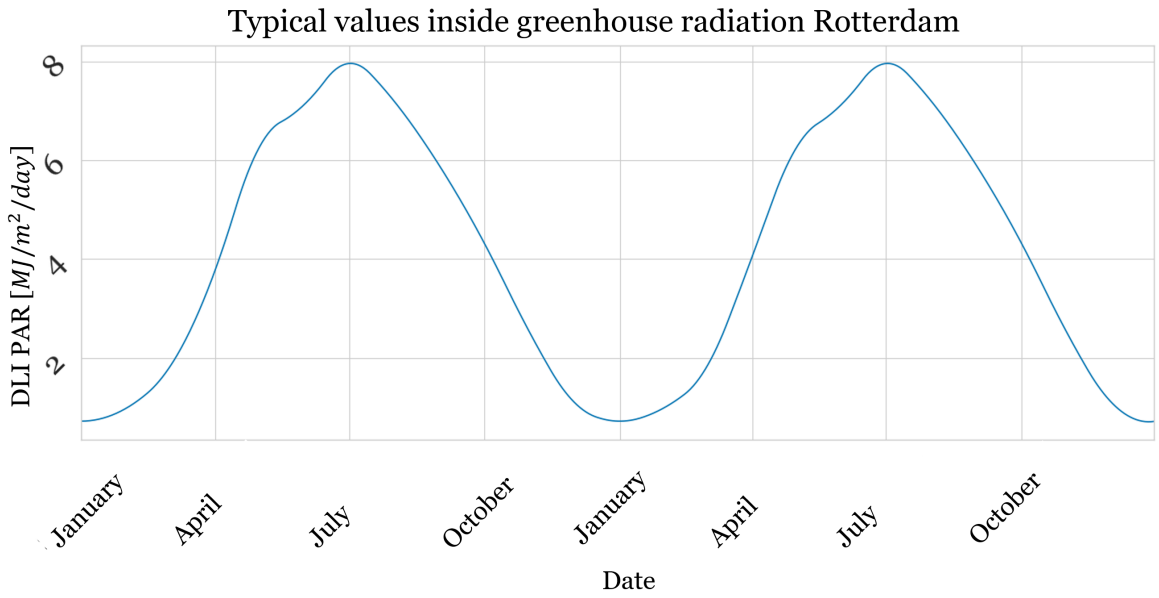


Figure 2.15: DLI PAR curve inside the greenhouse

The second main variable to vary is the truss density (number of trusses per square metre). There are three variables to change the truss density: first, the number of trusses per plant. Potentially, this number can be varied from one to three trusses. Economically, two to three trusses per crop seems most profitable; with one truss per plant, the rising costs are too high in relation to the revenue of a truss. Logendra et al. (2001) demonstrated that two or three trusses are more efficient than one. Therefore, the possible choice is restricted to two or three trusses per plant.

The second choice is the number of plants per gutter. A maximum of 32 holes per gutter is considered feasible for a gutter of 8 [m]. Theoretically, any number from one to 32 plants can be cultivated on a single gutter. Practically, an even distribution over the gutter is desired. A distribution

of 8, 12, 16, 24, 32 is therefore proposed as a starting point.

The third choice is the distances between gutters. These are also discrete steps, because the gutter should be connected to a watering system at discrete points. The starting point is that, in a compartment, there can be two to 10 gutters per metre of path, in steps of two. This layout offers the feasible options of 2, 4, 6, 8, or 10 gutters per metre. The simulation model should be used to remove a set of these options for a compartment. The model informs which gutter distances are rarely used for a compartment. Reducing the number of options is favourable, as fewer connection points to the watering systems need to be built.

The final adjustable variable is temperature. It is assumed that the temperature follows the trends of the outside radiation curve and remains within a feasible range. Figure 2.16 illustrates the growth of fruits for different temperature regimes in optimal light conditions. It is evident there is a limited feasible temperature regime, which is. Based on tests of De Koning (1994), this is set the 24-h mean temperature from the 17 to 23 degrees Celsius for the Netherlands. Once the temperature is below this range, there will be no fruit development. For regions with higher light intensities, this temperature can potentially be higher.

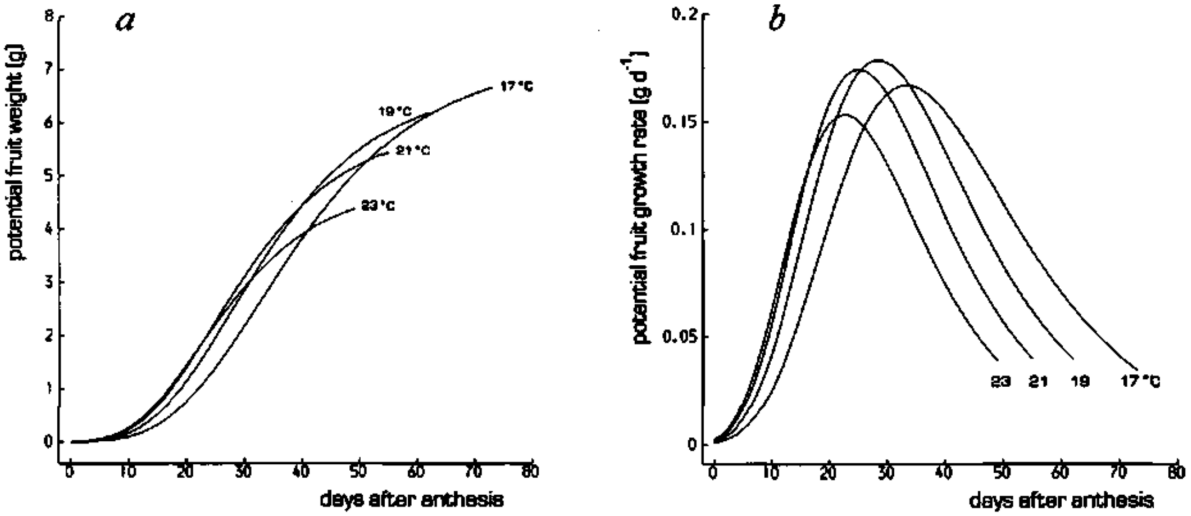


Figure 2.16: Curves that visualise the fruit growth for different temperature profiles. Figure a equals the integral of Figure b. (De Koning, 1994)

Another variation is using cultivar-dependent parameters. For the different cultivars, different cultivar-specific parameters are used in the relationships between temperature and FDR. For this study, the empirical parameters of the study of De Koning (1994) were employed. In principle, any cultivar could be implemented, but such variation was beyond the scope of this study.

The final key parameter is LUE. This parameter quantifies how many grams the truss weight will increase when 1 MJ of light is subjected to the crop. The parameter has been improved over the past century by introducing new cultivars (Heuvelink et al., 2007). Other, variables that influence LUE are the use and type of artificial light (Verheul et al., 2022), the season (Heuvelink et al., 2021) and the concentration of substances in the air, as the CO2 concentration (Heuvelink et al., 2021). There is no all-encompassing straightforward formula that represents this relationship. The absence of this relationship makes it complex to determine the LUE precisely. For this research project, this complexity is left out of scope and a constant average value of 45 [g/MJ] was taken (based on Heuvelink et al. (2021)).

2.4.4. Potential automation technologies for crop-handling tasks

The different crop-handling tasks that should be performed for the new cultivation concept are in Figure 2.7, which are sowing, transplanting, harvesting, and sorting. Therefore, only potential automation

solutions for these tasks are reviewed in this section.

The beginning of the cultivation process is sowing. This task can be performed automatically using the Visser Horti Systems Auto Seeder Granette. The machine can seed 220 trays per/hour with a maximum tray size of 600x400 [mm]. A requirement for reaching that capacity is that the trays are offered to the machine at the desired speed. Seeds also need to be fed to the machine occasionally. The machine has a big hopper that enables it to sow large batches without refilling. The dimensions of the machine are 2.5 x 0.8 x 1.2 [m]. If a higher capacity is required, redundant machines can be used. A switch to a higher capacity machine of 700 or 1000 trays/hour can be made. These machines are considerably larger than 220-tray machines. The dimensions for the larger machines are 3.5 x 1.9 x 2.2 [m] and 3.0 x 1.0 x 1.7 [m] for the 700 and 1000 trays/hour, respectively. However, these capacities are only the peak capacities. It is assumed that the average operational capacity equals 80% of the peak capacity.

Figure 2.17: Visser Horti Systems Auto Seeds Granette

Following the initial onset, the plants need to be transplanted from the high-density trays to a medium in which the plant density is smaller. The second transplant step is from a low-density tray to a gutter. Multiple solutions are available for transplanting the plants from a tray to other movable mediums. For example, the ISO Group Grade 7000 (Figure 2.18) can be used to plant the transplants until they reach a length of 25 [cm]. Therefore, the machine is a feasible solution for both transplant steps (i.e. from a high-density tray to a low-density tray, and from a low-density tray to a gutter). However, some adjustments to the machine should be made to handle the newly designed gutters. For this research, this aspect is assumed to be possible. The maximum capacity of this machine equals 7000 plants/hour. Applying the 80% estimation provides an operational capacity of 5600 plants/hour The considered dimensions of the machine are 3.5 x 3.2 x 2.2 [m].

Figure 2.18: ISO Group Grade 7000

Monitoring of the process is needed after the initial stage. This machine is only needed for the growing stages in gutters; inspection for the other stages occurs at the transplanting step. Monitoring in this context requires a machine that inspects the crops in gutters every time they enter the crop-handling area. This step ensures the health of a plant and monitors the growth process. After a plant passes the monitoring machine, it is necessary to decide whether the plant can return to the greenhouse, whether a specific treatment is needed, or whether the plants should be eliminated from the process. An automated vision system is not available for the tomato crop, as it is not used in a movable medium. It is estimated the inspection of one gutter takes 10 seconds, which means a throughput of 360 gutters/hour.

The final handling step in the cultivation process is harvesting. Harvesting the ripe tomatoes can be performed by robots or human labour. However, the success rate of robots is considered too low to be operational. Therefore, human labour will perform the harvest task. The gutters will be presented to the human labour, and they will harvest the trusses. The picking speed of a human is set as equal to 10 seconds/truss. This means that one human can harvest 360 trusses/hour. A human will handle one gutter at a time. The space one human operator needs is estimated as twice the gutter width. A gutter will have a width of 8 [m]. The space needed is assumed to be 16 x 2 [m] per human operator. In the future, this human labour can be replaced by Solutions robots. The most promising solution is the MetoMotion robot (Nir, 2021 and Nir and Nir, 2017, see Figure 2.19).

Figure 2.19: MetoMotion harvesting robot © Ridder Growing Solutions

Table 2.2 summarises the content of this section.

Table 2.2: Overview of the characteristics of automation technology per crop-handling task

Task	Solution	Throughput	Dimension per machine or operator [lxwxh [m]]	Comments
Sowing	Visser Horti Systems Auto Seed Granette	176 trays/hour	2.5 x 0.8 x 1.2	
Sowing	Visser Horti Systems Auto Seeder Speed Rouline	560 trays/hour	3.5 x 1.9 x 2.2	
Sowing	Visser Horti Systems Auto Seeder Roulette	800 trays/hour	3.0 x 1.0 x 1.7	
Transplant Step 1	ISO Group Grade 7000	5600 plants/hour	3.5 x 3.2 x 2.2	Transplanting from high to low density tray
Transplant Step 2	ISO Group Grade 7000	5600 plants/hour	3.5 x 3.2 x 2.2	Transplanting from low density tray to gutter
Harvesting	Human labour	360 trusses/hour	16 x 2.0 x 2.0	Most promising replacement for the near future is the MeToMotion robot
Monitoring	Human labour	360 gutters/hour	16 x 2.0 x 2.0	Such a machine is not available for this new cultivation process yet

2.5. Conclusion

This chapter answers the first sub-question: *What could be a viable path towards automation in tomato greenhouses?* Two perspectives were considered: 1) the cultivation concept, and 2) the state of automation. First, regarding the cultivation concept, only one commercially operational cultivation concept is available, which relies on the high-wire principle. Another cultivation concept involves using limited-truss tomato plants. These plants reach a length of up to 1.1 [m] and produce up to three trusses, in a cultivation cycle of up to three months. One advantage of these cultivation cycles concerns vulnerability to illnesses. If a plant becomes ill, only up to three trusses are lost, whereas in the old concept, up to 35 trusses can be lost. However, the claimed economic viability focuses on the

possible higher plant density and more successive cultivation cycles in a year. Although this concept was researched in the 1990s, it is still not commercially operational. However, although the concepts seem to differ greatly, they are built on the same principles, in which the balance of temperature and light is key for a viable cultivation process.

The second perspective considers the state of the art required for crop-handling in the cultivation concept. There is hardly any automation technology commercially operational, despite it having been researched for more than 15 years. The major problem is the high-wire concept causes a jungle-like environment. For other crops, a higher level of automation is achieved when a movable-crop concept is implemented. The crops are transported to a central crop-handling area, where the environment is adjusted to suit robots. Another advantage of performing the handling tasks centrally is that no one has to enter the greenhouse. Therefore, the climate can be fully optimised for crop growing, and walking paths are no longer necessary. Even if a handling task cannot be performed by a robot, it is still more efficient, as the work can be performed in an environment that is also optimised for human labour. Therefore, the automation-facilitating movable-crop concept applied to greenhouse tomatoes is researched in this study. A limited-truss concept is implemented, as the high-wire concept is not feasible for transport through the greenhouse. The combination of both the limited-truss concept and the movable-gutter concept could be a viable towards automation in tomato greenhouse. The next chapter will discuss what methodologies should be applied to develop a generic design framework for this new limited-truss tomato on a movable-gutter concept.

3

Applicable modelling methodologies

Chapter 2 explained that the limited-truss tomato on a movable-gutter concept is sensitive to light and heat; therefore, the design could change for different climate conditions. As the outside climate is an important factor for the inside climate, the design will differ in various geographic locations. Many other variables could also influence the design. Therefore, it is not feasible to develop a sample design that can be applied to various cases, as the design needs to be recalculated for each case. Thus, model-based simulation is applied. Another advantage of model-based simulation is that it can identify design errors before significant resources are spent. Moreover, multiple configurations and scenarios can be simulated, which gains insights into potential performance. In this chapter, the structure of the generic design framework is developed, and approaches for different modules in the simulation model are discussed. Discussing the structure and modules help to answer the second sub-question: *What modelling methodologies can be used to generate and simulate conceptual designs?*

The generic design framework developed to fill the knowledge gap can be reduced to four basic blocks. The blocks are 1) inputs, 2) simulation model, 3) evaluation, 4) output. These blocks are indicated in the schematic overivew of the framework in Figure 3.1. The inputs are the climate conditions, a target truss weight, the cultivar and growth dependent parameters, lot and case specific characteristics and the scenarios. In the simulation model the conceptual design and sowing strategy is generated and simulated. In the evaluation step, the simulated performance is evaluated and improvements for the conceptual design and strategy are proposed and implemented. This leads to an iterative improving design till a certain threshold is reached. Then the output is presented, which is a conceptual design and sowing strategy.

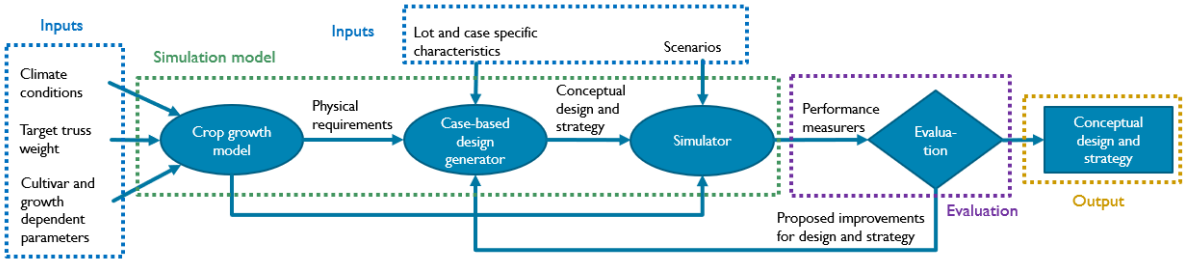


Figure 3.1: Structure of the framework developed in this research project

In the simulation model, multiple steps must be taken. First, the growth of the tomatoes should be modelled, which leads to the physical requirements. These physical requirements answer the questions about each sowing day, such as what the lead time of a batch will be and what the corresponding required area for that batch should be. Together with the lot-specific characteristics, an optimisation step can be applied to develop an optimal conceptual design with an optimal sowing strategy. This module is called the case-based design generator. The last step is to simulate this design and sowing

strategy to obtain a simulated performance, which is measured by KPIs. The structure of the simulation model, within the structure of the framework developed is in Figure 3.1

The evaluation block contains a protocol of multiple steps. The overall structure is in this section, while the individual steps are discussed in greater depth in Chapter 5, except for the first block, which is discussed alongside the development of the crop-growth model in Section 4.1. The first step is to examine lower thresholds for climate conditions. If there is too less light of the PAR spectrum, there will be no growth. If the temperature drops below a certain threshold, there will be no fruit development. The second evaluation step is a sensitivity analysis, which is followed by an optimal dimension analysis. These steps provide a greater insight into the system. After the optimal dimension analysis, case studies were constructed to prove that the framework is generic. These case studies also serve as an example for the next evaluation steps. In these steps the simulated performance is evaluated, the optimal capacities of the transport system and for crop-handling tasks are calculated, and the robustness is assessed. If in any of these steps, something is not satisfactory, an improvement is proposed, implemented and simulated. When the evaluation protocol is passed, the conceptual design and strategy are presented. The structure of this evaluation protocol is in Figure 3.2.

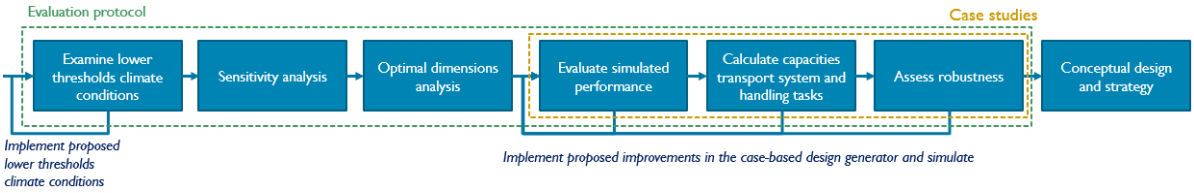


Figure 3.2: The evaluation protocol used to obtain a best conceptual design and strategy

Using simulation for design can be subdivided into two categories: 1) model-based simulation, and 2) test-based simulation (Mefteh, 2018). For test-based simulation, the designer should have a complete knowledge of the system. The designer wants to simulate a system before building the real system. In the model-based simulation, complete knowledge about the system has not been attained yet. The simulation is used to gain more information about the system and to explore different design possibilities. Because of the lack of complete knowledge of the system, model-based simulation applies to this study.

There are multiple workflows to develop such a simulation model. A widely used approach is the spiral workflow (Figure 3.3). In this workflow the developer begins simply, complexities are added after it is ensured the current model works. After every new round in the spiral, the model should become more complex, but also closer to the reality. The complexity of the real system means it is not feasible to implement it all at once. Even after this study, complexities could still be added. An advantage of this approach is that the model can be used during development. Even without implementing all the complexities, the model can be reliable, accurate, and precise enough to make decisions about the conceptual design.

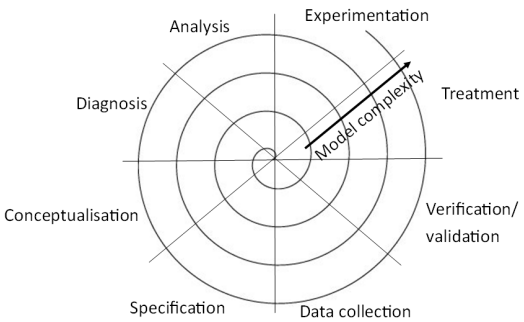


Figure 3.3: Spiral workflow, based on Verbraeck (2021)

3.1. Tomato crop-growth models

The first module in the simulation model is the crop-growth model, which simulates the growth and development of tomato plants. Given light and temperature conditions, the expected crop development

and crop growth can be calculated. A common application of growth models is to steer the growth in the desired direction. Growth models are specific per crop. Some existing models for the tomato crop are now discussed. One of the key features in growth models is the balance between leaf and fruit growth. One of the first all-encompassing models was developed by De Koning (1994). His model is still relevant, as it has been cited 107 times in the past decade, according to the Google Scholar search engine.

The developed model of De Koning is used to understand the growth process of a tomato plant and the dry matter that will be produced. The development of a tomato plant appears to be most sensitive to temperature. A set of parameters specific for the cultivar and the temperature is used to calculate the FDR for a day. When the development of the fruit (the cumulative FDR) reaches the stage of harvest ripe, the fruits can be harvested, and the cultivation cycle ends. One output of the model was already presented in 2.16. That figure emphasises the importance of balancing the light and temperature to produce trusses of a constant weight. For a constant light intensity, the fruit weight at the end of the cycle differs significantly (up to 40%) due to differences in length of stay in the greenhouse caused by a different temperature. The fresh fruit weight at the end of the cycle varies proportionally according to the length of stay.

Furthermore, also Vanthoor et al. developed a widely used model (Vanthoor et al., 2011), which was cited 55 times in the past decade. The development employed a variety of earlier studies for the model, including De Koning's. The model is validated for different temperature regimes and proves accuracy for these different regimes. In Vanthoor et al., the temperature and light intensity are the dominant variables, but the influence of CO2 concentrations, and relative humidity are also influential.

Another yield-prediction model that is cited often (51 times in the past decade) is TOMSIM. This model was developed by Heuvelink (1996). The TOMSIM model uses the principles of the De Koning model, but with a more thorough validation that substantiated the model's claim of being reliable and accurate when predicting tomato growth. The relationships of the variables display similar trends and sensitivities for TOMSIM and De Koning.

A fourth tomato-yield-prediction model that is still relevant, is the TOMGRO model (Jones et al., 1991). The model was cited 83 times in the past decade, with 24 times being in the past two years. Munoz et al. (2019) used the TOMGRO model as basis for their greenhouse modelling as a service product. The reduced TOMGRO model (i.e. reduced in terms of parameters) consists of 21 parameters. These parameters can be determined using genetic algorithms. A thorough sensitivity analysis is performed, in which the temperature, light, and CO2 appear to be the most sensitive parameters. In addition to promising validations in the initial study, other studies have also reviewed the performance of the TOMGRO model. For example, Gong et al. (2021) calibrated the model using generic algorithms. The output of a simulation performed by Gong et al., with using their found parameters, produced an average mean relative error of only 0.011. This result is an indication of good performance but cannot be compared one-to-one with other studies, because without detailed information on the dataset used, or detailed information about the validation, the results are difficult to place in perspective with other studies.

A study that places the performance of TOMGRO in perspective is that of Lin et al. (2019).The researchers compared the models of TOMGRO and Vanthoor et al. Both models were thoroughly analysed, and the strengths of both models led to a new integrated model. One difficulty of the TOMGRO model is that the parameters of Jones et al. (1991) are not fully accurate, which causes a greater deviation of the actual yield. This issue is also the reason that later calibration studies, such as Gong et al. (2021), were conducted. For the Vanthoor et al. model, the weakness is the level of detail of the crop mechanism. The developed integrated model by Lin et al. (2019) worked on the weaknesses for both the model of Vanthoor et al. and TOMGRO, which led the root means square error (RMSE) to fall from over 17 for Vanthoor et al. and TOMGRO, to below 2.6 for their developed model. This outcome is one indicator the Lin et al. model performs well. Therefore, this is a promising model, which could become the new standard, but it still must prove its performance in practice and for other datasets, which the older models of TOMSIM, TOMGRO, De Koning, and Vanthoor et al. already did.

For this study, it was preferred to implement one of the four proven models. All these four older models could haven been implemented. At the company that cooperated with this study (i.e. Ridder Growing Solutions), the De Koning model is used commercially for yield productions. Additionally, the company has deep knowledge about the model. Therefore, the De Koning model is formalised and implemented in this study. Note that this tomato crop-growth model is validated for the conventional process. It is assumed that the model still hold for the new cultivation concept. However, this hypothesis needs to be proven or rejected by a validation in another study.

3.2. Optimisation models for strategy in horticulture

The outputs of the crop-growth model are used as inputs for the case-based design generator, which has a greenhouse strategy element in it by calculating an optimal sowing strategy for maximising the turnover. In addition to this strategy, the required output is a conceptual design. An automatic area allocation aspect is new for the application in this study and cannot be found in the literature. To determine a greenhouse strategy, a set of solutions is available. For this study, greenhouse strategy is defined as sowing strategy. To determine the sowing strategy, the maximum capacities should be considered.

There are multiple greenhouse strategy aspects: cultivation planning, harvest planning, production planning, distribution planning, and inventory management. All these aspects should be well managed for a smooth operation. However, as sowing strategy was chosen as one of the decision variables, cultivation planning and harvest planning are most important for this study.

A search was conducted in the google scholar engine for models that at least mention cultivation planning. Solutions from the horticulture sector were found. Even if designed for other crops, the models can be adjusted to the tomato crop. The most suitable model depends on the application. By using the new cultivation concept (section 2.4.1), there are more opportunities to optimise the planning than for the conventional method. Therefore, more sophisticated models can be employed for the new concept, whereas for the conventional concept this adds less value.

Table 3.1 lists models with different characteristics. The list reveals that popular approaches are multi-integer programming (MIP), linear programming (LP), and a combination of both (i.e. multi-integer linear programming (MILP)). The MILP approach is widely used in complex parallel systems. The approach is claimed to be a quick and state-of-the-art method (Urbanucci, 2018). A drawback of this method is that MILP cannot handle non-linearities in a system. In addition, the entire period must be considered at once, and there is a risk of high dimensionality of the problem. However, there are methods to address this limitation. As the case-based design generator should generate a reasonable input, but is not the main focus of this research, it can be kept relatively simple; not too many variables are implemented, which will resolve the high dimensionality issue.

For this study, due to the new application, the case-based design generator cannot be based upon an existing model; too many adjustments must be made. Therefore, a new model should be developed. However, the structure of existing models can be used. For example, the structure of Wishon et al. (2015) can be implemented to determine a sowing strategy. Some key features are described in this paragraph, but for a complete understanding, the report of Wishon et al. is recommended. The Wishon et al. model begins by defining the five sets used. The crops, growing and harvesting seasons, and the workers are defined. Then, a list of variables and coefficients is provided, followed by the objective function and the constraints. The objective function of Wishon et al. is to maximise profit, and this includes the optimisation of operational factors. The variables included in the objective function are the units harvested, the units sold as surplus, the units purchased to meet contracted demand, human labour, and the newly planted units. The parameters in the model of Wishon et al. are the market price, the salvage unit price, the contracted unit price, the minimum contracted demand, different labour costs, and the costs to harvest. The modelling approach is MILP, which is a mathematical optimisation method in which the variables are restricted to integers. Moreover, the objective functions and constraints must be linear. There are multiple different solvers available once the model is mathematically defined, among which is the Gurobi solver (Gurobi Optimisation, LLC, 2021).

Table 3.1: Overview of state-of-the-art models for greenhouse strategy

Source	Decision Variable	Decision level	Time horizon	Objective function	Model features	Number of products	Product	Model type	Progr. type	Data	Solutions method
Banasik et al., 2017	CP, HP	T	MP	PM, OT	RL	M	Mushroom	MO	MILP	RC	e-constraint
Banasik et al., 2019	CP, HP	T	MP	PM, OT	RL, UC, PW, SU	M	Mushroom	MO	SP	RC	e-constraint,simulation
Wishon et al., 2015	CP, HP	S-T	MP	PM, OT	RL	M	Vegetable	MO	MILP	HP	Exact
Piewthongngam et al., 2009	CP, HP	T	MP	OT	RL	M	Sugar cane	SO	LP	HP	Exact
Huang et al., 2020	CP, HP	T-O	MP	PM	TW, RL, UC	S	Crops	SO	MIP	HP	Exact
Vitoriano et al., 2003	CP, HP	S-O	MP	PM	TW, RL	M	Multiple	SO	LP	HP	Goal programming
Tan and Çömden, 2012	CP, HP	S	MP	PM	TW, UC	S	Tomato	SO	SP	HP	Approximate
Accorsi et al., 2016	CP, PR, DT	S	SP	PM	RL, SU	S	Potato	SO	LP	HP	Exact
Ahumada et al., 2012	CP, HP, PR, DT	T	MP	PM	TW, PE, RL, UC	M	Vegetable	SO	LP	HP	Exact
Grunow et al., 2007	CP, HP, DT, IN	T-O	MP	CM, OT	TW, RL	S	Sugar cane	SO	MILP	RC	Exact
Costa et al., 2014	CP, HP, DT, IN	S-T	MP	PM	PE, RL, UC, PW, SU	S	Vegetable	SO	LP	JP	Exact
Ahumada et al., 2012	CP, HP, PR, DT, IN	T	MP	PM	RL	M	Vegetable	SO	MILP	HP	Exact
Flores and Villalobos, 2018	CP, HP, PR, DT, IN	S-T	MP	CM, PM	TW, PE, RL, UC	M	Vegetable	SO	SP	HP	Exact
Motevalli-Taher et al., 2020	CP, HP, PR, DT, IN	S-T	MP	CM, OT	RL	S	Unspecified	SO	IP	HP	Metaheuristics
Hajimirzajan et al., 2021	CP, HP, PR, DT, IN	S-T	MP	CM, OT	PE, RL, UC, SU	M	Tomato, potato, onion	MO	MIP	HP	Exact

CP: Cultivation Planning; HP: Harvest Planning; PR: Production Planning; DT: Distribution; IN: Inventory;

O: Operational; T: Tactical; S: Strategic;

SP: Single Period; MP: Multi Period;

PM: Profit Maximisation; CM: Cost Minimisation; OT: Other Objective;

TW: Time Window; PE: Perishability; RL: Resource Limitation; UC: Uncertainty; PW: Product Waste; SU: Sustainability;

S: Single product; M: Multiple products;

SO: Single-Objective; MO: Multi-Objective;

LP: Linear Programming; MILP: Mixed Integer Linear Programming; IP: Integer Programming; SP: Stochastic Programming (e.g. Markov); MIP: Mixed Integer Programming

RC: Real Case; HP: Hypothetical Case

3.3. Simulation methods for the new concept

There are many approaches to simulate the design generated in the previous module. The four most widely used approaches are as follows: 1) agent-based modelling (ABM), 2) discrete-event simulation (DES), 3) system dynamics modelling (SDM), and 4) model predictive control (MPC). All approaches have their strengths for the specific application. First, the main characteristics of the four methods were described. Then, the applicable method for this study is chosen.

The ABM approach simulates the actions and interactions of autonomous agents. The goal is to understand the system's behaviour and what leads to the outcomes (Hussain et al., 2014). The approach combines elements of a set of other approaches. Monte Carlo methods are used to

implement stochastic elements in the model. The ABM approach is used in a variety of domains, predominantly in biology, ecology, and social sciences. The strength of gaining insights into the system's behaviour means ABM is usually used for 'explanatory insight into the collective behaviour of agents obeying simple rules, typically in natural systems, rather than in designing agents or solving specific practical or engineering problems' (Muaz and Hussain, 2011).

A widely used simulation approach for system analysis and system engineering is DES. The DES approach can simulate how the system states will evolve. The principle is that the events that change the state will happen at discrete points in time. Since three decades, DES is also combined with object-oriented programming (OOP), to form OO-DES, which is a powerful basis for studying the dynamics of and capture a system's complexity (Zeigler, 1991).

The main goal of SDM is to understand the non-linear behaviour of complex systems, using stocks, flows, internal feedback loops, table functions, and delays. This mathematical technique is mainly applied in industrial processes, policy analysis, and system design, generally from a socioeconomic viewpoint (Radzicki and Taylor, 2008).

The MPC approach can be applied to a variety of systems. Commonly, the goal is to represent the behaviour of dynamic systems. The MPC models predict a change in dependent variables caused by changes in independent variables. Examples of such systems are reefer ships or chemical processes. The independent variables in these systems are often either setpoints, PID controllers, or flow control elements. Independent variables that cannot be changed are treated as disturbances, whereas dependent variables are measurements that represent control objectives or process constraints. Therefore, the most common application of MPC is the advanced control of processes. (Liuping, 2009)

There are major differences between the objectives and applicability of the different approaches. The OO-DES and ABM approaches seem most applicable to simulate the physical design and to obtain knowledge about parameter sensitivity. The MPC approach could be implemented in the system in a later control phase, but for tomato greenhouses it seems most applicable to the climate systems. The SDM approach seems inapplicable for the design of the physical system but could play an important role when deciding on the greenhouse strategy in an operational phase.

To choose the most applicable approach, it is essential to define the perspective from which the problem is treated. A historical overview of the different perspectives and their origins is in Figure 3.4. The most widely used modelling and simulation approaches are included in this figure. A *system* is defined as 'a combination of interacting elements organized to achieve one or more stated purposes.' (Fernandez and Hernandez, 2019). The new greenhouse concept fits this definition. An analysis of the proposed new system combined with systems engineering could lead to a feasible conceptual design. Once this conceptual, or even final, design is in place, control is the next step to consider. The design on that level is essential as well for a feasible final product. However, that step will be taken in a later design phase.

Therefore, OO-DES was the employed protocol to develop the simulator. The implementation of the model is done in Python by using the Salabim package. The simulator was developed using the spiral workflow (Figure 3.3). Initially, the model was kept relatively simple, but then, circle by circle, more complexities were added. Note that for this research, validation, and experimentation were not possible, as there was no physical system to validate the model, nor was there an experimental setup. What was possible was to conduct a verification and to check whether the outcome is within a feasible region.

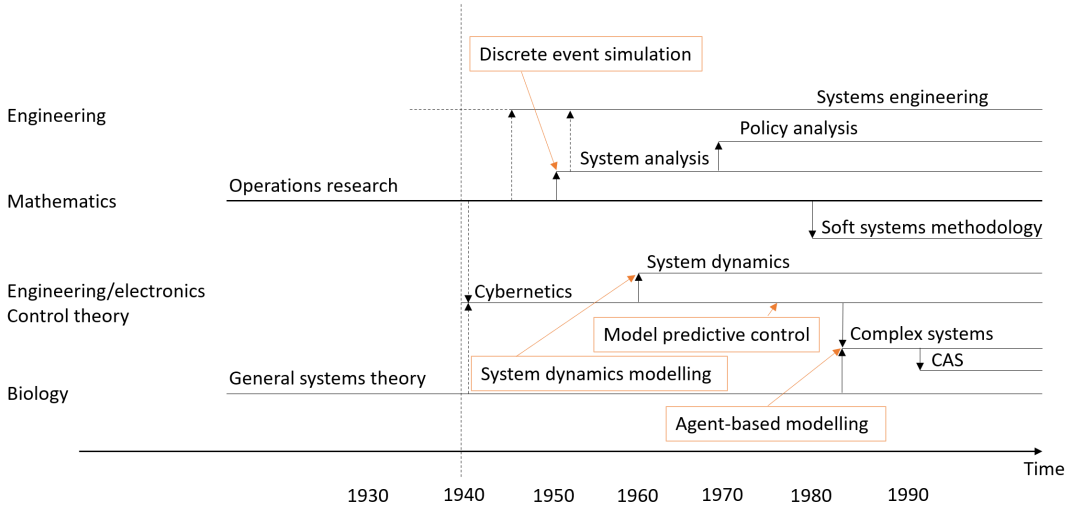


Figure 3.4: Historical overview of modelling techniques, based on Verbraeck (2021)

3.4. Conclusion

This content of this chapter was mainly based on literature, which is used to formulate an answer the second sub-question: *What modelling methodologies can be used to generate and simulate conceptual designs?* The first formalisation was the structure of the framework (Figure 3.1). Part of this framework is a simulation model. This simulation model can be split into three modules. The first module models the growth of a greenhouse tomato. The existing model of De Koning (1994) was used as the starting point. The relations of the De Koning model were used, but applied to the new cultivation concept. Based on the output of that part, a conceptual design was generated. The MILP approach was used to generate an optimal conceptual design for the case. Finally, this design was simulated to examine the performance. This simulation was conducted using the OO-DES protocol. In the next chapter, the development and implementation of these models is treated.

4

Structure of the model

In the previous chapter, the structure of the framework, which contains the simulation model was discussed. In this chapter, the three modules of the simulation model are developed, integrated, and implemented to answer the third sub-question: *How should the new concept be implemented in a simulation model?* In this chapter, the modules and their implementation is described by using text and diagrams. A pseudo code of the simulation model is provided in Appendix B, while the full implemented Python code is in Appendix C. Before the implementation steps, the inputs, outputs, and assumptions per module are discussed. The first module is the crop-growth model. In that module, the existing model of De Koning (1994), is used. The inputs, outputs and main assumptions of that first module are in Figure 4.1.

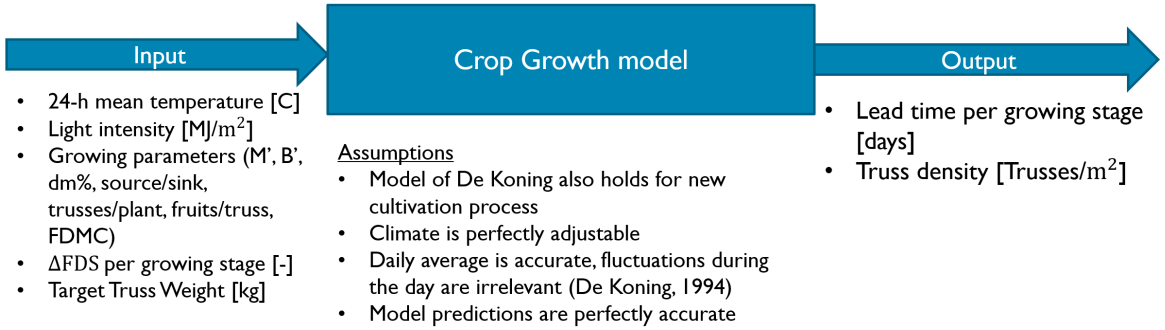


Figure 4.1: Inputs, outputs, and assumptions of the crop-growth model

There are two approaches that apply this model. With a given DLI PAR, the desired truss weight can be achieved by 1) calculating the desired temperature from a given truss density and DLI PAR, and 2) calculating a truss density from a given DLI PAR and temperature. This calculation means that for Approach 1, the truss density is an input, whereas the temperature is an output. Note that the feasible regions of the underlying physical parameters should be respected (e.g. respecting minimum temperature for viable fruit development). In the next section, the crop-growth model is formalised, and in the next subsection these two approaches are elaborated.

The case-based design-generating module (see Figure 4.2) uses the outputs from the growth model. In addition to these inputs, the specifications of the case study are presented. These specifications are the dimensions of the available area and the number of crop-handling areas. Furthermore, a tomato price curve is given, and possibly proposed improvements following from the evaluation are implemented. With these inputs, an MILP optimisation was conducted to obtain a sowing strategy and a list of tasks that must be performed on a specific day. The other output is the conceptual design,

which includes a floorplan with an area allocation and a list of the required machinery and labour.

To determine the area allocation, the dimensions of the total available area should be considered. A new greenhouse compartment can only start on a new row. By setting this constraint, the area per compartment can only be a set of discrete steps. A minimum row width is considered. The gutters have a width of 8 [m], whereas the width of a row of trays is 3 [m]. This leads to that with a row length of 100 [m], the available area for a compartment with gutters can only be an integer multiplication of 800 [m^2].

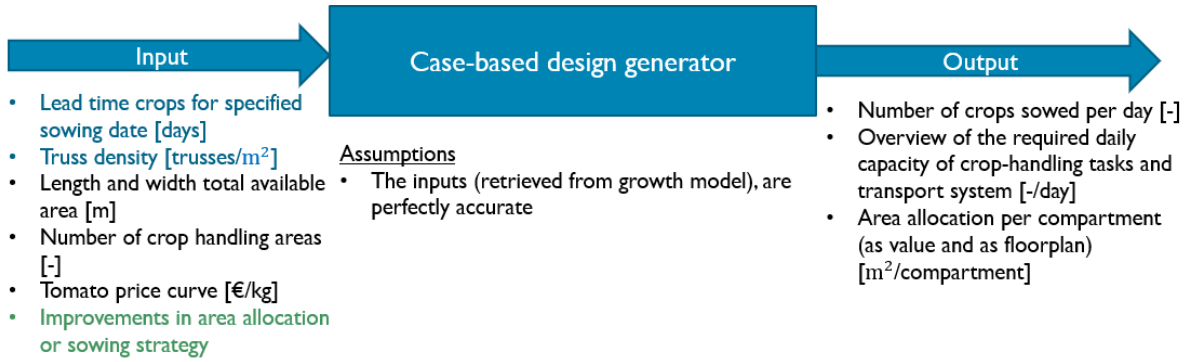


Figure 4.2: Inputs, outputs, and assumptions of the case-based design generator. Inputs in blue are direct outputs of the crop growth model. Input in green is an optional input following from the evaluation protocol.

A design was generated by the case-based design generating module. The next step was to simulate this design for fulfilling the evaluation protocol. The output of the case-based design generator was used as input for the OO-DES simulator. Scenario-specific inputs were also entered into the system. The outputs are performance measures.

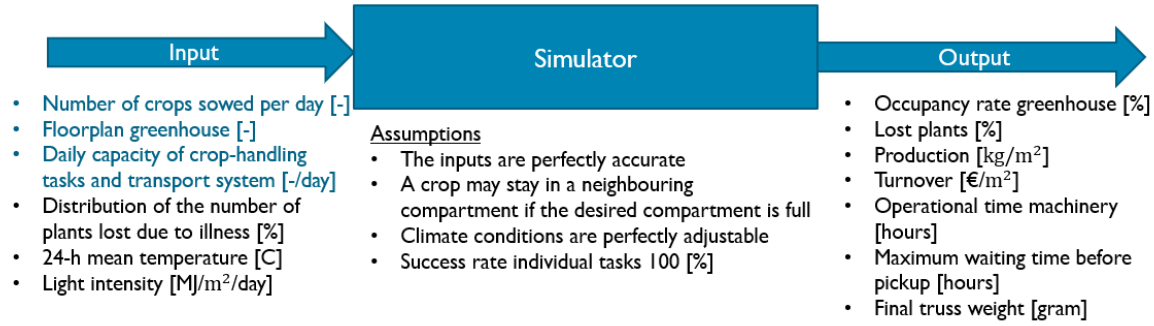


Figure 4.3: Inputs, outputs and assumptions of the simulation model. Inputs in blue are direct or indirect outputs of the case-based design generator.

4.1. Development of the crop-growth module

The crop growth model of De Koning (1994) was implemented, as this is a practice-proven and widely used model. Two approaches were tested with this model to conclude the most viable procedure (see Table 4.1). An overview of the calculation flows for the model is in Figure 4.4. The model performs a run of two years, to take one representative year. A two-year run for leaving one representative year was chosen because, for that case, the start up and end effects can be cut out.

Table 4.1: Overview of inputs and outputs for the different approaches

Approach	Inputs	Outputs
1a	DLI PAR, Truss density (constant)	Temperature profile, Truss weight
1b	DLI PAR, Truss density (3 discrete steps)	Temperature Profile , Truss weight
2a	DLI PAR, Temperature profile (constant)	Truss density (continuous values), Truss weight
2b	DLI PAR, Temperature profile (scaled with DLI PAR)	Truss density (continuous values), Truss weight
2c	DLI PAR, Temperature profile (scaled with DLI PAR)	Truss density (discrete values), Truss weight

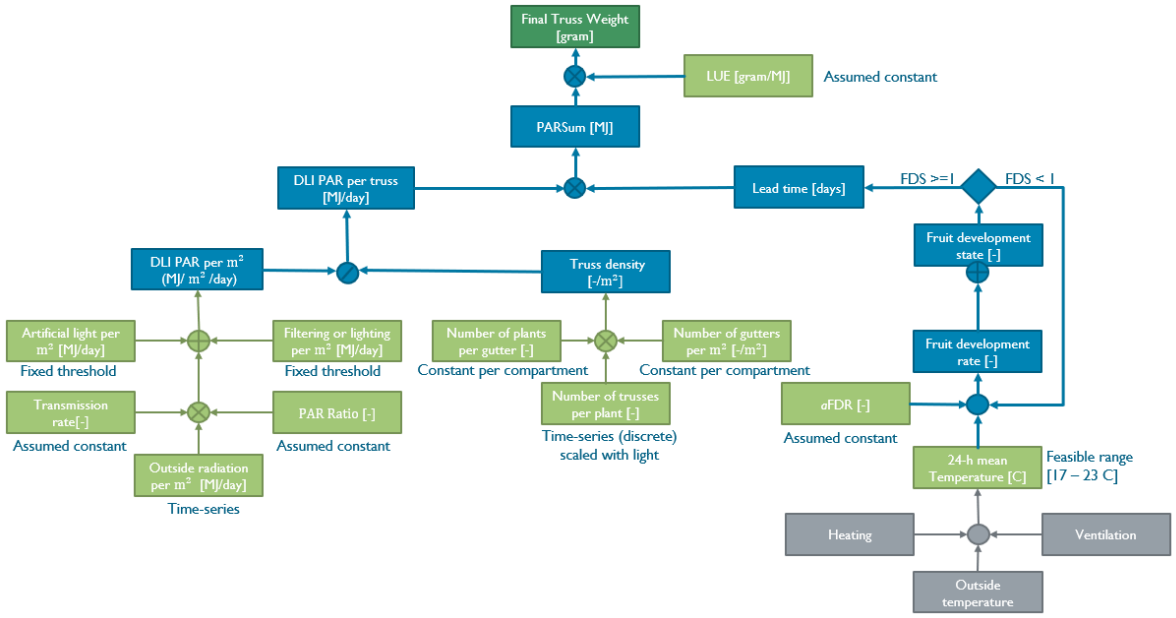


Figure 4.4: Schematic of the crop growth model as described by De Koning (1994)

Parameters

LUE	Light use efficiency	$gram/MJ$
αFDR	A cultivar dependent empirical parameter, defining FDR	$-$

Variables

$ADFG_t$	Actual daily fruit growth	$gram$
DLI_t	Daily light integral	MJ/m^2
FGR_t	Fruit growth rate	$gram/day/m^2$
FDR_t	Fruit development rate	$-$
FDS_t	Fruit development stage	$-$
T_t	24-h mean temperature	$Celsius$
$PlantDensity$	Number of plants per m^2	$-/m^2$
TW_t	Truss weight	$gram$
$Trusspp$	Number of trusses per plant	$-$
$TrussDensity$	Number of trusses per m^2	$-/m^2$

Equations

$TrussDensity_t =$	$PlantDensity_t * Trusspp$	$-/m^2$
$FGR_t =$	$DLI_t * LUE$	$gram/m^2$
$ADFG_t =$	$FG_t/(TrussDensity)$	$gram$
$TW_{t+1} =$	$TW_t + ADFG_t$	$gram$
$FDR_t =$	$\alpha FDR + log(T/20) * 0.02131$	$-$
$FDS_{t+1} =$	$FDS_t + FDR_t$	$-$

4.1.1. Approach 1: Calculating the optimal temperature

For Approach 1, the model as described in the previous formulas and Figure 4.4 was used. However, now a final truss weight was set as a target. This truss weight was used as the input, and the goal of Approach 1 was to calculate a corresponding temperature as output for a given truss density and light intensity. Converting Figure 4.4 to a flow with the named desired inputs and outputs leads to Figure 4.5. First, the truss density was constant (Approach 1a, Table 4.1). In the second run (Approach 1b, Table 4.1), it was assumed the truss density could also vary by changing the number of trusses per plant, which could be one, two, or three. In the conventional system, the number of trusses per plant can (slightly) vary as well. Therefore, changing the number of trusses per plant was varied in this test as well. The way this was coded in Python is illustrated schematically in Figure 4.6.

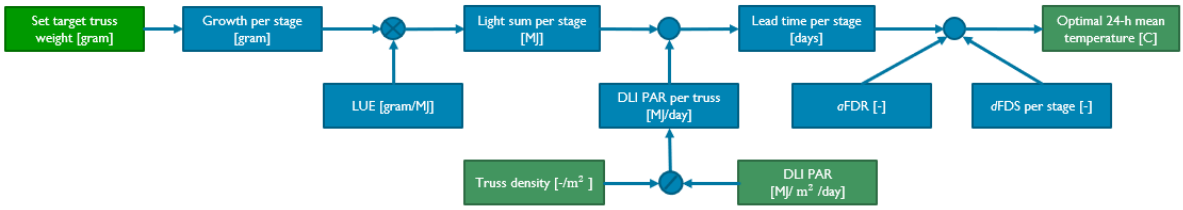


Figure 4.5: Flow to calculate the optimal temperature. This flow is converted from the scheme in Figure 4.4.

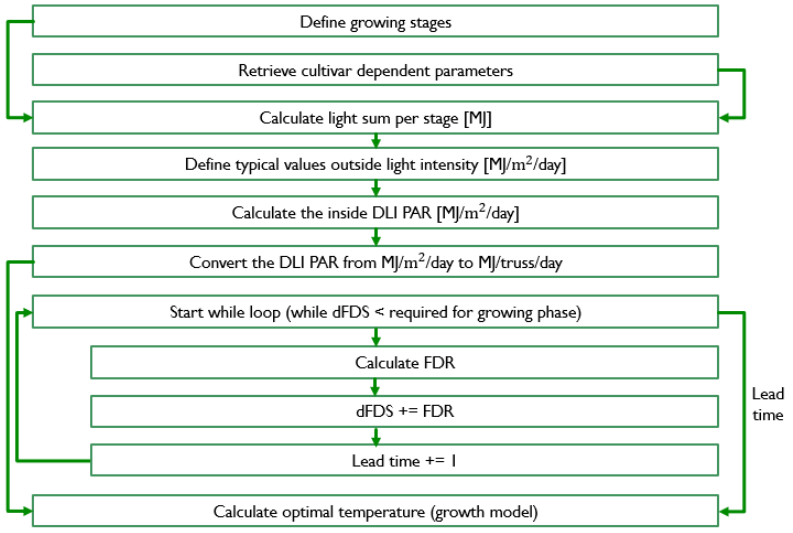


Figure 4.6: Implemented flow in Python corresponding to Figure 4.5

First, there was worked with a constant truss density. A more realistic case was varying plant density per compartment. The starting point for the plant density was calculated by using the growth model slightly differently. The number of trusses per plant was set to two, and an optimal (continuous) plant density was calculated, together with the lead time for a plant in a compartment. Calculating the

plant density was conducted by using the growth curves per plant. The formulas were retrieved from De Koning (1994). The base of the procedure with a varying plant density is similar to that with the constant plant density.

Parameters

FGW	Weekly fruit growth rate	$2/7 kg/m^2/day$
TFW	Target fruit weight	$1 kg/plant (= 500 gram/truss)$
T	24-h mean temperature	$21 Celsius$
αFDR	cultivar dependent parameter determining FDR	0.17

Variables

FDS_t	Fruit development stage	$-$
FDR_t	Fruit development rate	$-$
FGR_t	Fruit growth rate	kg
$PlantDensity_t$	Plants per square meter	$-/m^2$

Equations

$FDR_t =$	$\alpha FDR + log(T/20) * 0.02131$	$-$
$FDS_{t+1} =$	$FDS_t + FDR_t$	$-$
$FGR_t =$	$TFW * 6,000 * exp(-exp(-4.38 * (FDS_t - 0.397))) * 4.38 * exp(-4.38 * (FDS_t - 0.397) * (\alpha FDR + 0.02131 * log(T/20))$	kg/day
$PlantDensity_t =$	FGW/FGR_t	$-/m^2$

Applying a constant temperature of 21 degrees Celsius to this model provided the following outputs. Figure 4.7 illustrates the calculated plant density, and the corresponding FDS is scaled and added to split the growing stages. This figure was used to determine the plant density. For every stage, the minimum calculated plant density was taken as the starting point. Since, in Stage 1, there was no fruit development, there was no advised plant density. For Stage 1, the plant density was set as equal to 64. This is convenient, because in Stage 2 the minimum plant density was equal to 32. Stages 3, 4, and 5 had a plant density of 14, 9, and 16, respectively. The implementation in Python was conducted by using the scheme in Figure 4.8.

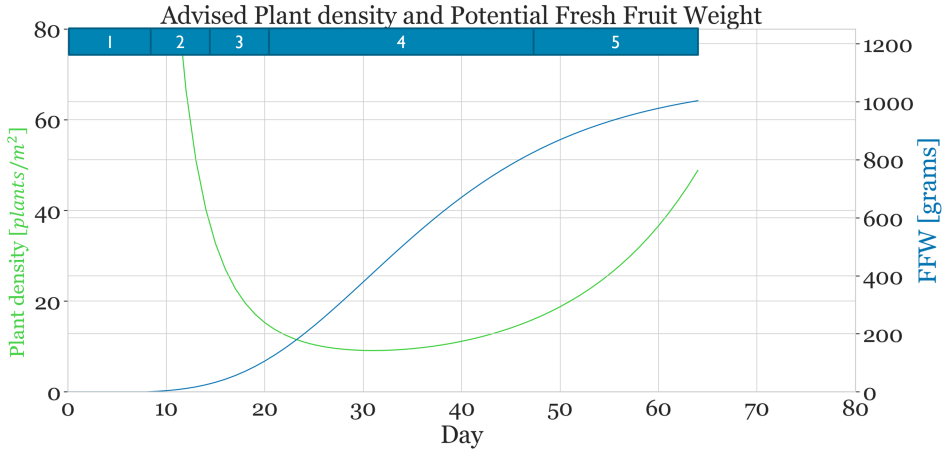


Figure 4.7: Optimal plant density and development of truss weight over the cycle for temperature equals 21 degrees Celsius. Numbers indicate growing stage, FFW = Fresh Fruit Weight (i.e. fruit weight of two trusses). At the top of the figure the growing stages are indicated.

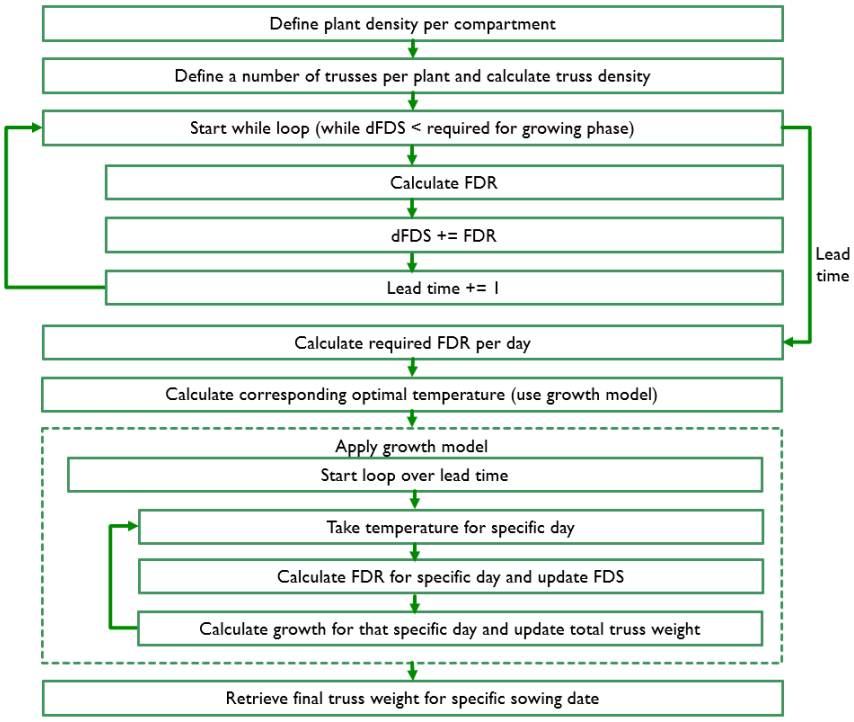


Figure 4.8: Implemented flow in Python for obtaining the optimal plant density for two trusses per plant (see output in Figure 4.7).

The obtained light curve was illustrated in Figure 2.15. With a fixed truss density, balancing the temperature provides unfeasible outputs for the winter period (see Figure 4.9 (right)). The DLI curve varies from roughly 0.5 to 8 $MJ/m^2/day$, which is variation of a factor 16. The growth of a crop is proportional with the DLI curve, so the daily growth is varying with a factor 16 as well. Wit this variable growth per day, the lead time for reaching the desired fruit weight is highly varying as well. Using this varying lead time, leads to unfeasible calculated temperatures (Figure 4.9). his temperature is, in periods with low light, as low as 10 degrees degrees Celsius. The drops towards zero degrees are when there is no growth at all. This profile is as expected, as in practice it is impossible to grow tomatoes during winter with natural light conditions in the Netherlands. In addition, the temperatures for periods with maximum light are unfeasibly high, for which the modelled growth here exceeds the potential growth. As the temperatures are volatile, and they cannot be optimised for all plants in a compartment, this leads to a highly variable truss weight at harvest (Figure 4.10).

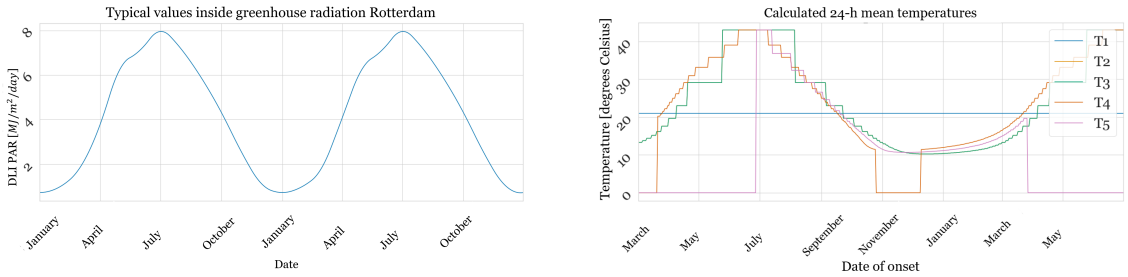


Figure 4.9: Calculated optimal temperatures (right) for constant truss density and given light curve (left). 'T1' means temperature Compartment 1, 'T2' temperature Compartment 2, etc..

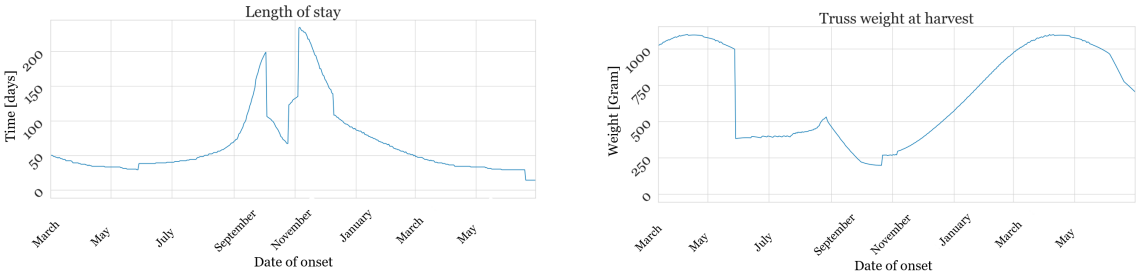


Figure 4.10: Calculated Lead time (left) and final truss weight (right) for Approach 1a for an unbounded input light curve (see Figure 4.9 (left))

One way to resolve this issue is by adding artificial light, up to a threshold, and by using screens to filter out the light, up to a threshold. Setting the thresholds equal to 3 and 4 $MJ/m^2/day$, the temperature just stays between the feasible range for Rotterdam, the Netherlands of 17 to 23 degrees Celsius (Figure 4.13).

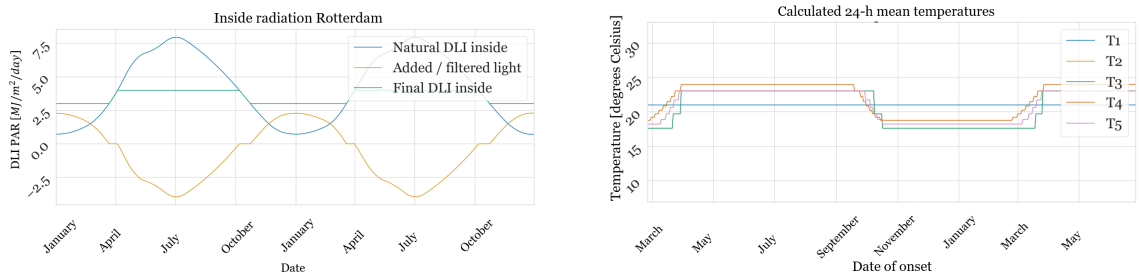


Figure 4.11: Approach 1a: Calculated optimal temperatures (right) for constant truss density and given light curve (left). 'T1' means temperature Compartment 1, 'T2' temperature Compartment 2, etc..

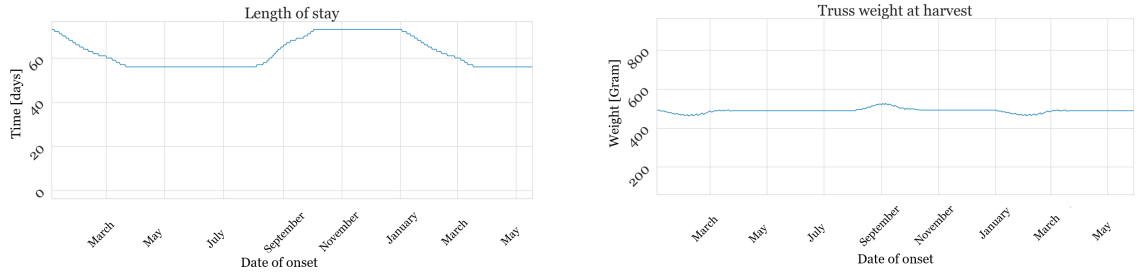


Figure 4.12: Approach 1a: Calculated Lead time (left) and truss weight at harvest (right) for a bounded input light curve (see Figure 4.11)

The temperatures (Figure 4.11) are within the feasible range, but there are high lighting costs during winter, and a loss of light during summer. A better option is to make the number of trusses per crop dynamic. The possible options are one to three trusses per plant. This action means the truss density can vary through the year by a factor three. The assumption is that the thresholds can be released by a factor three as well in that case. Therefore, they are set equal to 2 $MJ/m^2/day$ as lower bound and 6 $MJ/m^2/day$ as upper bound. This leads to the following Figure 4.13.

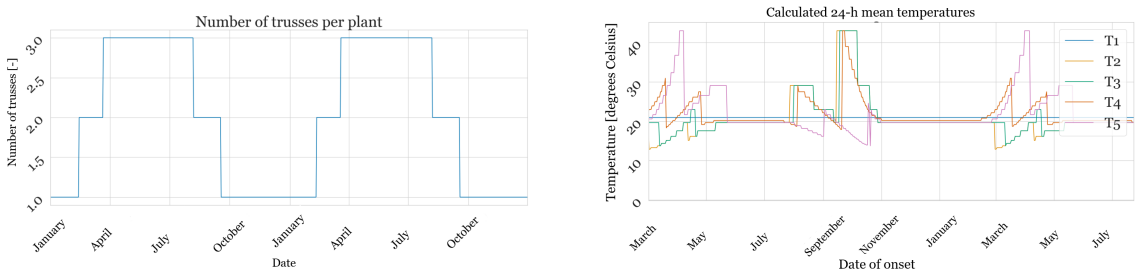


Figure 4.13: Approach 1b: Defined number of trusses per plant (left) and calculated temperatures for light curve of Figure 4.11 (right). 'T1' means temperature Compartment 1, 'T2' temperature Compartment 2, etc..

Figure 4.13 illustrates that downscaling and upscaling the number of trusses is just ahead of the DLI curve. This difference is because the number of trusses is for the day of onset, whereas the DLI curve is for the current day. The temperature data are spiky, often exceeding the feasible range, but this is only for small periods, so this temperature could be cut-off without too much damage. Worrying is the truss weight at harvest (see Figure 4.14). The variations in truss weight are unacceptable when aiming for a standard product to sell. More and smaller compartments could reduce this effect, but it is impossible to remove this effect fully.

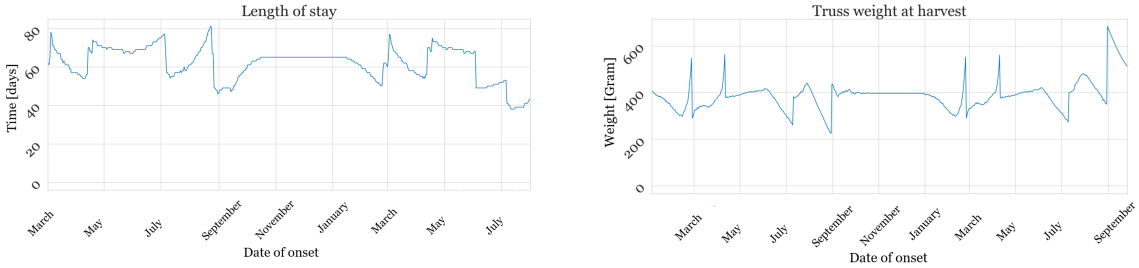


Figure 4.14: Approach 1b: Lead time in system (left) and truss weight at harvest (right)

4.1.2. Approach 2: Calculating the optimal truss density

The previous subsection revealed that determining a temperature for a given light intensity and truss density is not the ideal path for the new cultivation process. Therefore, another approach is examined. The existing model of De Koning (1994) can also be used to determine a truss density for a given light intensity and temperature profile. With the new cultivation concept, the truss density can be changed during the cultivation process. The first test is to make the truss density a continuous variable. Later, the required and feasible number of discrete steps is determined if this approach seems viable.

This approach is schematically indicated in Figure 4.15. Again, this is a flow is converted from Figure 4.4. In first instance an optimal number of trusses per m^2 is calculated. In a next step, it is calculated how this truss density can be obtained.

What is different in this approach compared with Approach 1 is that now Compartment 4 is split into two compartments, as that is where the largest growth occurs. This compartment is larger than the other compartments, and some variable climate conditions might be favourable for this compartment. Another difference between the approaches is that only two or three trusses per plant are allowed. The previous approach used one truss per plant. A single truss per plant is relatively expensive and is therefore undesirable. Moreover, the thresholds of light are slightly changed. For 2 $MJ/m^2/day$, the truss density becomes that low that not all the light is intercepted anymore, which leads to a drop in the LUE, and less growth. This situation leads to undesirably low temperatures and long lead times. Having a lower boundary of 3 $MJ/m^2/day$ seems to prevent that. Additionally, there is a relaxation on the upper boundary. Because, it is undesirable to lose natural light there is no upper bound implemented on light anymore. The implemented calculation scheme is in Figure 4.16.

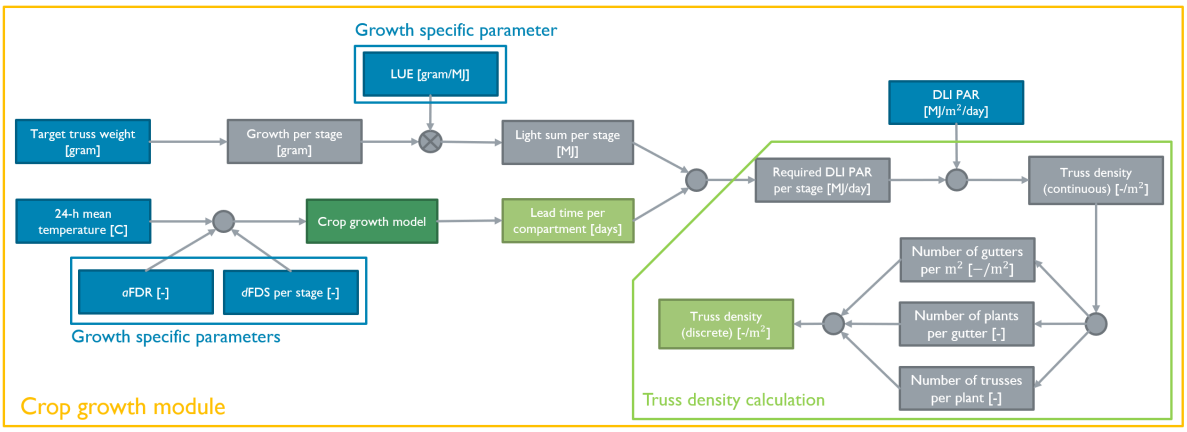


Figure 4.15: Calculation flow for truss density. This is a converted version of Figure 4.4.

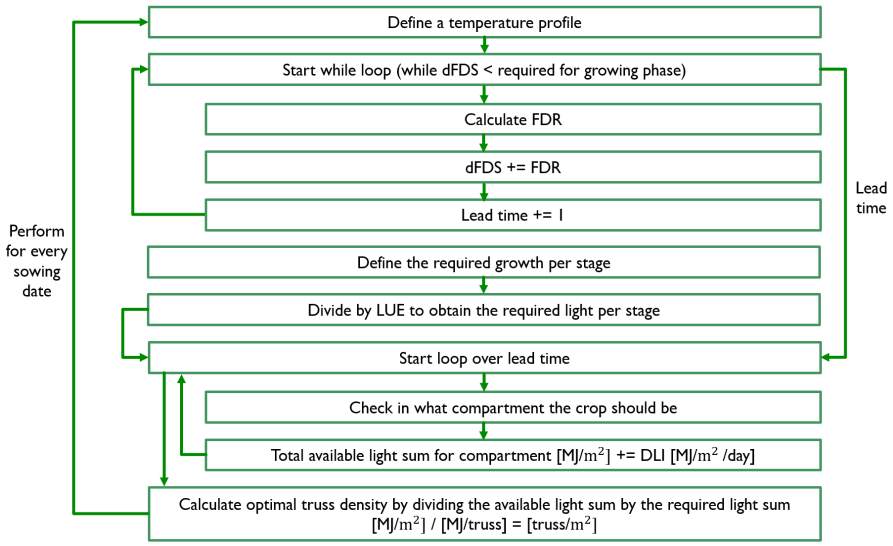


Figure 4.16: Implemented flow in Python for truss density calculation. In correspondence with Figure 4.15

First, a constant temperature of 21 degrees Celsius was used (Approach 2a), and the light curve of 4.17 was employed. The outputs are in Figure 4.18, which reveals that the optimal truss density varies with the light intensity curve, whereas the truss weight at the end of the cycle remains almost constant.

However, although these outputs look neat and desirable, this path is not viable. In a greenhouse, the inside temperature is related to a set of factors as the outside temperature, outside light intensity, and wind (Mesoudi et al., 2010). Because, there are no straightforward relationships available to calculate the inside temperature, the starting point taken is that the inside temperature follows a similar trend as the inside light intensity. Therefore, a scaled curve of the light intensity was used in the next simulation. The 24-h mean temperature and the calculated lead time are provided in Figure 4.19. Then, the corresponding calculated truss density and truss weight at the end of the cycle are presented in Figure 4.20. Due to the discrete variations in the lead time, there are small spikes in the truss density and some variations in the truss weight at the end of the cycle. The deviations in truss weight at harvest are within 5%, which makes it an acceptable output.

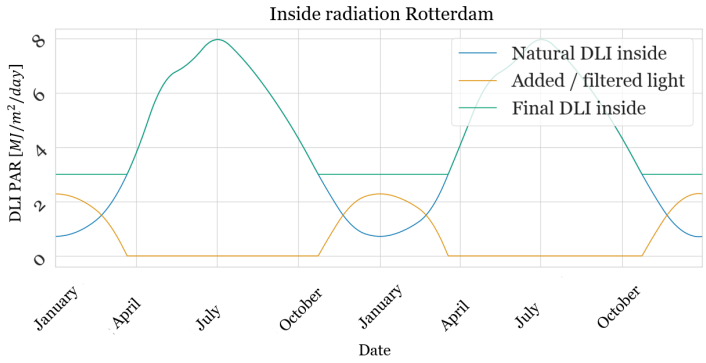


Figure 4.17: Input DLI PAR curve inside greenhouse

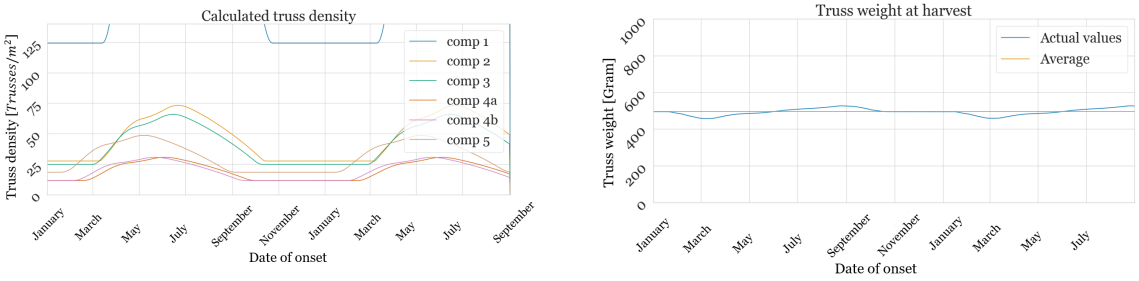


Figure 4.18: Approach 2a: Calculated plant density for day of onset (left) and corresponding truss weight (right), 'comp' means compartment.

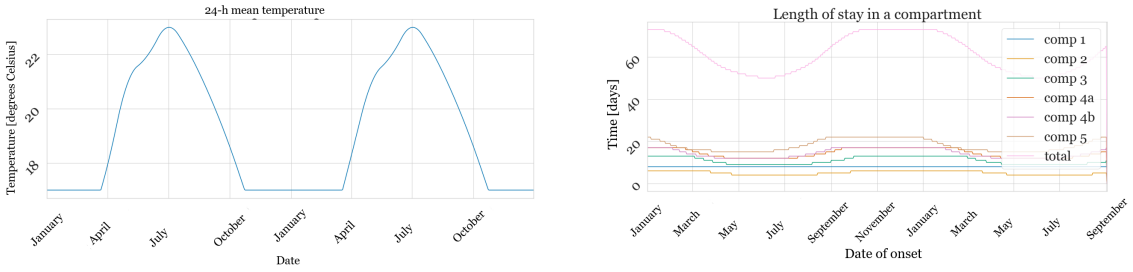


Figure 4.19: Approach 2b: 24-h mean temperature (left) and lead time for day of onset (right), 'comp' means compartment.

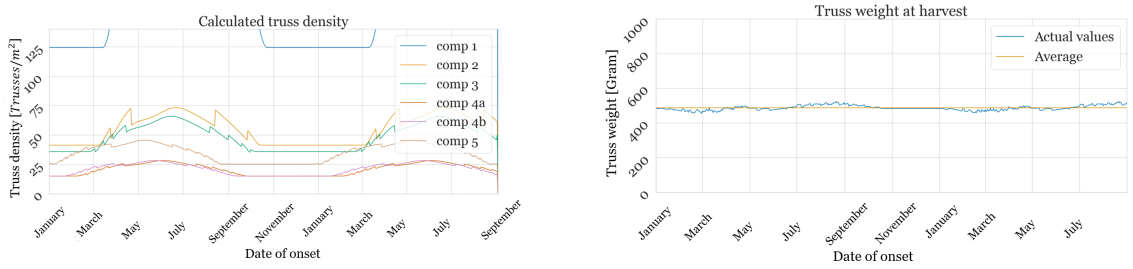


Figure 4.20: Approach 2b:Calculated truss density for day of onset (left) and corresponding truss weight (right), 'comp' means compartment.

As discussed in Section 2.4.3 truss density is determined by a multiplication of three variables: 1) trusses per plant, 2) plants per gutter, and 3) gutters per metre of path. Note that the trusses per plant are constant through the cultivation cycle of a plant. The number of trusses per plant can be varied only at the onset of the process, whereas the others can be varied when the plants move to another compartment. The number of plants per gutter can only be varied if there is a transplant step. This constraint means that the plants per gutter is fixed over Compartments 3, 4, and 5. The largest growth occurs in Compartment 4. Therefore, the number of trusses is based on the closest discrete step to

the desired continuous value for Compartment 4. The starting point for options in truss density are in the next enumeration. Note that not all options are viable. For example, it makes no sense to place the gutters very close to each other with a little number of plants on a gutter. A feasible list is in Table 4.2. What the reduced list of options in truss density per compartment should be, can be explored by using the developed simulation model in this study.

1. Trusses per plant: [2, 3]
2. Plants per gutter: [8, 12, 16, 24, 32]
3. Gutters per metre: [2, 4, 6, 8, 10, 12]

Table 4.2: Overview of the feasible truss densities. The truss density is the multiplication of the first three columns

$Trusses/plant$	$Plants/gutter$	$Gutters/m$	$Trusses/m^2$
2	8	2	4
2	8	4	8
2	8	6	22
2	8	8	16
2	12	4	12
2	12	6	18
2	12	8	24
2	12	10	30
2	16	4	16
2	16	6	24
2	16	8	32
2	16	10	40
2	24	6	36
2	24	8	48
2	24	10	60
2	32	6	48
2	32	8	64
2	32	10	80
3	8	2	6
3	8	4	12
3	8	6	18
3	8	8	24
3	12	4	18
3	12	6	27
3	12	8	32
3	12	10	45
3	16	4	24
3	16	6	36
3	16	8	50
3	16	10	60
3	24	6	54
3	24	8	72
3	24	10	90
3	32	6	72
3	32	8	96
3	32	10	120

This input is implemented in Python in accordance with Figure 4.21 The nearest discrete option to the calculated continuous value is searched for Compartment 4. Then, the corresponding number of trusses per plant is saved (i.e. fixed), with a preference for three trusses over two trusses per plant for similar truss densities. Furthermore, the number of plants per gutter is saved (i.e. fixed). If there are multiple options, the highest number of plants per gutter is chosen. By using the fixed chosen number of trusses per plant and plants per gutter, for every compartment the best number of gutters per metre is selected truss density for every compartment is selected.

Running this script when using the calculated truss density of Approach 2b led to a truss density and final truss weight, as presented in Figure 4.23. There are more spikes in the truss weight for Approach 2c than for Approach 2b due to the discrete truss densities. However, the output is still considered within the acceptable range. If fewer variations in truss weight are desired, more possible discrete steps for truss density should be added to the design.

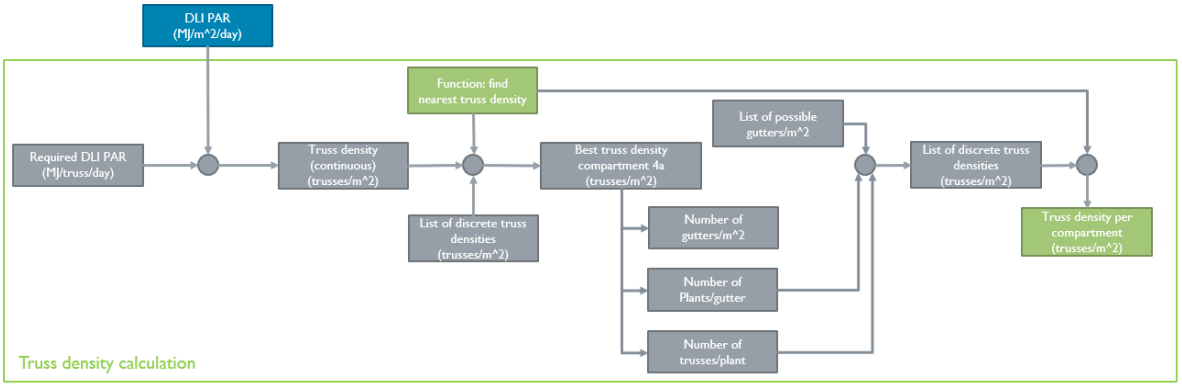


Figure 4.21: Implemented calculation flow optimal truss density. Green frame corresponds to Figure 4.15.

Approach 2 is clearly better than Approach 1. From the different approaches in 2, 2c is the most realistic, as the inputs are closest to the reality. The implemented procedure for calculating the truss density was schematically presented in greater detail in Figure 4.21.

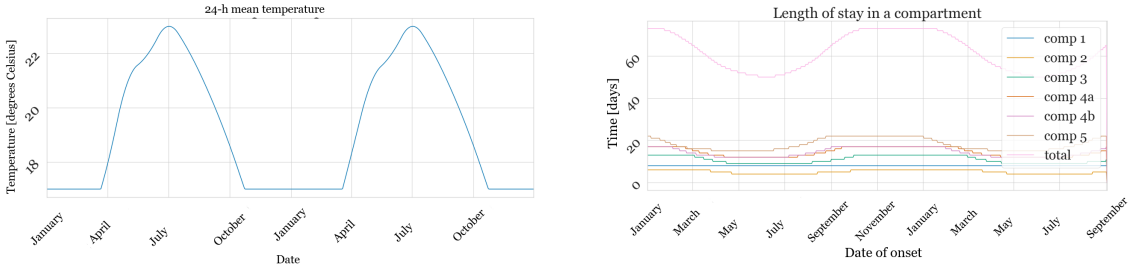


Figure 4.22: Approach 2c: 24-h mean temperature (left) and lead time for day of onset (right). 'Comp' means compartment

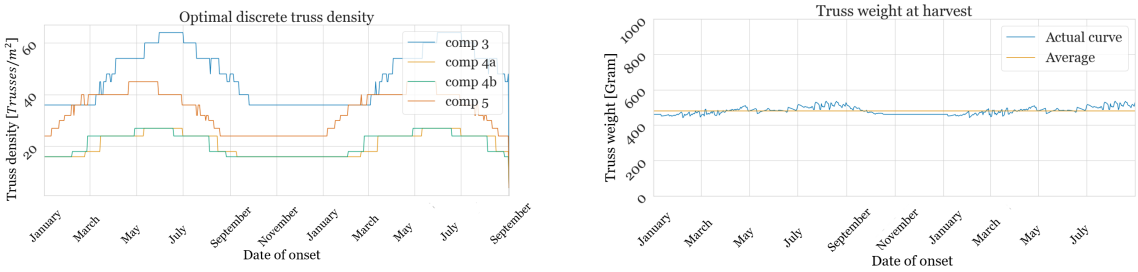


Figure 4.23: Approach 2c: Calculated truss density for day of onset (left) and corresponding truss weight (right). 'Comp' means compartment

4.2. Development of the case-based design-generating module

This section discusses how the conceptual designs should be generated. A case-based design-generating model is developed and verified. An overview of the part of the model in which the conceptual designs are generated is presented in Figure 4.24. Then, the model is mathematically formalised and implemented. The first step in this formalisation is to define the indices and sets.

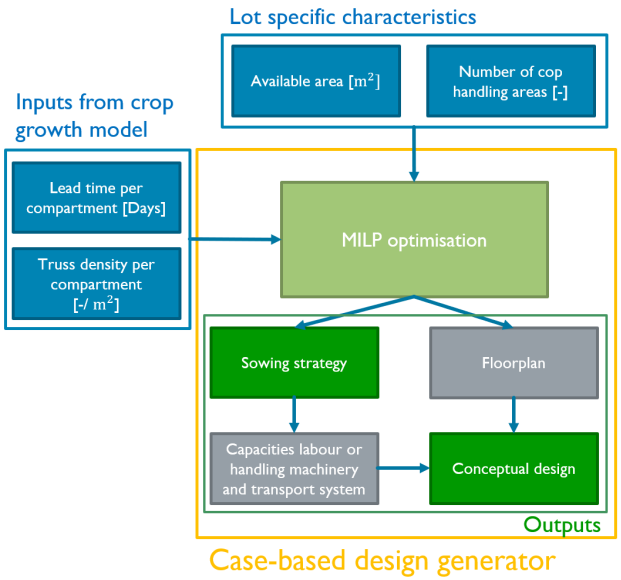


Figure 4.24: Schematic overview of the case-based design generator.

4.2.1. Implementation

Indices

i	Day number
j	Number of rows of crops
k	Compartment number

Sets

I	0,1..,730 Set of days for two years
J	1,2,..,50 Set for the number of rows
K	0,1..,5 Set of numbers for compartments

The second step is to define the parameters, variables, and decision variables. This process begins with a variable for the number of days a crop is in a compartment. There is also a parameter that represents a price curve, which is one input of the models. There are two parameters for the area allocation: the total available area and the area needed for a single row and a help parameter that defines that a new compartment may only start at a new row. The target final truss weight at harvest is also given as an input here. The final parameter is the truss density per compartment.

The variables are the capacity of a compartment and a binary variable used to determine the number of rows per compartment. These variables are linked to the decision variable for the area allocation. The other decision variable is the number of plants to be sown.

Parameters

LOS_{ik}	Integer	Days in a compartment for sowing day i in compartment k
P_i	Continuous	Selling price tomato for a sowing day i
AT	Integer	Total available area for cultivation in greenhouse compartments
AR_k	Integer	Area per number of rows for compartment k
TW_{ik}	Continuous	Truss weight at the end of the cycle at compartment k for sowing day i
R_k	Integer	Truss density for compartment k
$Trusspp_i$	Integer	Trusses per plant for day of onset i

Variables

S_{jk}	Binary	Help variable to select the area per compartment from a list

Decision variables

A_k	Integer	Area per compartment k
x_i	Integer	Number of plants sowed at day i

There is optimised (i.e. maximised) for turnover. This maximisation implies that the price is multiplied by the truss weight at harvest times the number of sowed plants. The sum is the total turnover. The price is volatile, and therefore it can be difficult to determine a realistic input for this price. The production is, therefore, a more reliable measure for the performance than the turnover is. By setting the price curve as a constant or a step function, there will be optimised for maximum production. This is a choice the user of the model can make.

Objective function

$MAX \sum_{i \in I}(x_i * P_i * TW_{i,4})$	Turnover maximisation

The constraints of the model are now defined. The first constraints are there to ensure the decision variable cannot be negative.

$A_k \geq 0$	$\forall k \in K$	Non-negativity constraint
$x_i \geq 0$	$\forall i \in I$	Non-negativity constraint

Then, the areas of the different compartments together cannot exceed the total available area.

$\sum_{k \in K} A_k \leq AT$	Constrain the total available area

The third constraint links the area of a compartment to the number of cultivated crops.

$C_K = A_k * R_k$	$\forall k \in K$	Link the area to the number of plants per compartment

The next set of six constraints limits the number of plants sowed. It is necessary to ensure there is space in the greenhouse for every crop in the entire process.

$x_i / Trusspp_i / plantdensity_{k,i}$ $A_k / LOS_{i,k}$	$\leq$	$\forall i \in 0, 1, \ldots, 650; \forall k \in K$	Limit sowing by capacity per compartment
$x_i = 0$		$\forall i \in 650, 651, \ldots, 730$	No sowing for the last 80 days

The final two constraints concern the area per compartment, which involves the discrete steps for area per compartment.

$\sum_{j \in J} S_{jk} = 1$	$\forall k \in K$	Per compartment only 1 of the number of row option selected
$(S_{jk} == 1) >> (A_k = AR_k)$	$\forall k \in K, \forall j \in J$	Select area per compartment

The described case-based design generator is implemented using the Gurobi software package for Python (Gurobi Optimisation, LLC, 2021). Gurobi Optimisation, LLC claims to be the leading software package in performance (speed of the solver) for MILP models. The structure of the case-based design generator is presented in Figure 4.25.

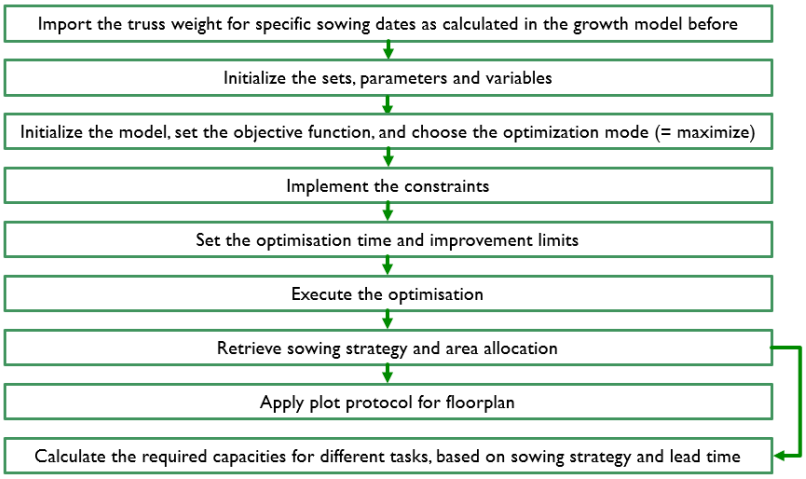


Figure 4.25: Implemented flow in python for the case-based design generator.

4.2.2. Verification

Verification was conducted by executing the checks stated in Table 4.3. Different verification tests are conducted. First, there is the balance test, which checks whether the inflow equals the outflow over the entire simulation window. This should be the case as the greenhouse begins and ends empty. Second, a degeneracy test is done. In this test, there is checked if the model stops when if no system at all is present (i.e. there is no area available). Third, the consistency is checked. If there is a change in the inputs, there should be a proportional change in the output. Fourth, the continuity is checked. In this check, a very small change in the input, should lead to a negligible change in the output. The scenario that is implemented as base for the verification is a 300 [m] x 150 [m] lot size (which is around average for Dutch nurseries (CBS, 2018)), with one crop handling area. The climate conditions are typical values for Rotterdam, the Netherlands (Vermeulen, 2016), the approach taken is Approach 2C. The cultivar and growth dependent parameters are in correspondance with De Koning (1994).

Table 4.3: Verification of the case-based design-generator

No.	Check type	Description	Hypothesis	Result	Pass/Fail
1	Balance	Input minus output check	Increase at the beginning, constant in the middle, going back to zero at the end	Starting at zero, increasing, constant, decreasing towards zero	Pass
2	Degeneracy	Area is set to 0	No capacity in greenhouse, no sowing; error	Error: division by zero	Pass
3	Consistency	Area is doubled	Average number of crops sowed per day slightly more than doubled	Increase from 8,640 to 18,360	Pass
4	Consistency	Truss density is decreased by 10%	Number of plants in greenhouse 10% less	from 457,116 to 411,441 plants present (10% decrease)	Pass
5	Consistency	Temperature increased by 2 degrees degrees Celsius	Increase average in number of plants sowed	From 8,640 to 10,308 per day	Pass
6	Consistency	Temperature decreased by 2 degrees degrees Celsius	Decrease in average number of plants sowed	From 8,640 to 7,079 per day	Pass
7	Consistency	Temperature variable	Variations in number of plants sowed, fewer for a longer lead time	Lower point at desired moment	Pass
8	Consistency	Limit the sowing capacity to 6,000 per day	The number of plants sowed will be lower, with a maximum of 6,000 per day	The upper boundary of 6,000 is observed	Pass
9	Consistency	Increase in price by 20%	Objective function increases 20%	From $1.77 * 10^6$ to $2.12 * 10^6$ (+20%)	Pass
10	Continuity	Increase width of greenhouse by 1 [m]	No significant changes	The same number of plants present in greenhouse	Pass
11	Continuity	Increase plant density by 0.1	No significant changes	From 457,116 to 457,801 plants present (0.15% increase)	Pass

4.3. Logic of the simulator

After generating the conceptual models and the required inputs, it is time to simulate the configurations for multiple scenarios. As discussed in Section 3.3, an OO-DES simulator was used. A schematic overview of this part of the model is in Figure 4.26. An activity diagram of the OO-DES block is given in Figure 4.27.

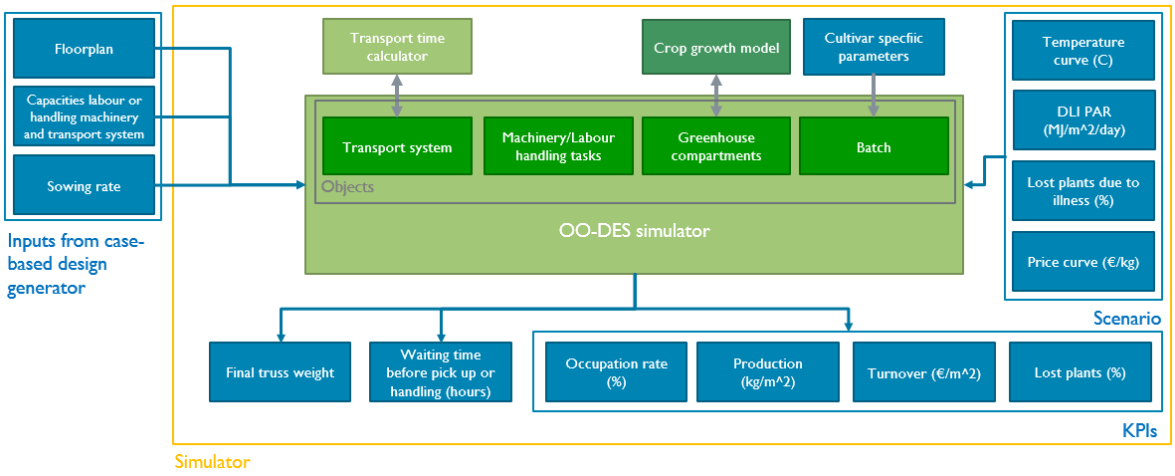


Figure 4.26: Schematic overview of the simulator part of the model.

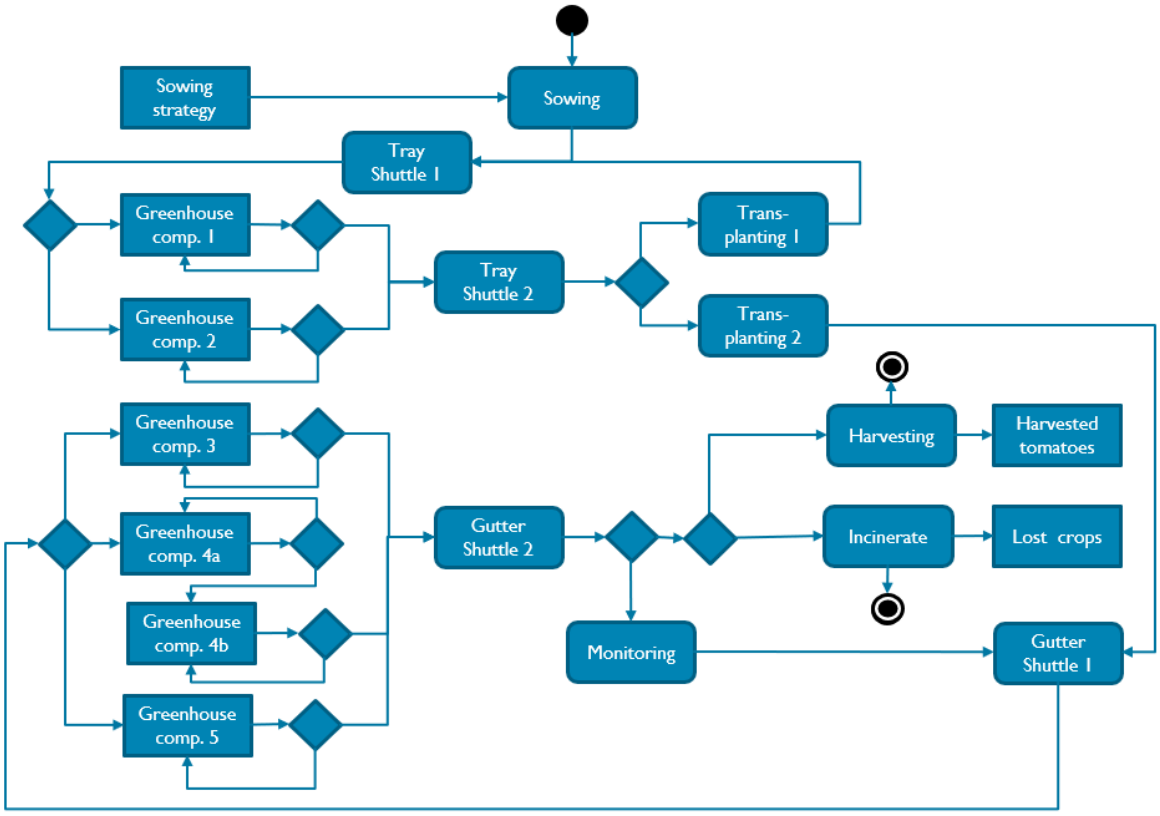


Figure 4.27: Activity diagram of the new concept (simplified).

4.3.1. Implementation

The simulation model is implemented in Python by using the Salabim package. The full code is presented in Appendix C, whereas the main parts are highlighted in this section. In addition, a pseudo code that can be implemented in any language is provided in Appendix B. The first step in the simulator is defining the smallest object for simulation. There are a few options for smallest object, from small to large: 1) the truss, 2) the plant, 3) the cultivation medium (i.e. tray and gutter), and 4) the batch. Generally, a smaller object means more objects in the simulation model (i.e. there are more trusses than plants), which implies a slower simulation. The first three object choices are related to each other, as a truss is carried by a plant, and a plant is carried by a cultivation medium. Therefore, choosing for Options 1 or 2 over Option 3 does not contribute to the accuracy, while these options do affect the performance of the model negatively.

Therefore, the approach of having the cultivation medium as smallest object could be a logical choice. However, in practice, a grower will not sow a single tray; there will always be a minimum batch size. Another complication of choosing a cultivation medium as the smallest object is that the cultivation medium changes in the process from a tray to a gutter. Additionally, the number of plants per cultivation medium varies according to different onset days. Furthermore, for the different compartments, the distances between the gutters are changed. Thus, the required areas for an object change per growing stage. This different gutter distance could be implemented as an assigned attribute to a medium. However, a more generic method for dealing with varying gutter distances is taking a batch as the smallest object. For that batch, at least, the following attributes are assigned: the number of plants, the truss density, the number of cultivation mediums, number of plants per medium, and the number of trusses per plant. An advantage of this approach is that it is quick, due to the smaller number of objects in the simulation. A disadvantage is that when the number of plants per batch is a given, rounding errors are induced; it is not the calculated number of plants that is sowed, but the closest value to a multiplication of the batch size. Additionally, unused places can remain in a cultivation medium when the number of plants in a batch is not divisible by the number of plants per medium. Setting the batch size to 500 seems a feasible number. The calculated average sowing rate for the Netherlands for a 300 [m] x 150 [m] greenhouse (i.e. average size Dutch greenhouse) is 8,640 plants/day, which makes 500 just over 5% of the total, whereas 16 plants per gutter is only just over 3% of 500. Therefore, a batch size of 500 is chosen as the best starting point; however, this batch size can vary per geographic location and greenhouse size.

In addition to the mentioned attributes, an FDS and truss weight are implemented. The FDS is used to keep track of the growing stage, whereas the truss weight is assigned to obtain a final truss weight at harvest. Furthermore, some variables keep track of the process, including the day, the timer, and the lead time. Finally, a process is assigned to the batch when there are plants lost due to illness. Note that it is an option to implement the number of plants lost due to illness in the batch, but that it can, for example, also be implemented in machinery in the crop-handling area. The definition of the batch class is in Figure 4.28.

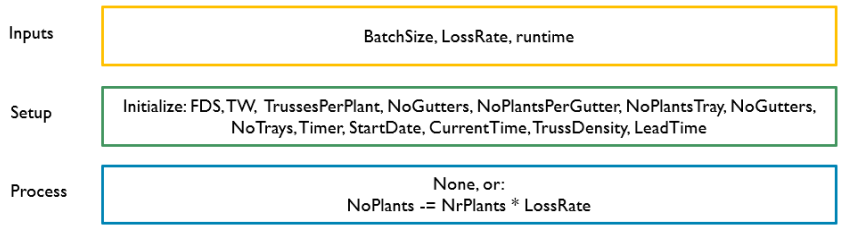


Figure 4.28: Definition of the class Batch

The sowing machine is now defined. This is the onset of the process. First, the variables are initialised, and the parameters are loaded into the object. Then, the process is defined. The process

starts by looping over the runtime and keeping track of the number of days. Subsequently, the cumulative number of plants that should have been sowed according to the case-based design generator is calculated. Next, the number that should be sown today to achieve that number is calculated. This number is divided by the batch size to calculate the number of batches to be sown. This number is rounded down, and the remaining number of plants that should be sowed is added to the batch for the following day. The next step is to calculate the time needed to sow a batch. There is a sowing time per tray, so the number of trays should be calculated and multiplied by the sowing time per tray. This time is used to calculate the maximum number of batches that can be sowed that day. If the calculated number of batches exceeds the maximum number, the maximum number of batches is sowed. A warning is given that the desired capacity exceeds the existing sowing capacity, so the sowing capacity can be updated.

This stage is followed by looping over the number of batches. The batches are created, and the attributes are assigned. They are held in the machine for the calculated sowing time. The batch is placed in a queue, where it is ready to be transported to the greenhouse. The final step, after creating the batches, is to update the time tracker. The time left in a day (i.e. the machine is not used) is waited for, so the new day starts at time unit zero for the next day.

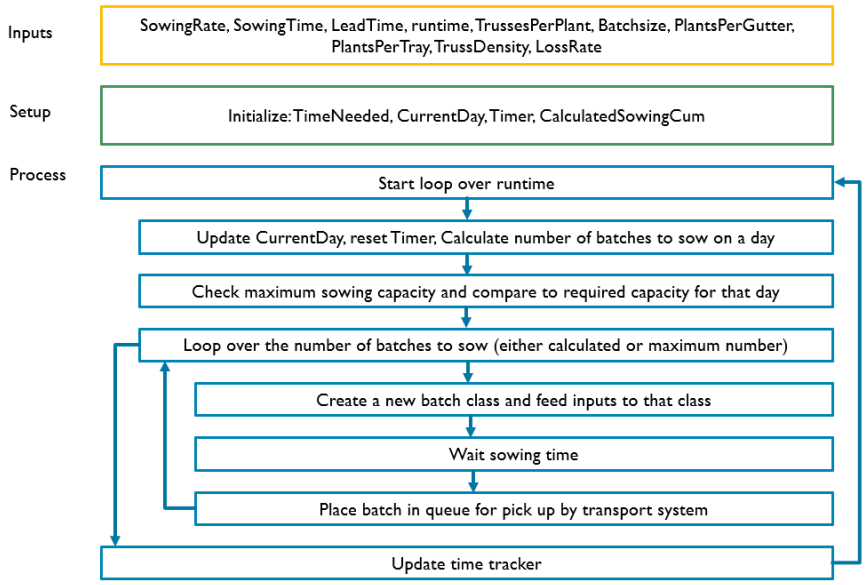


Figure 4.29: Definition of sowing generator class

For the case study, shuttles pick up the batch. There are shuttles on the market able to transport crops: for example, the 2Dshuttle of Logiqs b.v. (Logiqs, 2021). However, it may be necessary for the new cultivation concept to redesign the shuttles. Four different shuttles are defined: two for transporting the trays and two for transporting the gutters. One of the two shuttles transports the cultivation medium from the greenhouse to the crop-handling area, whereas the other transports from the crop-handling area to the greenhouse. Again, the details of the design of the shuttles should be developed in a later study. Therefore, in the simulator, there is freedom to change the characteristics of the shuttles. First, the dimensions can be changed. Second, the characteristics regarding capacity per transport action, velocity, and acceleration can be changed. The velocity and acceleration result in a drive time. This drive time is implemented as an attribute in the shuttle and can also be retrieved and updated via a function for every new transport action.

The shuttle that transports gutters from the crop-handling area to the greenhouse is the described example, but the others are similar. The parameters and variables are loaded in and defined. The drive time is loaded as an array. The drive time varies per compartment, but is fixed over the run, and

fixed for a given compartment number to transport to. The next step is to define the process of the shuttle.

A decision tree is implemented for the shuttle. The decision tree for the greenhouse to the crop-handling area is presented in Figure 4.30, and for the crop-handling to the greenhouse in Figure 4.31. The greenhouse-to-crop-handling-area decision tree begins by checking whether the batch is in the right compartment. It is checked whether the batch is temporarily stored in another compartment, as perhaps there was no space in the desired compartment. If the batch was in the wrong compartment, it may be ready for the next stage, and therefore needs to pass to the crop-handling area. The other case is that it is not. If it is not ready, another attempt is made to store it in the desired compartment. The acceptable compartments are the next and previous compartment. The boundary condition is that the cultivation medium (either tray or gutter) must be the same for storing the batch in another compartment.

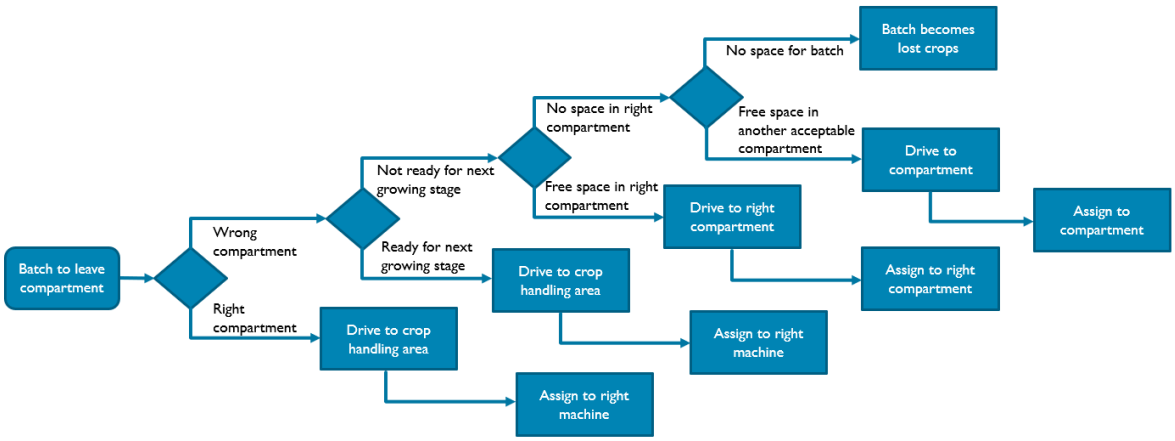


Figure 4.30: Decision tree for the shuttle for transport from a greenhouse compartment to the crop-handling area

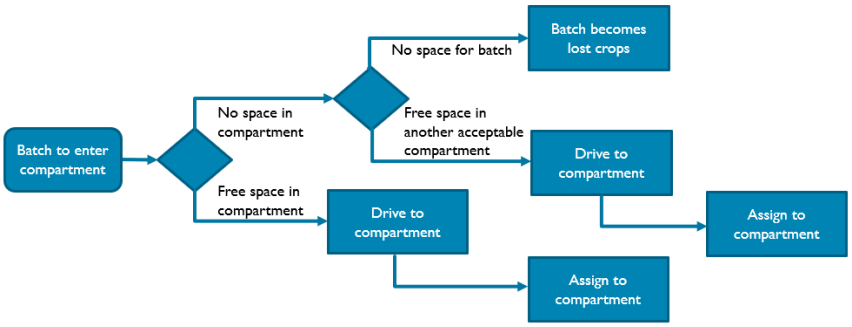


Figure 4.31: Decision tree for the shuttle for transport from crop-handling area to a greenhouse compartment

The next stage is defining the greenhouse compartments. Only one compartment is defined here, as the others are similar. An aberrant compartment is Compartment 4a. Instead of going to the crop-handling area, 4a goes directly to Compartment 4b, which is a similar compartment for the same stage. The reason for the split is that the climate conditions can vary for crops with different characteristics (i.e. match temperature better regarding light intensity and truss density).

The process of a greenhouse compartment is visualised in Figure 4.32. First, there is a loop started over the entire runtime. Then, the number of batches present in the compartment is considered and there is looped over these batches. These batches are taken, and the attributes are updated. Most

important, the FDS and truss weight are updated. The batch is checked for whether it is in the correct compartment. If it is not, it is placed in the queue to leave the compartment. Subsequently, the FDS is used to check whether the batch can proceed to the next stage or should stay in the compartment. The increase in truss weight is calculated for a day and added. At the end, a day is added to the time tracker and the occupancy rate of the compartment at the end of the day is saved.

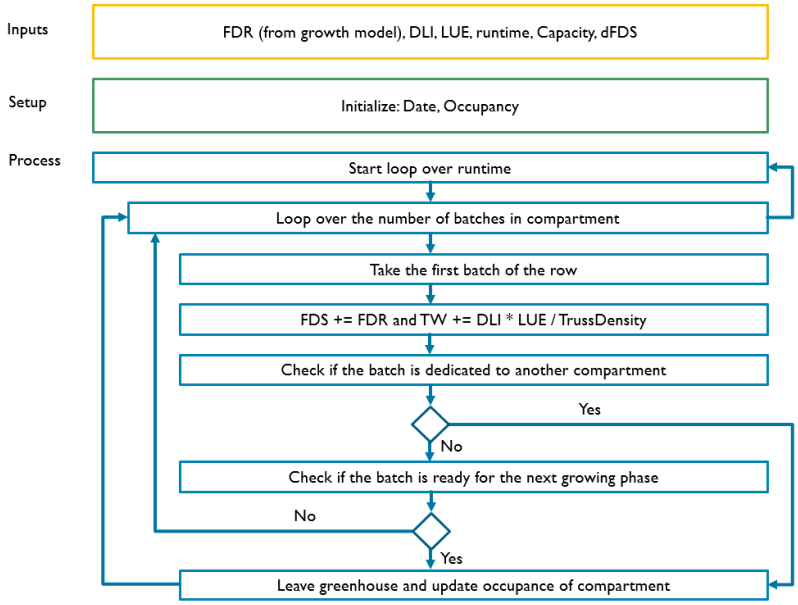


Figure 4.32: Definition of a greenhouse compartment class

The activity diagram (Figure 4.27) reveals a set of crop-manipulation tasks to be performed. All the manipulation tasks are defined in the pseudo-code in Appendix B and the full code in Appendix C, one of which was arbitrarily chosen to be highlighted in Figure 4.33. The transplant machine for the second transplantation step is presented. The time handling for a single tray in a batch and the loss rate are loaded. In the process, the machine checks whether there is a batch waiting for the handling. If not, then the machine is held on standby. If yes, the tray is handled. How long the batch has been waiting is then checked. If the wait is more than six hours, the batch is considered lost. Otherwise, the batch is held for the time all trays can be handled, and the timer is restarted to time how long it takes before the tray is picked up.

With all the objects defined, it is time to start the simulation. First, the environment is initialised with the ‘Environment’ command. The time step and random seed are defined. Then, all the queues are defined and initialised. The objects are then initialised and the relevant parameters and variables are fed to these objects. Finally, the run environment is run via the command ‘Environment.run’. The simulation monitor and the plots are defined as well.

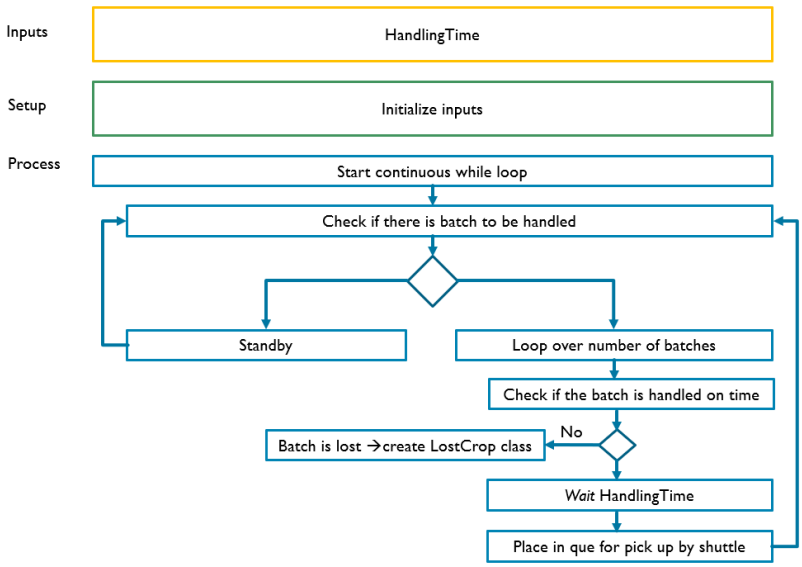


Figure 4.33: Definition of a transplant machine class

4.3.2. Verification

The verification process was conducted by executing the checks stated in Table 4.4. Different verification tests are conducted. First, there is the balance test, which checks whether the inflow equals the outflow over the entire simulation window. This should be the case as the greenhouse begins and ends empty. Second, a degeneracy test is done. In this test, there is checked if the model stops when if no system at all is present (i.e. there is no area available). Third, the consistency is checked. If there is a change in the inputs, there should be a proportional change in the output. Fourth, the continuity is checked. In this check, a very small change in the input, should lead to a negligible change in the output. The scenario that is implemented as base for the verification is a 300 [m] x 150 [m] lot size (which is around average for Dutch nurseries (CBS, 2018)), with one crop handling area. The climate conditions are typical values for Rotterdam, the Netherlands (Vermeulen, 2016), the approach taken is Approach 2C. The cultivar and growth dependent parameters are in correspondance with De Koning (1994).

Table 4.4: Verification of the simulator

No.	Check type	Description	Hypothesis	Result	Pass/Fail
1	Balance	Check if the inflow equals the outflow	The number of sowed plants equals the number of lost plants and harvested plants	3,171,678 sowed, 60,006 lost, 3,111,672 harvested	Pass
2	Degeneracy	Area is set to 0	No capacity in greenhouse, no sowing; error	Error: division by zero	Pass
3	Consistency	Increase the plants lost due to illness to 10%	The production will drop by 10%	The production drops from 63.8 kg/m^2 to 57.4 kg/m^2	Pass
4	Consistency	Limit the sowing capacity to 6,000 per day	The occupancy and production will be about 6% lower, because the average sowing rate is about 5% than the maximum	Occupancy rate drops from 85.7% to 77.6%, whereas the production drops from 63.8 to 56.7 kg/m^2, the number of plants lost approaches 0	Pass
5	Consistency	Increase in price by 20%	Turnover per m^2 increases 20%	From €76.60 to €91.80 (+20%)	Pass
6	Continuity	Increase width of greenhouse by 1 [m]	No significant changes	Changes in KPIs <1%	Pass
7	Continuity	Increase plant density by 0.1	No significant changes	changes in KPIs < 1%	Pass

4.4. Conclusion

In this chapter, the central was the third sub-question: *How should the new concept be implemented in a simulation model?* To answer this question, a simulation model that can be used to generate and simulate conceptual designs was developed. First, the crop growth was modelled. The relationships employed in a commercially used model were implemented, but the approach taken was different. In this new approach the optimal truss density is calculated as output. This truss density is input to the next part, the case-based design generator. In this part, the designs were generated by using an optimisation model. A sowing strategy was also calculated, and a starting point for the capacities of the transport system and the crop-handling tasks. These factors can be implemented in the simulator, in which the growth and location of the tomato plants in the greenhouse are simulated. The inputs of this simulation model can be changed to examine different cases or scenarios. The verification proves the model behaves as expected, whereas the validation and experimentation are left to follow-up studies. This developed simulation model is in accordance with the structure in Figure 3.1. The structure of the simulation model is in greater detail provided in Figure 4.34. The next chapter will use the developed simulation model to perform the evaluation protocol.

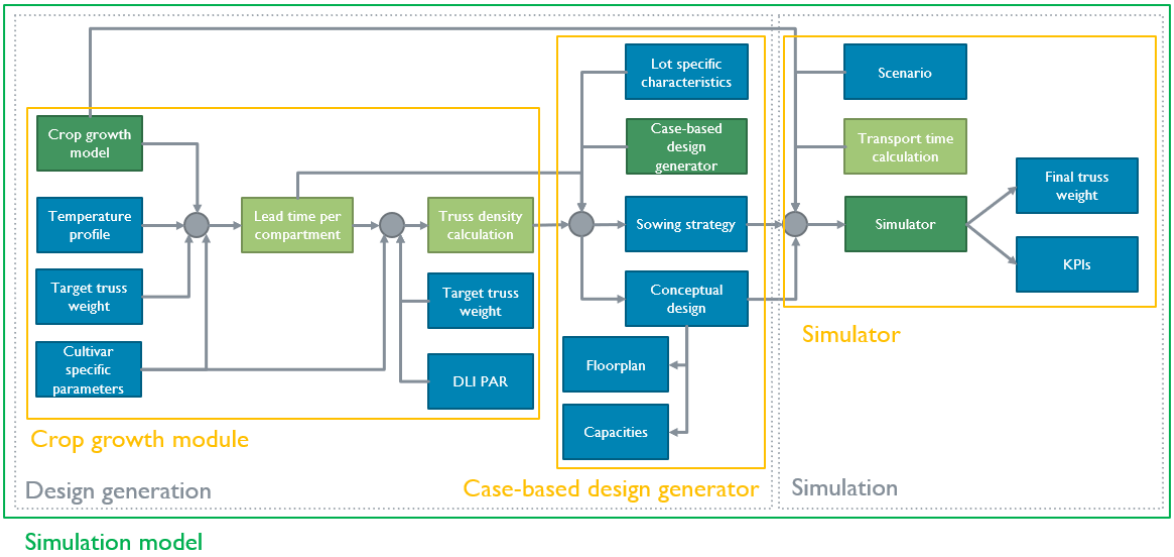


Figure 4.34: Flow of the simulation model in greater detail than the global overview in Figure 3.1. The inputs and outputs are in blue, while the three modules are in darker green.

5

Results

This chapter presents and evaluates the results obtained by employing the evaluation protocol (Figure 5.1). This protocol represents the evaluation block in the developed design framework (Figure 3.1). The blocks in Figure 5.1 are discussed in this chapter, except for the first block, which is already examined in Section 4.1. The sensitivity analysis and optimal dimension analysis are performed for a base case to identify relationships (Section 5.1 and 5.2). For the other three blocks in the evaluation protocol, three case studies are applied (Section 5.3). The three case studies, with different characteristics, serve for proving that the framework developed in this study is generic. The results obtained alongside by employing the evaluation protocol are also interested and discussed in Section 5.4. Performing the different steps in the evaluation protocol helps to answer in Section 5.5 the fourth sub-question: *How can the conceptual design be evaluated, and the simulated performance of the conceptual design be improved?*

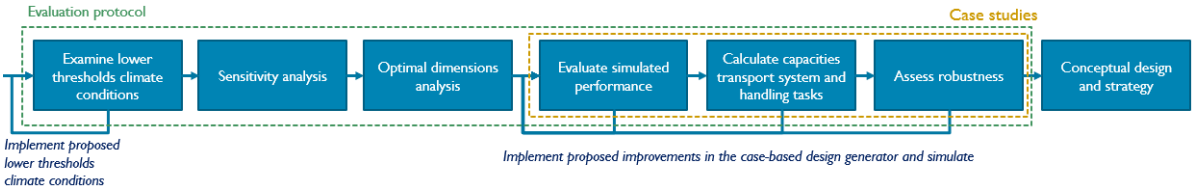


Figure 5.1: The developed evaluation protocol schematically

In the evaluation protocol, KPIs play a central role in examinations. The KPIs should be measurable, comparable, specific and relevant for decision-making. The first KPI is the turnover per m^2, with one m^2 being one m^2 of net area used for crop growing. The KPI turnover is considered most relevant for decision-making, as this is an important factor for the investor's profit. Related to that is the KPI that calculates the production per m^2. This KPI is more fairly comparable with other concepts as this KPI is not dependent on a price curve which could be volatile. Therefore this production KPI is most important for comparing results. Another KPI is the mean occupancy rate of the greenhouse over a year. Furthermore, a KPI is the percentage of crops lost due to an over-occupation of the greenhouse, the transport system or handling tasks. These two KPIs are relevant for operational cost and efficiency, which is also important in decision-making. In addition, the waiting times before a transport pick up or handling are measured in the fifth evaluation block. Final, the spread in truss weight at harvest is reviewed, which is not a performance indicator but a boundary condition in the operational phase. For different cultivars, there are differing needs for uniformity of the product.

The sensitivity analysis and optimal dimension analysis are performed with typical climate conditions for Rotterdam. The reason the choose for Rotterdam is that for that region there is widely used and validated data of Vermeulen (Vermeulen, 2016) available. Moreover, this is one of the regions with highest tomato production. The relationships found here are generic for multiple cases. The two

typical climate conditions that are referred to are temperature (24-h mean [degrees Celsius]) and light intensity (DLI PAR [$MJ/m^2/day$]) (see Figure 5.2 for inputs and Subsection 4.1.2 for an explanation). The cultivar-dependent parameters are constant and in accordance with De Koning (1994). No stochastic elements are added to the input signal, which keeps both the input and the simulation model fully deterministic. After the sensitivity analysis and optimal dimension analysis, there is worked with three other datasets to fulfill the next steps of the evaluation protocol.

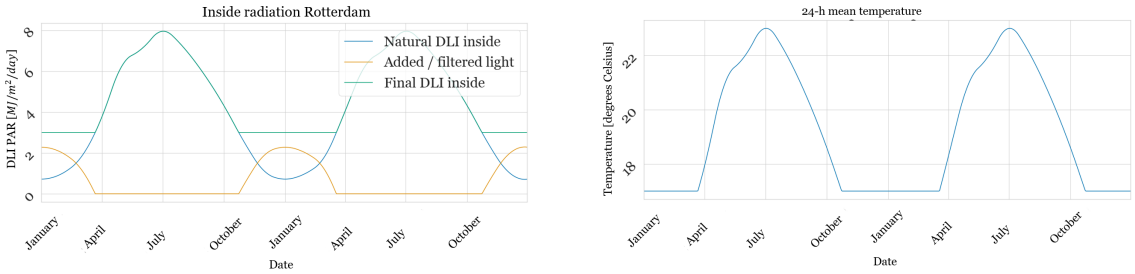


Figure 5.2: Input light intensity and temperature curve (typical values for Rotterdam, also see Subsection 4.1.2)

5.1. Sensitivity analysis

A set of methods was available to conduct a sensitivity analysis. Two equations that are often central in these methods are 1) $||\delta y/\delta x||/||y||$ (Simske, 2019), and 2) $S = (\delta x/x)/(\delta y/y)$ (Bhuvanagiri et al., 2018). Equation 1) calculates the partial derivative compared with the norm of the dependent variable. This provides factors for the dependency of a relative magnitude. The second equation sets the sensitivity equal to the change of state variable x compared with the change in parameter x. Often, a change of $\pm 10\%$ and $\pm 20\%$ is subjected. In the sensitivity analysis for this study, a sample lot was established. An area of 300 x 150 [m] (4.5 ha) is used as a case for this study (see Figure 5.3), which is around the average area for Dutch nurseries in 2017 (CBS, 2018). The mentioned inputs in Figure 5.2 are the reference inputs for this analysis.

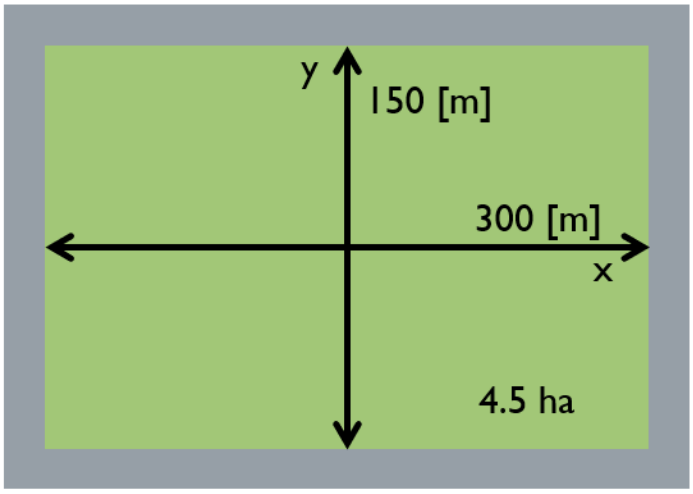


Figure 5.3: The area of the case study

The case-based design generator optimises both the the sowing strategy and the area allocation for turnover maximisation. Therefore, the measuring variable is the turnover per m^2 for one representative year, in which one m^2 is one m^2 net growth area. Simulating with the typical values (Figure 5.2) leads to a reference conceptual design and corresponding turnover visualised in Figure 5.4. Note that this conceptual design is not improved in the evaluation protocol yet. Therefore, it could be that better

performance is observed later. At this stage, the final turnover equals 76.9, euro per m^2 net growth area.

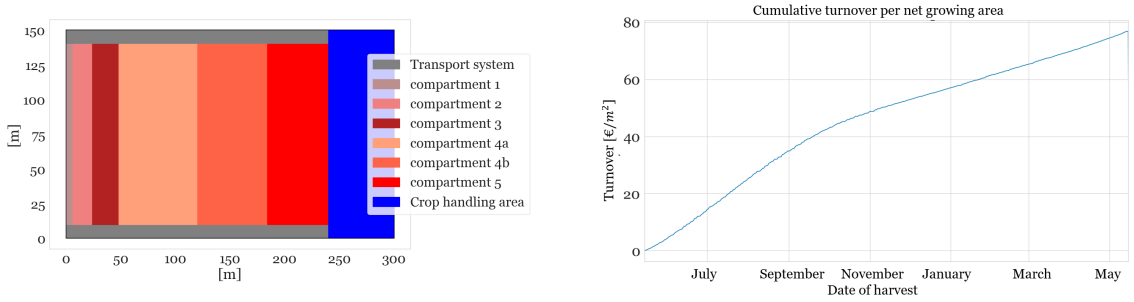


Figure 5.4: Reference conceptual design (left) and corresponding turnover (right) for the sensitivity analysis.

The external variables are varied for the sensitivity analysis. In the first instance, these external variables are the 24-h mean temperature [degrees Celsius] and DLI PAR [$MJ/m^2/day$]. Other varied variables are mainly related to the cultivar, which are for this study the αFDR, LUE and target truss weight. Additionally, there can be differences in the transmission rate and the PAR ratio. The scheme in Figure 2.13 illustrates that a change in transmission ratio or PAR ratio sorts a proportional change to the DLI PAR. Therefore, in this sensitivity analysis, only the DLI PAR is varied, not the underlying variables that cause that change. The number of trusses per m^2 is not measured in the sensitivity analysis as this is a calculated result of other variables. The results of the sensitivity analysis are presented in Table 5.1.

Table 5.1: Sensitivity analysis, in the columns, the relative change in turnover per m^2 growth area for one representative year is given.

Variable	+10%	+20%	-10%	-20%
Temperature ($degreesCelsius/day$)	-0.0338	0.0237	-0.0386	-0.199
DLI PAR ($MJ/m^2/day$)	0.113	0.197	-0.146	-0.215
LUE (kg/MJ)	0.113	0.197	-0.146	-0.215
$\alpha FDR(-)$	-0.0285	-0.539	-0.0149	-0.356
Target truss weight ($gram$)	-0.0169	0.00474	0.0136	-0.00916

The results of the sensitivity analysis (Table 5.1) can be explained as follows. First, a change in temperature does not cause a change in the total growth in a tomato greenhouse (see Section 2.4.3). Therefore, it is not expected that a changing temperature causes a significant variation in the final output. A decrease of 20% in temperature resulted in a proportional decrease in turnover. This reduction is not caused by less growth, but by there being no fruit development beneath a certain temperature. During winter, the temperature falls beneath 15 degrees Celsius, which causes a loss in turnover.

Although the LUE and DLI PAR are a different locations in the scheme in Figure 2.13, they have a similar result. This proportionality is because the LUE and DLI PAR are in the same branch of the scheme. The LUE is a conversion factor from MJ light to kg of tomatoes. Whether the conversion factor changes or the input before that factor, does not matter for the result.

Furthermore, there is the αFDR, which is an empirically found cultivar-dependent parameter by De Koning (1994). The αFDR is not the only cultivar-dependent parameter, and it is not independent. Usually, when the αFDR changes this much, it is out of balance with other cultivar-dependent parameters such as, for example, M', B', or the source/sink ratio. These plant physiological aspects

are beyond the scope of this research project. What can be concluded from the sensitivity analysis is that such a large change in only αFDR is a disturbance of the process.

Finally, the analysis reviewed how a change in target truss weight affects the system as a whole. In principle, there should be no difference in the target truss weight at the end of the cycle. However, there can be small differences, as another truss weight means another optimal truss density. The choices in truss density are discrete steps that deviate from the optimal calculated values. Therefore, there can be an insignificant variation in the final turnover, which was observed in the sensitivity analysis.

5.2. Optimal dimension analysis

The first step in the optimal dimension analysis was to determine varying the dimension of the greenhouse for a fixed length:width ratio. The model calculates an optimal area allocation for a greenhouse, which is fixed from that moment on. The plants should be in a dedicated compartment for specific growing periods. Because the sizing of the compartments is in discrete steps, the optimal continuous sizing of a compartment is not approached. The hypothesis is that increasing the number of rows converges the area allocation towards the optimum. The size of the discrete steps is a multiplication of the row length and width. If the size of a greenhouse changes, for example, from 200 x 200 [m] to 100 x 400 [m], the number of rows is doubled. In addition, the size of the lot is also an important parameter. If, for example, the area expands from 100 x 200 [m] to 200 x 400 [m], this size is multiplied by a factor of four, whereas the number of rows in the greenhouse is multiplied by a factor of two. The difference between the new concept and the conventional concept is that the row length is not limited. Currently, rows are normally at most 100 [m], because otherwise the walking distances for employees that perform the crop-handling actions would be too long. For this new concept, no one enters the greenhouse. Therefore, the length of a row is not an issue.

To test the different greenhouse sizes, a fixed length:width ratio of 1:2 was applied. Compared with a ratio of 2:1, the 1:2 ratio has more rows, which means more possible discrete steps. This ratio was chosen because more discrete steps lead to a better-functioning greenhouse. Very large capacities for the transport system and machinery were chosen so the only influencing factor is the area allocation. Figure 5.5 proves the hypothesis that larger greenhouses of the same length:width ratio have a larger final turnover per m^2. The data points indicate an increasing trend for increasing size. This trend is strongest for smaller greenhouses.

For three sizes, the effect of the length:width ratio on turnover was simulated. First, a small greenhouse of only 1-ha (Figure 5.6) was simulated. Then, simulations with 3-ha (Figure 5.7) and 6-ha (Figure 5.8) greenhouses were conducted. The increasing trend in turnover for lower length:width ratios (i.e. more rows) is stronger for smaller lot sizes. This trend can be explained because for larger lots there are already enough rows to make a viable division in compartment sizes due to the more discrete options. For the 6-ha size, hardly any trend was observed, which suggest that from that size on, the length:width ratio is of less importance. Evident from the figures for length:width ratio variations, is that for the simulations with larger greenhouses, the turnover is higher, which is in line with Figure 5.5. The conclusion is there is an increasing trend in turnover for larger greenhouses and smaller length:width ratios. This trend is strongest for the smallest greenhouses. However, there are always specific areas in which a continuous calculated value fits the possible discrete values better or worse than the trend.

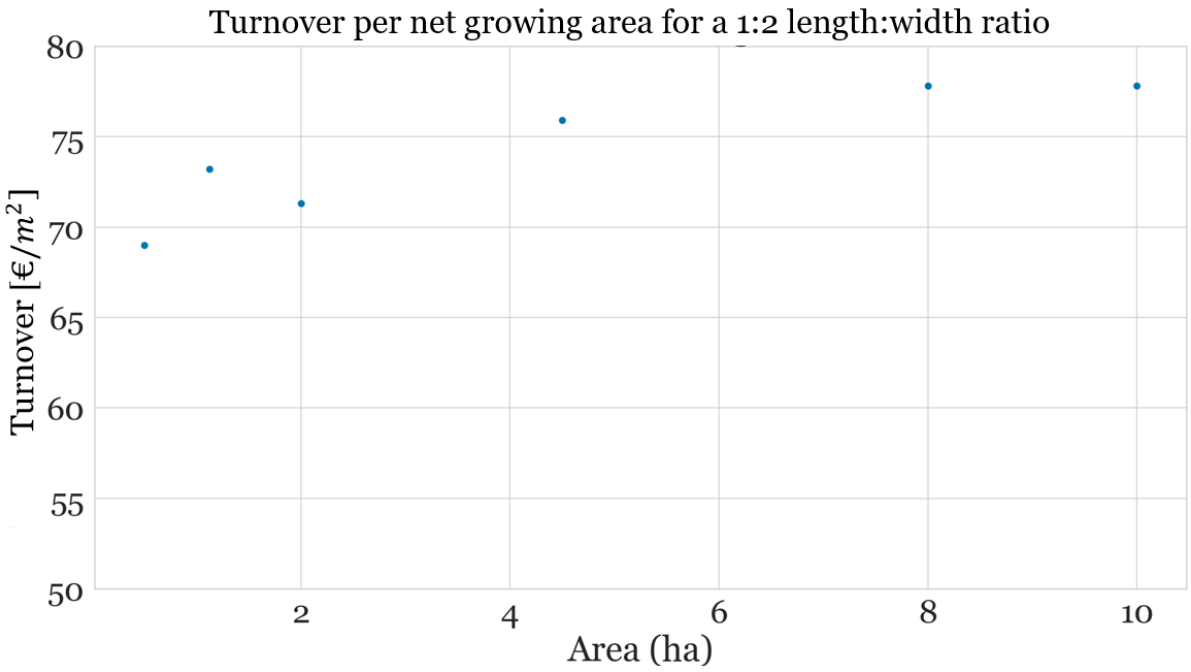


Figure 5.5: Turnover per net growing area for multiple lot sizes

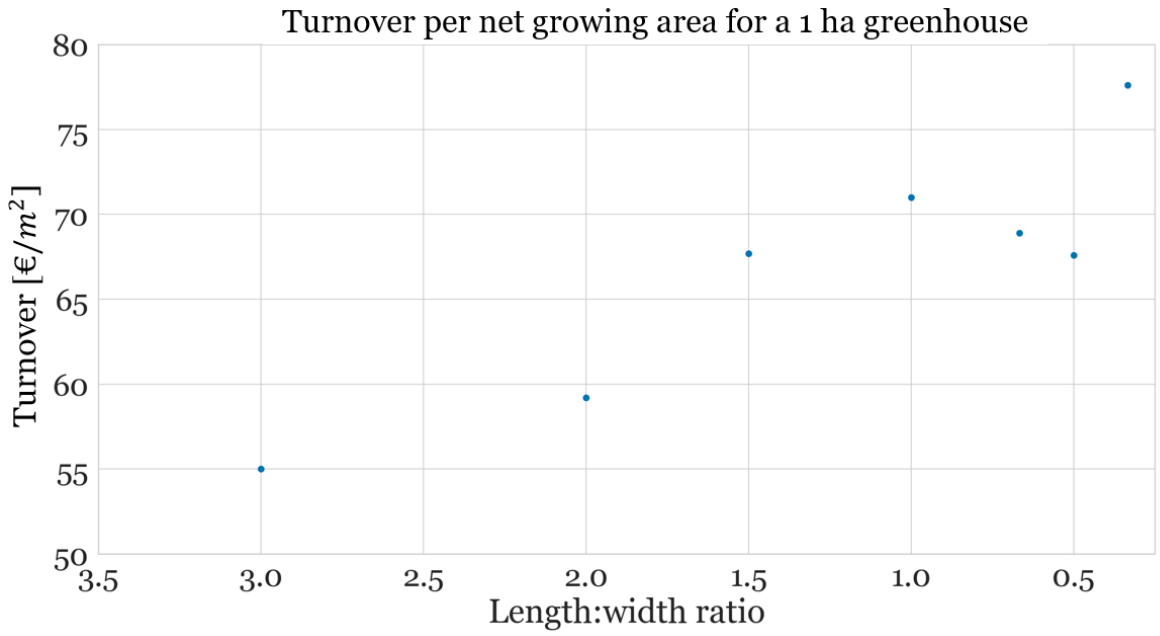


Figure 5.6: Turnover per net growing area for multiple length:width ratios for a 1-ha lot

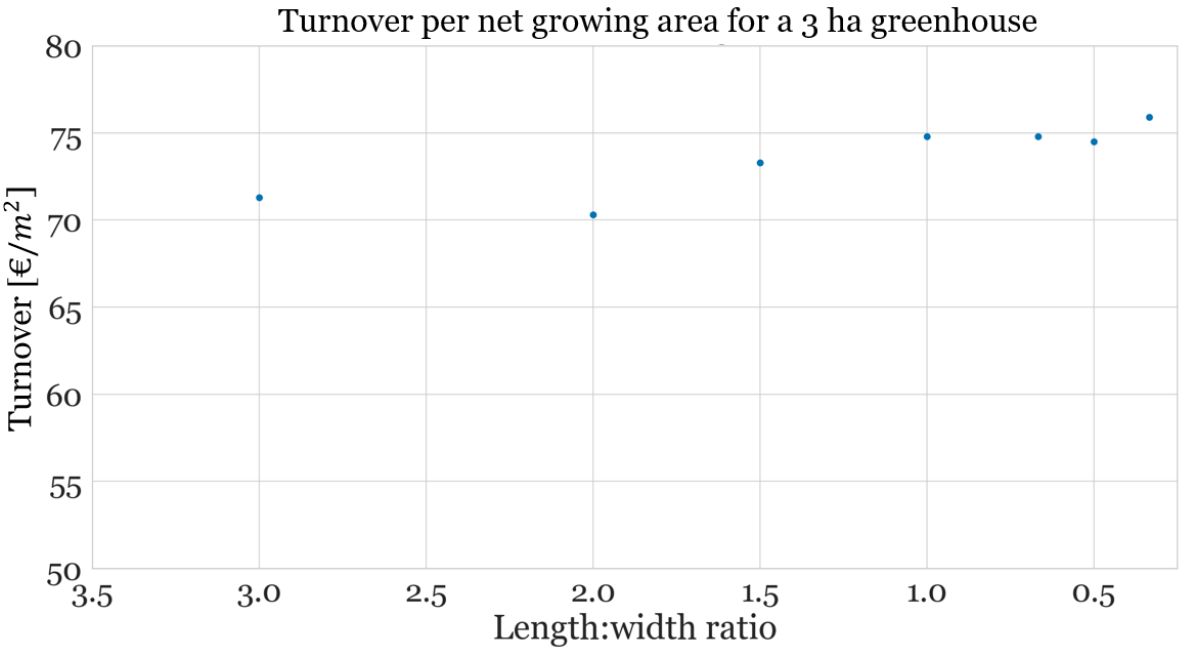


Figure 5.7: Turnover per net growing area for multiple length:width ratios for a 3-ha lot

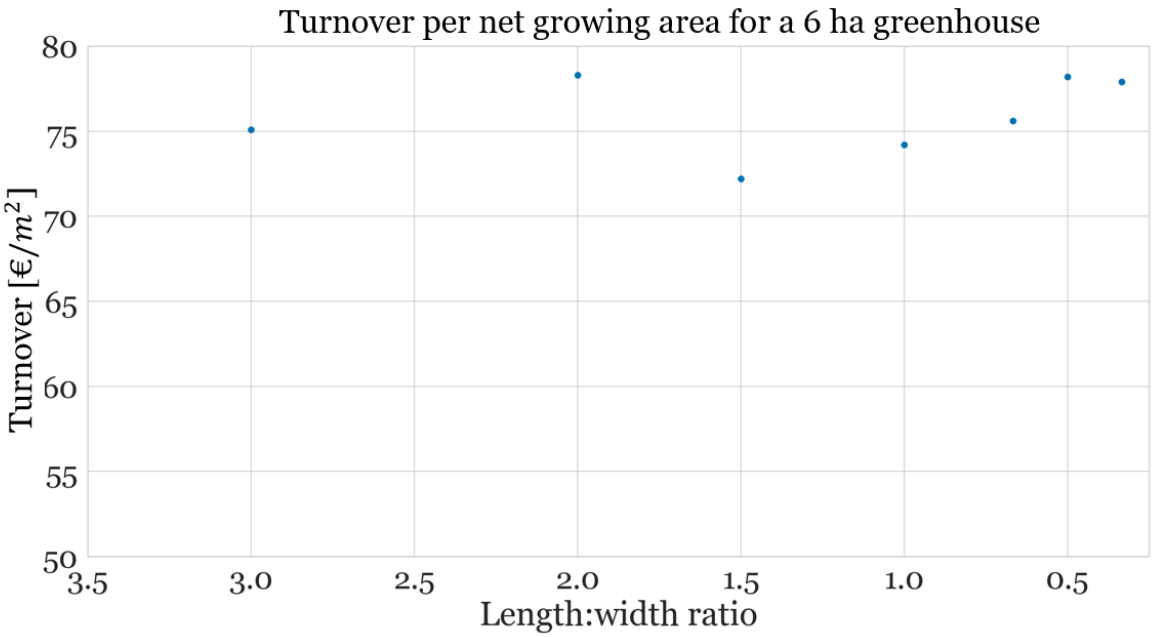


Figure 5.8: Turnover per net growing area for multiple length:width ratios for a 6-ha lot

5.3. Case studies

In the previous two sections, general trends were identified. From this section on, the steps in the evaluation protocol are performed for multiple case studies. The purpose of using case studies is to prove that the design framework is generic. In addition to that, these case studies show how the simulated performance is for different geographical locations on earth. For all case studies, a 4.5-ha lot with a 1:2 length:width ratio (see Figure 5.3) is chosen as this seems to be a well-performing configuration following the figures in Section 5.2. Moreover, this size is about average for nurseries in the Netherlands (CBS, 2018). In the upcoming subsections, a case is defined as a geographical location. This implies that each case has another input climate dataset. There is one reference case based on typical values, which is discussed more elaborately as example. To prevent redundancy, the next three case studies are discussed more concise.

The definition of the case studies in the next subsections is that there is one reference case study plus three case studies. These three case studies are based on weather data retrieved from governmental weather institutes. Global radiation and 24-h mean temperatures are retrieved. These values are converted into inside greenhouse values. For the radiation, still the scheme in Figure 2.13 is applied. This means that the inside DLI PAR is obtained by multiplying outside data with the overall transmission rate of the greenhouse and the PAR ratio. For the outside-inside temperature conversion, there is no straightforward equation. This outside-inside temperature difference depends highly on the temperature-regulating measures, wind speed, and the outside radiation (Mesoudi et al., 2010). For this simulation test, it was feasible to work with the temperature with an added term, which was set equal to 3 degrees degrees Celsius.

The location for the reference case is Rotterdam, the Netherlands. This location is chosen as this is in one of the regions that produce the most greenhouse tomatoes in the world. In addition to that, for Rotterdam, validated and widely used data is structured and freely available. That location is also used for case study 1; however, for case study 1 another dataset is used. For the reference case typical values are used, while real weather data is used for case 1. This enables a comparison between the simulated performance for typical numbers and weather data. Then, there was searched for other locations with different conditions. A country that has data structured and freely available in Australia. In Australia, the temperature is generally higher and the radiation is more intense. Two locations in Australia were picked, with different characteristics. The locations are Darwin and Melbourne. Darwin has a relatively constant climate, whereas in Melbourne there are more variations between summer and winter. So the case studies are case 1: Rotterdam, case 2: Darwin, and case 3: Melbourne.

The values for the case studies are from governmental institutes, the data for the reference case is found in the literature (Vermeulen, 2016). Numbers for typical values and weather data are different, but there is not necessarily one better dataset. It is valuable that both is simulated such that excessive differences could have been identified if that had been the case. Because the nature of the typical value data is different than weather data, there is a deviation in the approach in robustness assessment step for the reference case and the case studies. The robustness assessment for cases with weather data, is based upon simulating for different years, whereas for the typical values, the price curve is changed, lost crops due to illness are included, and noise is added to the input data. Furthermore, the calculation of capacities for the transport system and crop-handling tasks is done more elaborately for the typical numbers than for the weather data. For weather data this is done more concise to prevent redundancy. One can perform its own evalation by employing the steps in the example of the reference case.

The first step in the evaluation protocol for the case studies, is evaluating the simulated performance. For the case studies, this step is done by using typical or 10-year mean values. This data is preferred over data sets for a specific year, as in practice, there is no knowledge about upcoming weather conditions in the next years. In a next step, the robustness assessment, there is reviewed whether the performance is affected by weather data for specific years, which varies from mean values. The procedure for evaluating and improving the simulated performance is in Figure 5.9. This part of the evaluation uses the KPIs occupancy rate of the greenhouse and the share of lost plants of the total due to over-occupation.

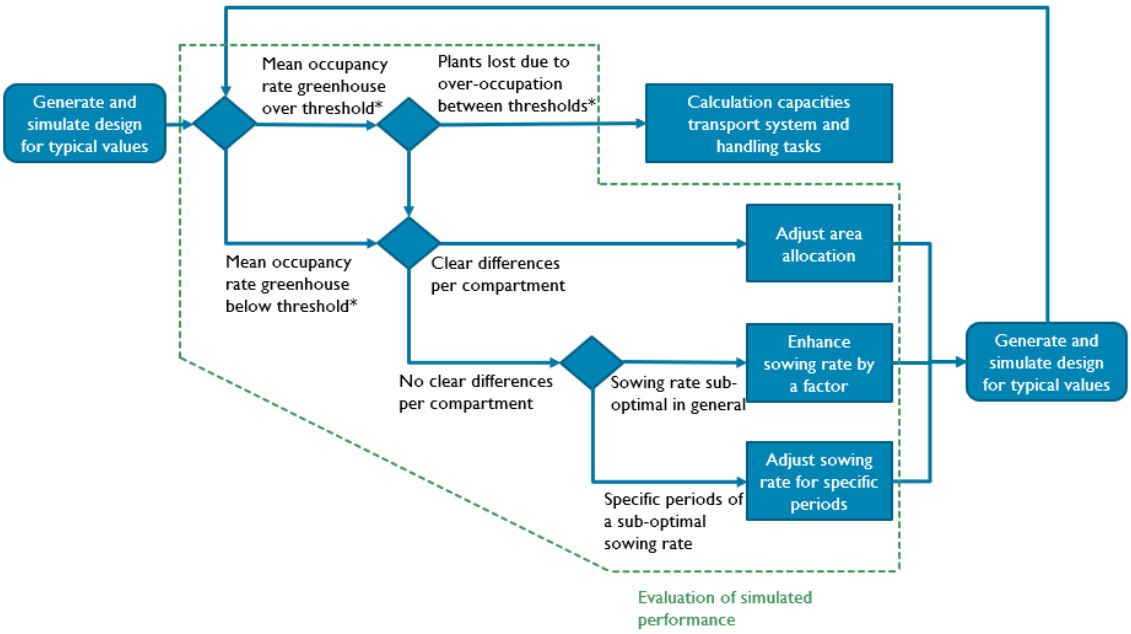


Figure 5.9: Protocol for the evaluation of the simulated performance. *Thresholds are discussed in this section.

In Figure 5.9 the terms 'thresholds' and 'sub-optimal' are used. The definition of these terms can differ per application. In this project there is aimed for an average occupancy rate greenhouse over 90%. This occupancy rate greenhouse is used as starting point and can be adjusted if that appears to be better in a later phase. An occupancy rate greenhouse of 100% is not reachable, because the climate conditions are too volatile. In conventional greenhouses the average occupancy rate is lower. When a new growing phase is started, the greenhouse is empty for a period. Moreover, in the first period the plant density is lower than it potentially can be, which is marked as a low occupancy. The threshold for plants lost due to over-occupation is set at 2% to 7% of the total number of plants sowed, at this stage. In an operational phase, there will be unexpected events (i.e. differing climate conditions), which increase the share of plants lost. It is considered that when the share of plants lost is lower than 2% of the total plants sowed, there is room to fit more plants into the greenhouse. If the share of plants lost is higher than 7% there are too many plants lost for viable operations. These thresholds on lost plants are dependent on the costs of seeds, which are beyond the scope of this research project. After quantitatively examining whether the thresholds are met, the evaluation is qualitatively. The evaluation is based on a visual inspection of the occupancy rate greenhouse and lost plants curves for a representative year. The first step is to inspect the occupancy rate greenhouse per compartment. If there are compartments with a low overall occupancy, a row of this compartment can be added to that of another with a high occupancy.

The word 'sub-optimal' (Figure 5.9) is defined as that there is a clear optimisation possibility observed by a visual inspection of how the occupancy rate greenhouse evolves. The first option is that the sowing rate is sub-optimal in general. This means that over the entire representative year the occupancy rate greenhouse is either too high or too low. It is too high if the share of lost plants is too high and the occupancy rate greenhouse is touching 100% over the entire period. It is too low if the occupancy of 100% is not reached at all. The second option is that, in volatile climate conditions, the sowing rate is sub-optimal. This could lead to periods of falling occupancy or periods with an overcompensation that leads to more plants lost. If this fall or overcompensation is too big, the sowing rate is adjusted for these periods.

The next evaluation step is calculating the capacities of the transport system and of crop-handling

tasks. In the simulation model there is implemented that the transportation system has a maximum acceleration of 0.50 m/s^2 and a maximum speed of 3 m/s, and the load and unload time equals 1 second. One can argue about these numbers. They are in correspondence with the standard NEN 13002-2 (2014) for the movement of trolleys in cranes but could have been based on another norm as well. The number of transport lanes is minimised. The reason for minimisation is that a transport lane is expected to be expensive, and therefore should be used efficiently. The trade-off is that for a more dense use of the transport system, it will happen more often that a plant have to wait too long before pickup and is therefore considered to be lost. Waited too long before pickup is defined as more than 24 hours. A plant is lost if it is collected from the greenhouse for treatment more than 24 hours later than desired. For the other way around, from greenhouse to crop-handling area, this time was set to six hours, as the plant is not connected to the watering system and will therefore degrade quickly in the crop-handling area. The boundary value for the number of transport lanes was set at 1% lost plants. One can argue about this cut-off, but it depends mainly on cost. If the costs for seeds and the first growing stages are very high, it is desirable no plants are lost, and then the cut-off should be lower, or vice versa. For this study, 1% was considered low. If a cost analysis is conducted in a follow-up study, it will also be important to quantify the cost of a lost plant.

The last step is a robustness assessment. In the robustness assessment, there is simulated for five individual years. For the reference case, there is simulated for another price curve, lost plants due to illness are included, and noise is added to the signal. One can argue about when the system is considered robust. In this study, the rule is implemented that the KPI turnover per m^2 net growth area may not be affected more than 5% for a test. This rule is used under the constraint that there may not be a significant systematic offset in the dataset for the robustness assessment compared to the reference dataset.

5.3.1. Case study 0: Typical values for Rotterdam, the Netherlands

First, a preliminary conceptual design was generated for the typical climate conditions for Rotterdam, the Netherlands (Figure 5.2). This generated conceptual design in Figure 5.10 is the starting point of evaluating the simulated performance. In the sections before, the KPI turnover per m^2 was used. This section uses the occupancy rate of the greenhouse and the number of plants lost due to over-occupation. These KPIs are interfering with each other. The sowing rate is an important factor for these KPIs. Generally, a higher sowing rate leads to a higher occupancy rate greenhouse and a higher number of plants lost and vice versa. As illustration how the occupancy rate greenhouse evolve over a year, for this case study Figure 5.11 is constructed.

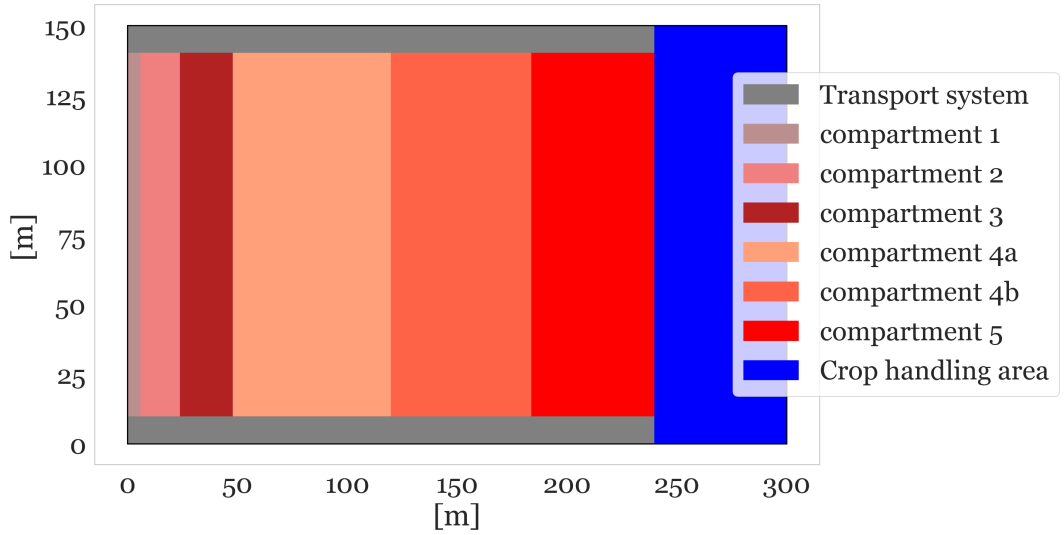


Figure 5.10: Preliminary conceptual design for a 150x300 [m] lot.

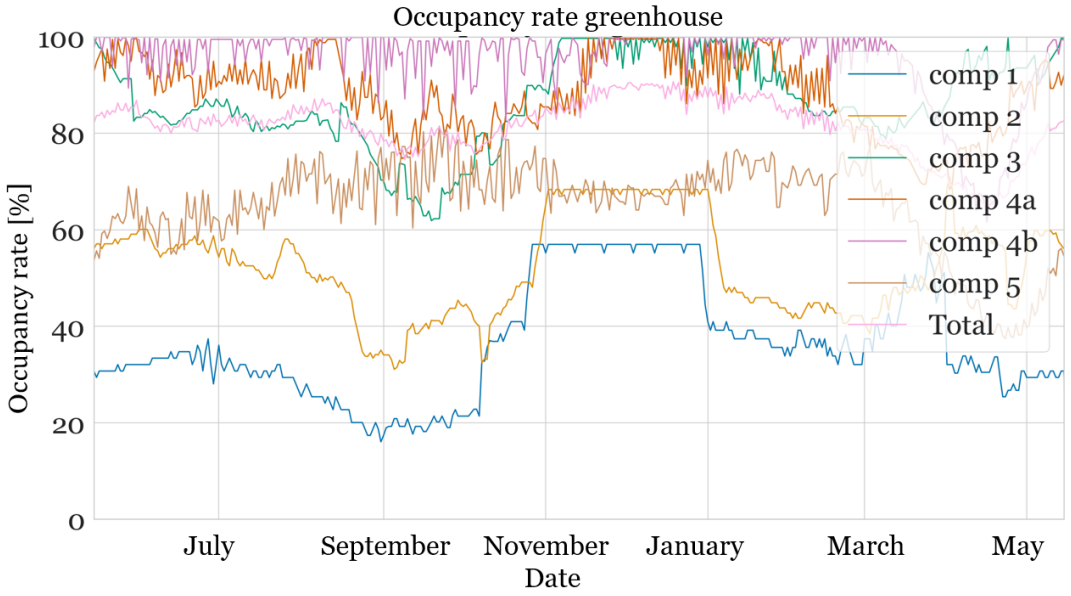


Figure 5.11: Corresponding occupancy rate greenhouse of the generated greenhouse in Figure 5.10 for a simulation with typical values for the Netherlands (Figure 5.2). 'Comp' means Compartment.

In the case study, compartments 1 and 5 are too large. This issue mainly concerns how the optimisation is implemented. There is run for two years, in which the greenhouse begins and ends empty. One representative year was used to retrieve results from the simulation. The model tends to begin and end the run year with as many viable plants as possible. This approach could be optimal for an empty start and finish but is not the optimum for a continuously filled greenhouse. Something the model does not account for is there is a buffering capacity in neighbouring compartments. However, the number of plants in the wrong compartments for a day should be minimised; it could be beneficial that Compartment 4b is used as an overflow compartment for Compartment 5. Therefore, it is proposed to dedicate one row of Compartment 5 to Compartment 4b. In addition to that, there is enough space left to interchange one row of Compartment 1 and one of Compartment 2, with one row of Compartment 3. There is also room to enhance the sowing rate; this is done by +12%.

Simulating for the proposed improvements on area allocation and sowing strategy led to the outputs in Figures 5.12 and 5.13. The results of these improvements seem good. The overall occupancy rises from 82.4% to 90.7%. The share of lost plants is 5.0% of total sowed. These values are viable to go to the next step in the evaluation process. Before going on to the next stage, a closer look on the lost plants is taken. A plant can stay in a neighbouring compartment. The limitation is that a plant in a gutter may only stay between other gutters and trays in a tray compartment. This situation leads to a lower number of plants lost. Figure 5.14 illustrates when crops are lost and how many crops stay in another compartment. Initially, this number seems substantial; however, this is the number of days a plant is in a wrong compartment. Therefore, on average, a plant stays for less than one day in the wrong compartment, which is viable.

The next step in the evaluation protocol is to calculate the capacities of the transport system and the crop-handling tasks. First, the capacities for the transport system were reviewed. Second, the capacity for crop-handling tasks was treated. These calculations were based on the improved conceptual design (Figure 5.12). For the transport system, the transport distance, velocity and acceleration are important. The element that can be changed by varying the floorplan is the transport distance. In this research project, the distance from a compartment to the crop handling area is constant per compartment. However, there are two design options at this stage: 1) using one central crop-handling area (Figure 5.12), or 2) using two central crop-handling areas (Figure 5.15). The only difference between these two is either one or two crop-handling areas. The advantage of two crop-handling areas, is that the total transport distance is about half of the distance for one crop-handling area. Therefore, for the efficiency of the transportation of crops it is beneficial to have two crop-handling areas over one. Whether this

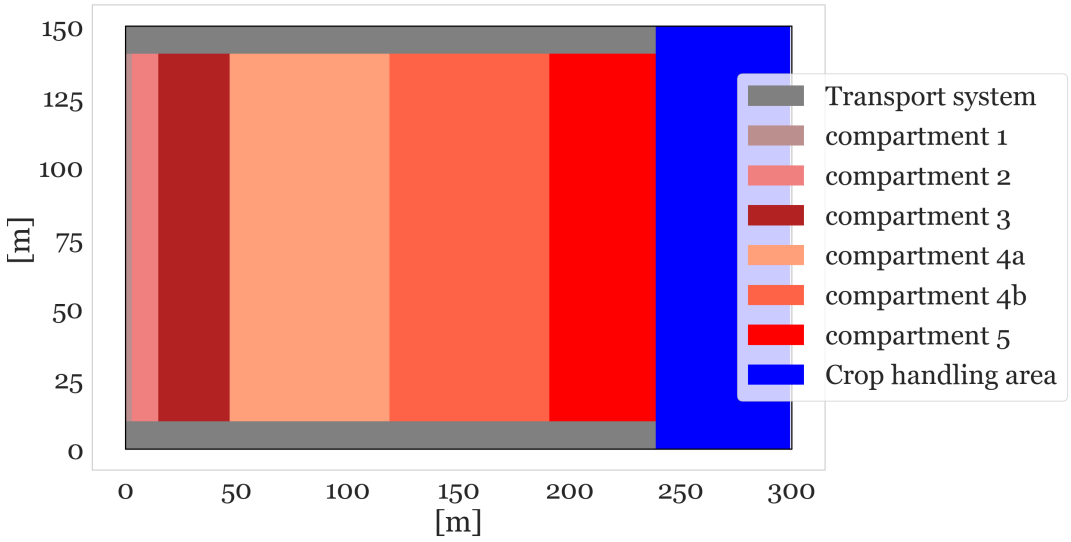


Figure 5.12: The improved conceptual design. Improvements are made in respect to the preliminary conceptual design (Figure 5.10)

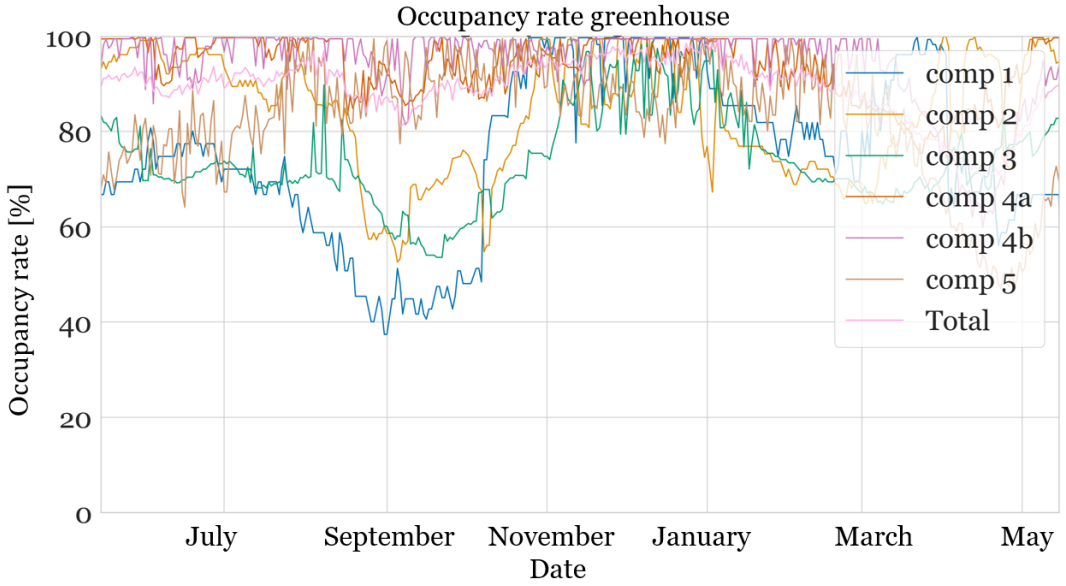


Figure 5.13: Occupancy rate for the greenhouse of the improved conceptual design (Figure 5.12). 'Comp' means Compartment.

is possible depends per lot, as the lot must be reachable from two sides. For this case study it is assumed that the lot is reachable from two sides. Therefore, from now on, the configuration with two crop-handling areas was used.

First, there is simulated with 10 transport lanes. Because 10 lanes is many for this configuration, all the required transport movements were executed. Next, the number of transport lanes is minimised. The results of simulating for multiple configurations in number of transport lanes are presented in Table 5.2. It is important to emphasise that one transport lane is defined in this context as one inbound / one outbound lane for trays, and one inbound / one outbound lane for gutters. A design choice could be to use the gutter transport lane either for transport to both crop-handling areas or to have its own shuttle per crop-handling area. The conclusion drawn from 5.2 is that one transport lane is the best choice based on the criteria that the number of transport lanes should be minimised and the plants lost due to an over-occupied transport system should be lower than 1%. A more detailed design of the transport system should be implemented to make a final decision regarding capacity; however

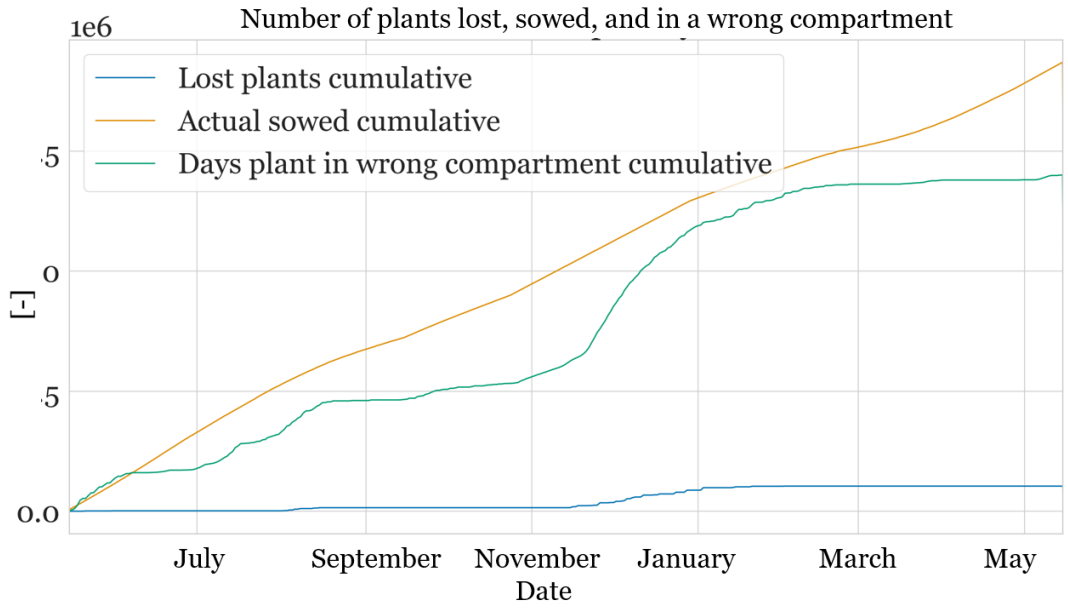


Figure 5.14: Number of plants lost due to over-occupation and total number of days one plant is in a wrong compartment

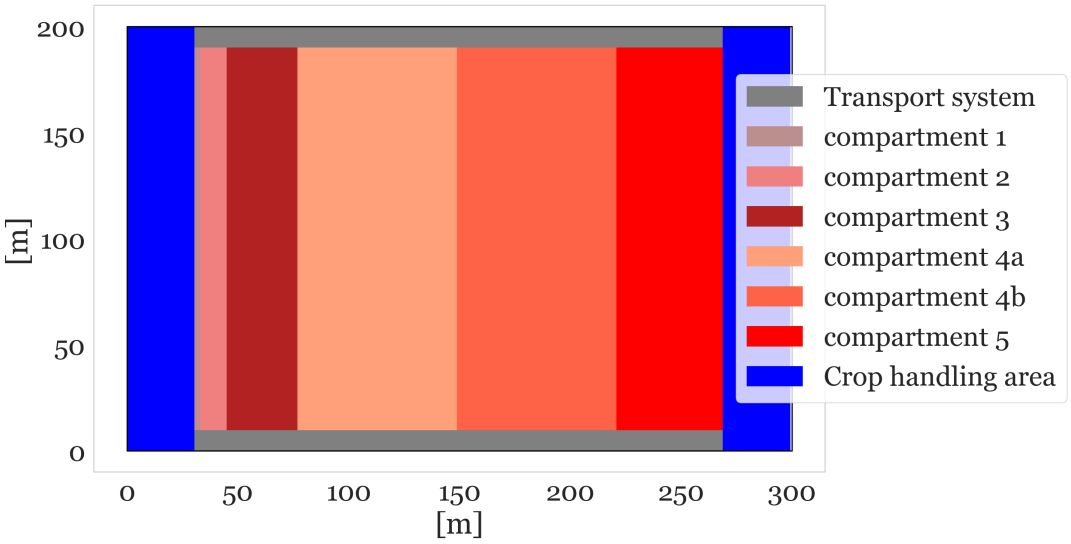


Figure 5.15: The improved design (Figure 5.12), with the variation of having two central-crop handling areas instead of one.

the capacity corresponding to the case with one transport lane is considered realistic. The required width of the transport system should also be reviewed, because it will probably be smaller than the currently implemented 10 [m]. The width should be minimised to make the most efficient use of the area available in the greenhouse.

To visualise the results in Table 5.2, for the case of one and two lanes, the graphs illustrate the maximum waiting time before transporting on a specific day (Figure 5.16). If this time is more than 24 hours, the crop is lost. An order of actions is implemented, which causes the waiting time for some pickups to be systematically higher than for others. This issue need not cause any problems, but the strategy will be different during real operations. This implemented strategy is no problem for the capacity of the system.

Table 5.2: Output of simulations for the transport system.

Number of transport lanes	Lost crops percentage due to too long pick up times
10	0
5	0
3	0
2	0
1	0.89

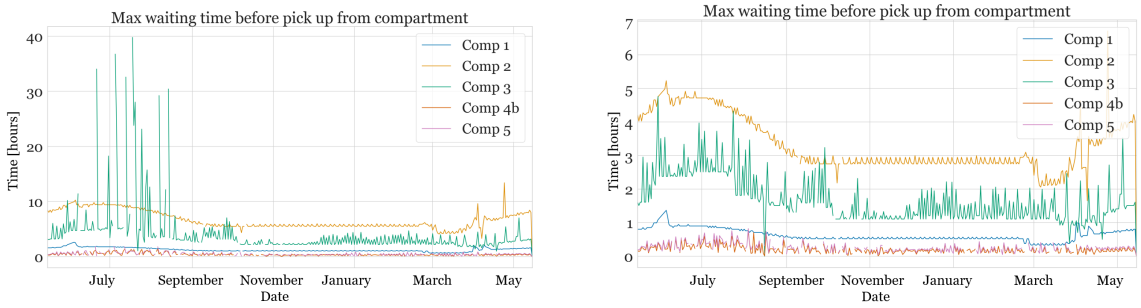


Figure 5.16: The maximum waiting time for pickup per day from a greenhouse compartment for one transport lane (left) and two transport lanes (right).

The final step in the conceptual design is implementing the capacities of crop-handling activities. The case-based design generator was also used to calculate the minimum and maximum number of handling activities. As a starting point, the maximum required capacity was taken and matched with the possible equipment in Table 2.2. This capacity led to an implementation of the list provided in Table 5.3. By simulating this equipment configuration, the hours a machine should be operational on a specific day are listed in Figure 5.17. This figure equals the number of hours a machine must work on the given capacity to meet the requirements. The cut-off point at which a plant is lost was set again to six hours. Possibly, these waiting times can be reduced by employing a smarter pickup strategy. When a machine is occupied, waiting can occur before transporting the plant from the greenhouse to the crop-handling area. These implemented waiting times lead to a slightly lower turnover. In Figure 5.15, the first three machines follow the trend of the sowing curve, whereas the last two (monitoring and harvest) have variable lead times that cause a spiky output.

Table 5.3: Specifications of robotic solutions based on the required capacities and the available equipment of Table 2.2

Component	Specification
Sewing Machine	Visser Horti Systems Auto Seed Granette, 176 trays/hour, up to 48 plants/tray of 600x400 [mm]
Transplant Step 1	ISO Group Grade 7000, 350 trays/hours, up to 48 plants/tray of 600x400 [mm]
Transplant Step 2	ISO Group Grade 7000, 175 gutters/hours, up to 32 plants/gutter
Harvesting	30 manual stations, 7.5 gutters/hour/station
Monitoring	1 automatic station of 360 gutters/hour/station
Transport System	4 shuttles, max velocity equals 3 m/s, max acceleration equals 0.5 m/s^2, 1 gutter per time transported, loading & unloading time both 1 second

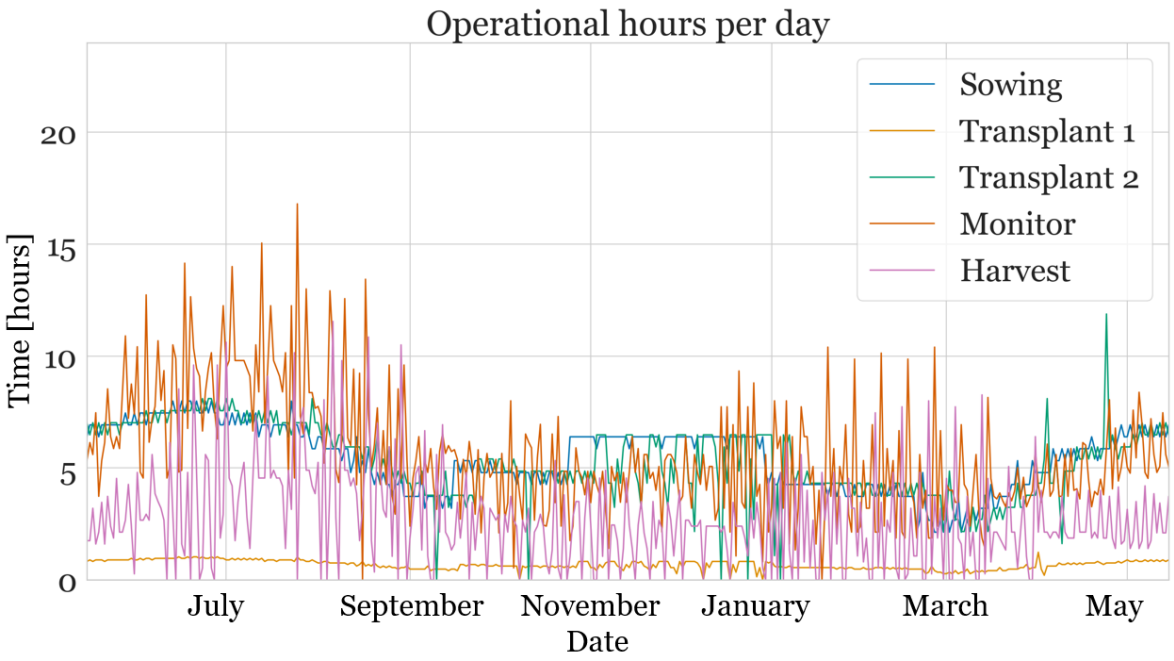


Figure 5.17: Simulated hours that an implemented machine of Table 5.3 should be operational

The robustness assessment is done in two steps. The first step was implementing another price curve and dedicating a number of plants lost due to maltreatment or illness. The second step is adding noise to typical values input signal. The first robustness test involves providing a percentage of lost plants due to illness and providing a tomato-price curve as input. A block price curve is implemented based on the EU curve (European Commission, 2022). Although the price curve is volatile and sensitive to what happens in the market, for a user of the framework, it is interesting how an artificial turnover changes in responses to a changing price curve. The reason a block curve is implemented is primarily because the EU curve is highly volatile. In the block curve, general trends are implemented as a lower price during summer and a higher price during winter. Additionally, in the new concept price agreements can be made before the onset of a batch, making the curve less volatile for a user. The implemented block curve and EU curve are presented in Figure 5.18. This implementation leads to a change in overall turnover from 84.0 to 83.4 $€/m^2$. This is a drop of 0.5% for the other price curve, which is a robust behaviour. In Figure 5.19 the difference in how that turnover develops can be observed.

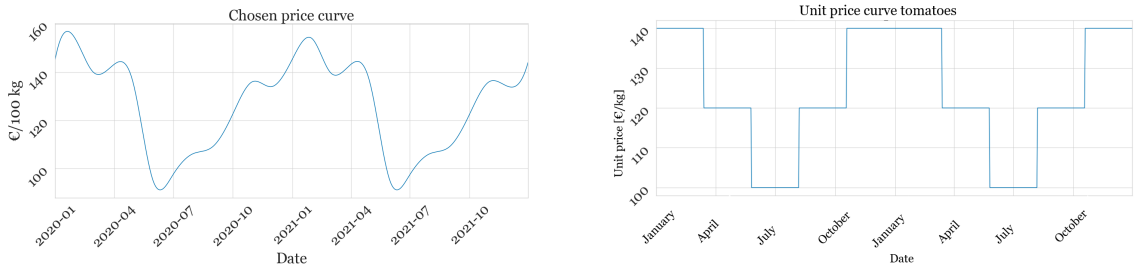


Figure 5.18: EU 5-year average price curve for tomatoes (left) (European Commission, 2022) and the implemented block curve (right)

The next step in the robustness assessment was adding a share of crops lost due to illness. Two percent crop lost due to illness was added. This results in an increase in lost plants from 5.9% to 7.9%. This leads to a fall in turnover of 2% as well, which is proportional to the share of plants lost due to illnesses. The system is therefore still considered robust.

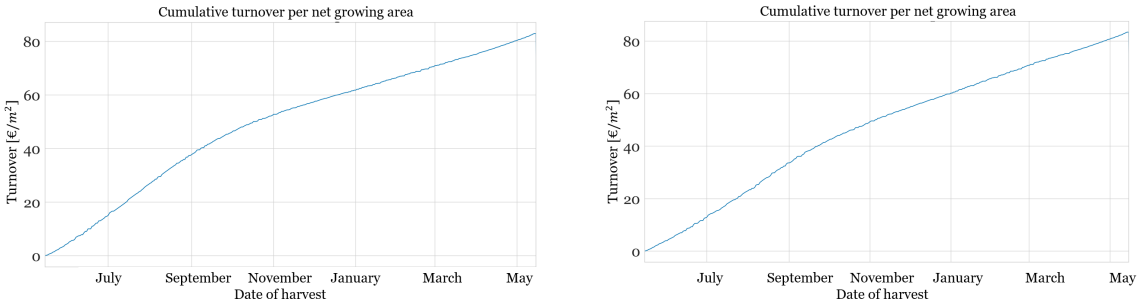


Figure 5.19: Turnover per m^2, in which a m^2 is defined as net growing area, for the constant price (left) and block curve of Figure 5.18 (right)

The last step in the robustness assessment was performed by adding Gaussian noise to the temperature and light intensity curve in Figure 5.2. A noise is added with a maximum of ±1 $MJ/m^2/day$ and of ±2 $MJ/m^2/day$. The new inputs are in Figure 5.20 and 5.21. Three random seeds of this added Gaussian noise were simulated. The results are in Table 5.4. The conclusion drawn from the table, is that the concept is robust. If there is noise added of ±1 $MJ/m^2/day$, which is at least 12.5%, and up to 33.3% of the initial light intensity, there is only a fall in turnover of just below 2%. The case in which noise is added that is at least 25% and up to 66% of the initial light intensity, there is a decrease in turnover of just below 3%. This figure means the decrease in turnover is smaller than one-tenth of the added noise. As this result small, the conclusion from this test is that the system is robust.

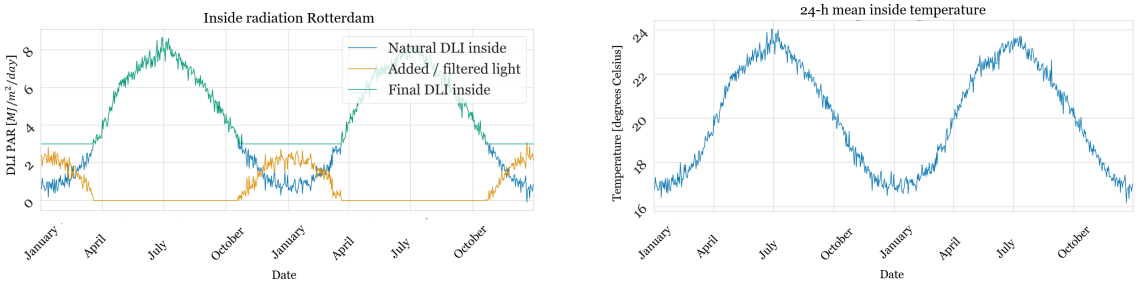


Figure 5.20: Input curve with ±1 $MJ/m^2/day$ noise added to the DLI curve of Figure 5.2

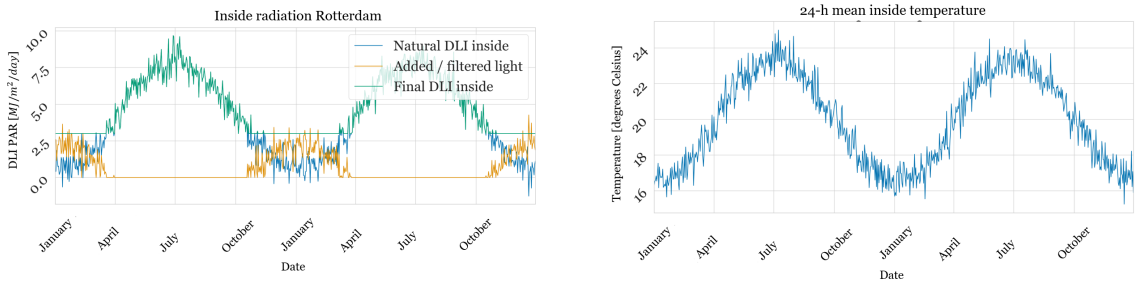


Figure 5.21: Input curve with ±2 $MJ/m^2/day$ noise added to the DLI curve of Figure 5.2

A boundary check assessed whether the truss weight at harvest is constant. This assessment was conducted by creating boxplots of the truss weight for every run in Table 5.4. Figure 5.22 illustrates how the boxplot is related to the confidence interval of a normal distribution. If the distribution is skewed, the location of the median will not coincide with the arithmetic mean. The acceptable range varies per tomato species. In principle, on the market, there is higher demand for uniformity when tomatoes are smaller. For the mid-size truss tomatoes implemented in this study, the acceptable range is set at 10%, which means ±50 grams. Figures 5.23, and 5.24, present the results of the assessment. This spread can be even smaller if the temperature in Compartment 5 is slightly varied from time to time.

Table 5.4: KPIs for a simulation with typical values and added noise.

Run	Input	occupancy rate greenhouse	Total lost plants	Production kg/m^2	Turnover €/m^2
0	Typical values (Figure 5.2)	90.7	5.9	70.0	84.0
1	Typical values plus block price curve (Figure 5.18)	90.6	5.9	70.0	83.4
2	Typical values plus 2% plants lost due to illness	90.6	7.5	68.6	81.8
3	Typical values ±1 $MJ/m^2/day$ noise (Figure 5.20)	89.4	7.1	68.5	81.7
4	Typical values ±1 $MJ/m^2/day$ noise (Figure 5.20)	89.7	7.0	68.6	81.8
5	Typical values ±1 $MJ/m^2/day$ noise (Figure 5.20)	89.4	7.0	68.6	81.8
Mean		89.5 ±0.1	7.0 ±0.0	68.5 ±0.0	81.8 ±0.0
6	Typical values ±2 $MJ/m^2/day$ noise (Figure 5.21)	88.6	8.2	67.3	80.5
7	Typical values ±2 $MJ/m^2/day$ noise (Figure 5.21)	88.2	8.1	67.3	80.6
8	Typical values ±2 $MJ/m^2/day$ noise (Figure 5.21)	88.7	8.8	67.3	80.5
Mean		88.5 ±0.2	8.3 ±0.3	67.3 ±0.0	80.5 ±0.0

This variation is preferably in Compartment 5, as at the the end of the process, a good forecast on the final truss weight can be made. By increasing or decreasing the temperature, the truss weight at harvest can be steered in the desired direction.

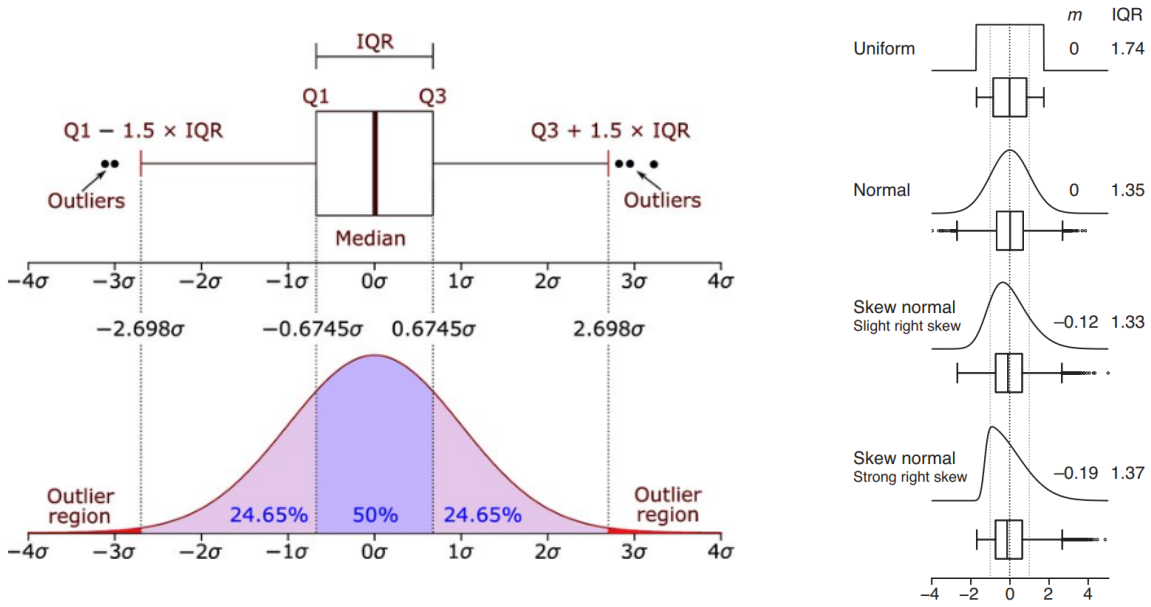


Figure 5.22: The definition of the boxplot regarding the confidence interval. The left figure is reprinted from Verma and Range (2020), the right figure is reprinted from Streit and Gehlenborg (2014)

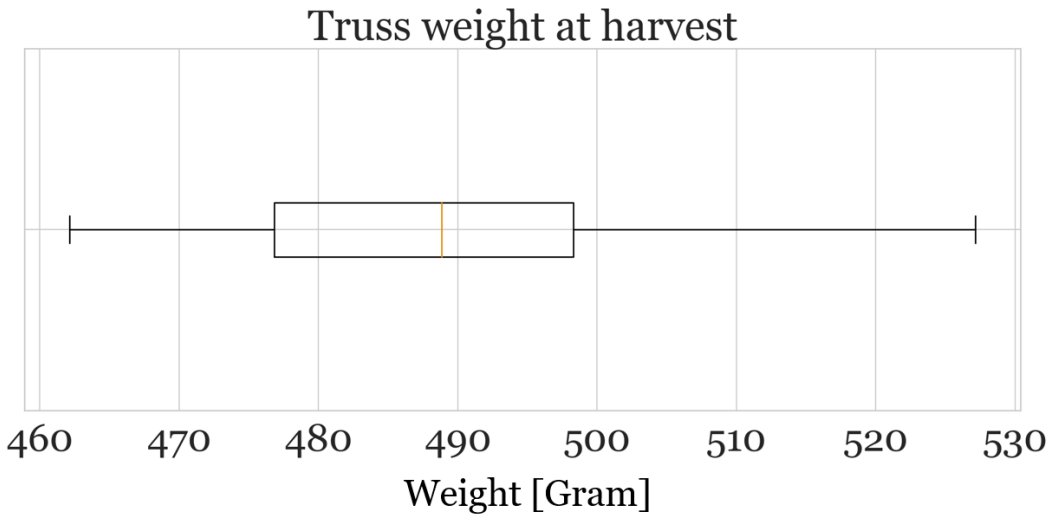


Figure 5.23: Boxplot truss weight for Run 0 of Table 5.4

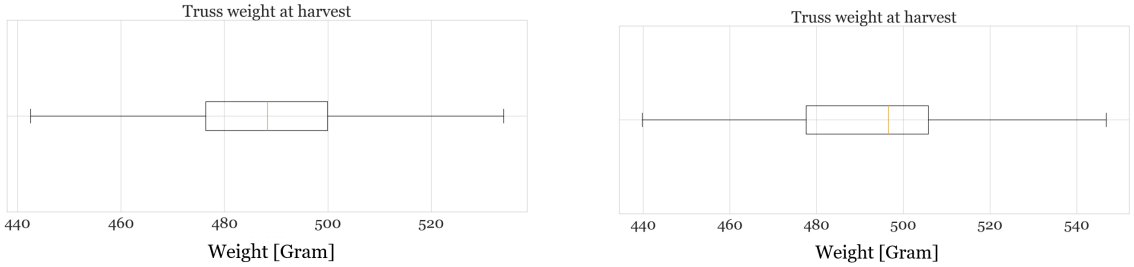


Figure 5.24: Boxplots for truss weight at harvest for Runs 3 (left) and 6 (right) of Table 5.4

The robustness assessment is passed, which means that the improved design. with the two-crop handling areas (Figure 5.12) and calculated capacity of the transport system and handling-tasks is the design, which is presented. This design and strategy passed the thresholds. All the four KPIs are reviewed in the evaluation protocol. In addition, the boundary condition of the maximum variation in truss weight is respected. Furthermore, the performance indicators of the operational time of machinery and waiting time before pick up show viable numbers. Using these results, there next design phase can begin. In this design phase the preliminary or even detailed design of sub-systems as the transport system, and components as the gutter itself, should be done.

In that next design phase, the developed framework is valuable as well. The framework is able to produce other design requirements as loads. The loads in the different compartments can be calculated from the growth curves in combination with the plant densities. For Compartments 1 and 2, the loads are mainly determined by the weight of the cultivation medium; there is hardly any growth of the plants. The growth for Stages 1 and 2 is the first and least steep part of the curve in 5.25. The growth in Compartments 3, 4a, 4b, and 5 is more significant. The weight at the end of the compartment equals approximately 90, 250, 410, and 510 grams, respectively, for the aforementioned compartments. The ratio between truss weight and fruit weight varies for the different compartments. It is assumed the truss weight will be 30%, 60%, 70%, and 80% of the total plant weight, at the end of Compartments 3, 4a, 4b, and 5, respectively. Therefore, per truss, there is a load of 300, 417, 586, and 638 grams at the end of the respective compartments. Note that this weight will vary proportionally to the different truss weights, as the presented numbers are only the target. The truss density per compartment is at largest 64, 30, 30, and 48 $trusses/m^2$ for compartment 3, 4a, 4b, and 5, respectively. Multiplying the truss density with the corresponding plant weight leads to a maximum load of 19.2, 12.5, 17.6 and 28.1 kg/m^2 of plants. This should be incorporated in the gutter design, the suspension system and the transportation system of the gutters. Multiplying the plant weight by the maximum number of plants

per gutter provides the maximum gutter load. The maximum number of plants per gutter is considered 16. In combination with a maximum of three trusses per plant, this figure leads to a maximum gutter load of 14.4, 20.0, 28.1, and 30.6 kg/gutter, for Compartments 3, 4a, 4b, and 5, respectively.

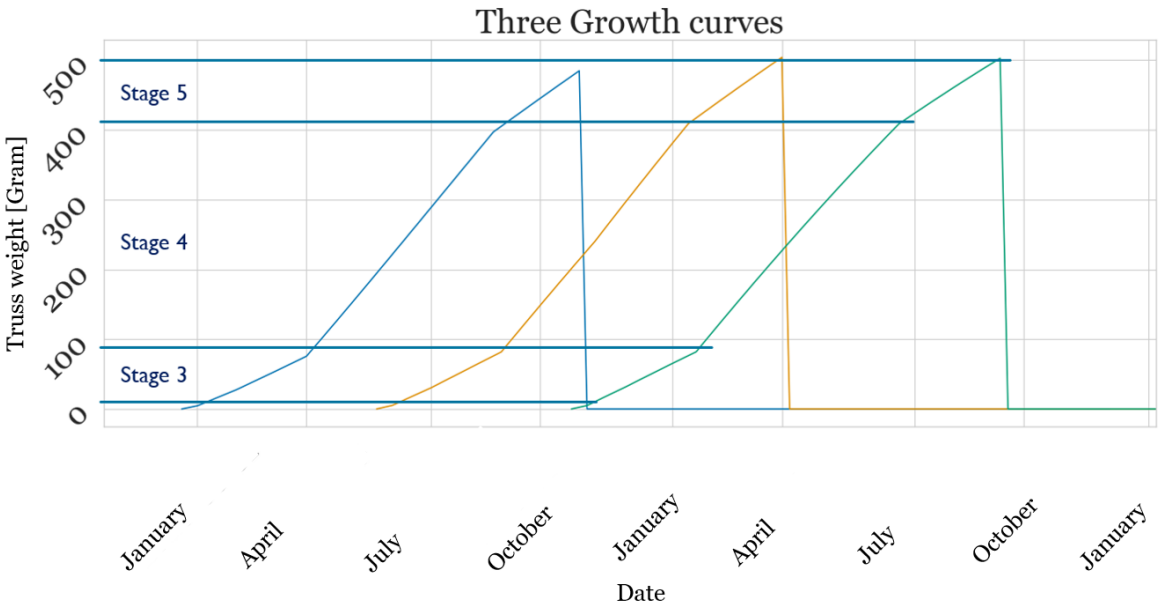


Figure 5.25: Three growth curves of single plants for scenario 1

5.3.2. Case study 1: Based on weather data for Rotterdam, the Netherlands

The first case study with based on real weather data is for Rotterdam, the Netherlands. The dataset is retrieved from the Koninklijk Nederlands Meteorologisch Instituut (KNMI), which is a Dutch governmental organisation that measures weather data for the Netherlands. Measurement station number 344 corresponds with Rotterdam. From the full weather dataset, the 24-hour mean temperature and global radiation were taken. This data is handled as stated in the introduction of this section. First the data from the past 10 years is visualised (see Figure 5.26). In this figure, the data are already converted to the relevant values, which are the inside DLI PAR values and the inside 24-h mean temperature. Note that for this case study there is worked again with two crop-handling areas, because the reference case revealed that two crop-handling areas was preferred. However, it is dependent on the lot whether this is possible. For this case study it is assumed that the lot is reachable from two sides, and therefore two crop-handling areas is feasible.

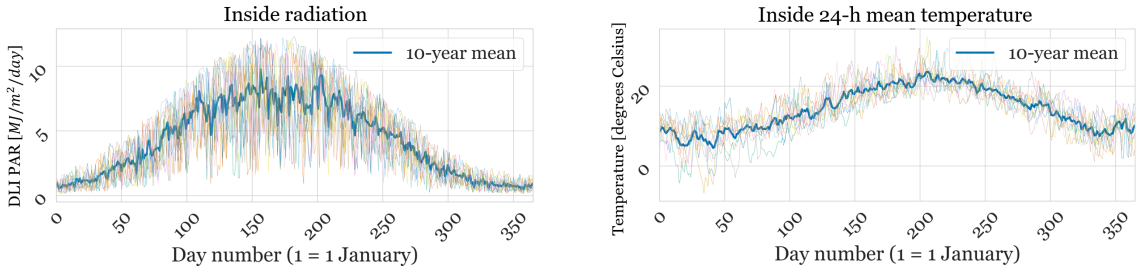


Figure 5.26: Data on inside DLI PAR (left) and inside 24-hour mean temperature (right) for the years 2012-2021, based on KMMI data for station number 344.

The same evalation procedure as for case study 0 is employed. The design is based upon the 10-year mean values in Figure 5.26. There is simulated with a constant price, and without crops lost due to illness. As starting point the capacities for the transport system and crop-handling tasks found

for the typical values for Rotterdam are used. These are used because for the weather data and the typical values the capacity is expected to be similar. The preliminary design (Figure 5.27) that is generated by the simulation model is similar to that of for the typical values. In Figure 5.28 is visualised how the two relevant KPIs for this evaluation step evolve over a year.

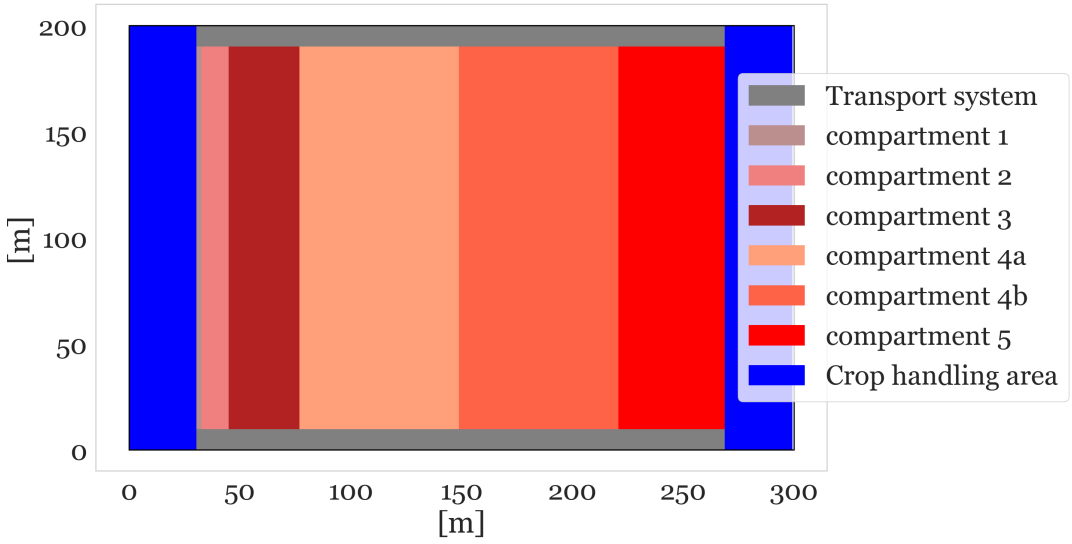


Figure 5.27: The preliminary design for weather data in Rotterdam.

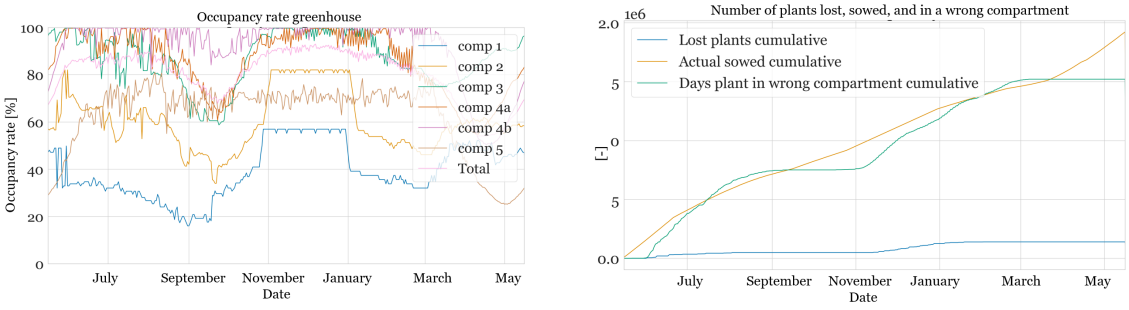


Figure 5.28: Case 1, preliminary design: simulated occupancy rate greenhouse (left) and plants lost (right) over the runtime.

Inspecting the occupancy rate greenhouse in Figure 5.28, leads to similar findings as for the typical values. Therefore, it is proposed to dedicate one row of Compartment 5 to Compartment 4b. In addition to that, there is enough space left to interchange one row of Compartment 1 and one of Compartment 2, with one row of Compartment 3. There is also room to enhance the sowing rate; this is done by +8%. The proposed increase in sowing rate is slightly less than for the typical values, because the share of crops lost is 7.2%, which is already too high. This higher share of crops lost is probably due to the higher variability than for the typical values. Even the 14-day moving mean of the 10-year mean of the weather data is more volatile than the typical numbers (see Figure 5.32 and 5.33). Another suggestion is to enhance the sowing rate during periods of increasing light intensities; this is done by applying a factor of 1.6 on the differential of the calculated sowing rate and add that to the sowing rate. The design for simulation with the proposed improvements is in Figure 5.29, while the corresponding evolvement of the relevant KPIs for this evaluation step is in Figure 5.30.

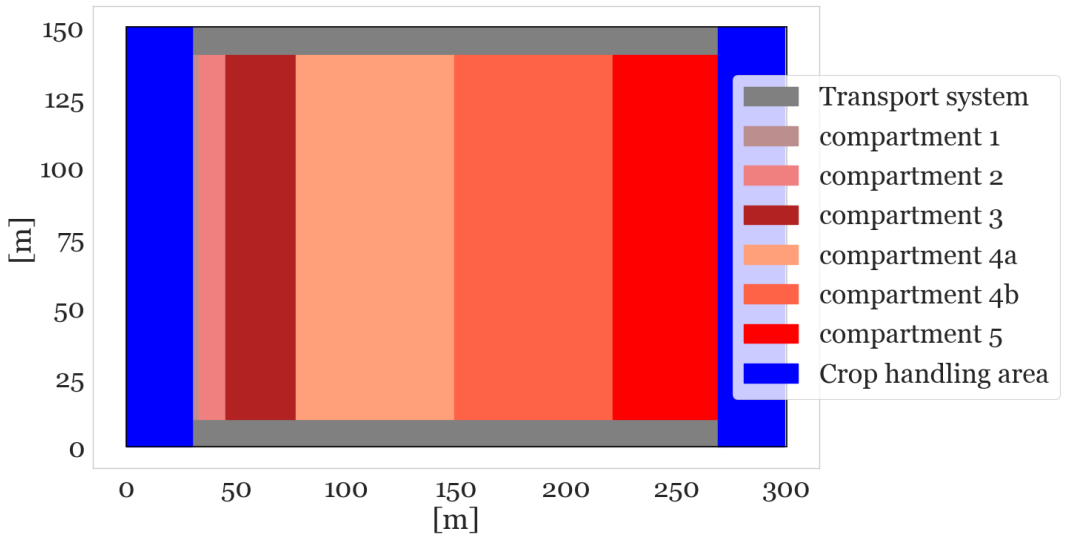


Figure 5.29: The improved design for weather data in Rotterdam.

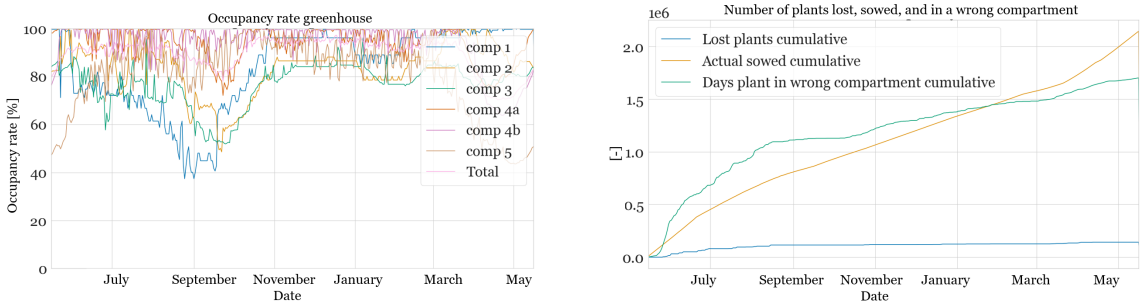


Figure 5.30: Case 1, improved design: simulated occupancy rate greenhouse (left) and plants lost (right) over the runtime.

The occupancy rate greenhouse for the improved design is substantially higher at 92.1%, compared to the preliminary design. The lost plants remain within the threshold at 6.6%. In Figure 5.31, the operational time for the crop-handling tasks and the maximum waiting time before pick up from a compartment is visualised. Regarding operational time, one can think about optimisation of the capacities. However, at this stage, there is still uncertainty about what the real capacities of individual machinery will be. Therefore, the implemented capacity, based on a sample list of equipment, as discussed in the reference case, is considered acceptable. Regarding the transportation system, there is a share of lost plants due to over-occupation of 0.73%, which is below the 1% threshold. Moreover, it is bigger than 0%, which implies that the capacity is minimised. Therefore, the transport capacity is considered feasible as well. A more thorough discussion about this evaluation step is done in the previous sub-section.

Next, there is simulated for five individual years. These results are in Table 5.5. There are variations from year to year, but the standard deviation from the mean for the production and turnover are smaller than 2%, which is small. Therefore, the design is considered robust for the dataset. The input temperature is in Figure 5.32, whereas the input DLI PAR is in Figure 5.33.

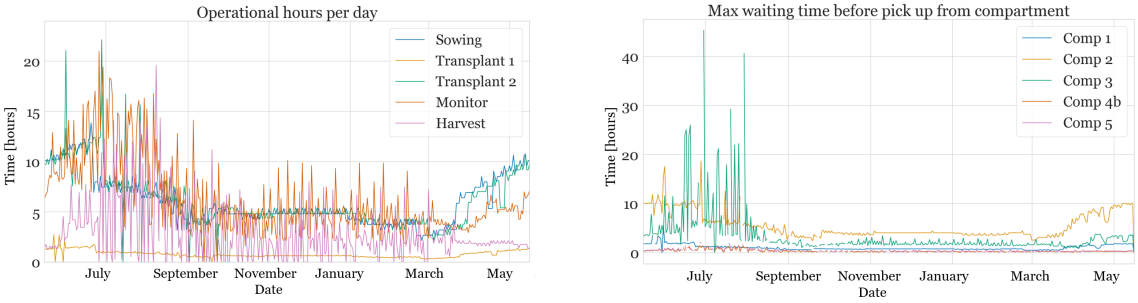


Figure 5.31: Case 1, improved design: operational time of a machine per day (left) and maximum waiting time before pick up from a compartment by a shuttle (right) over the runtime.

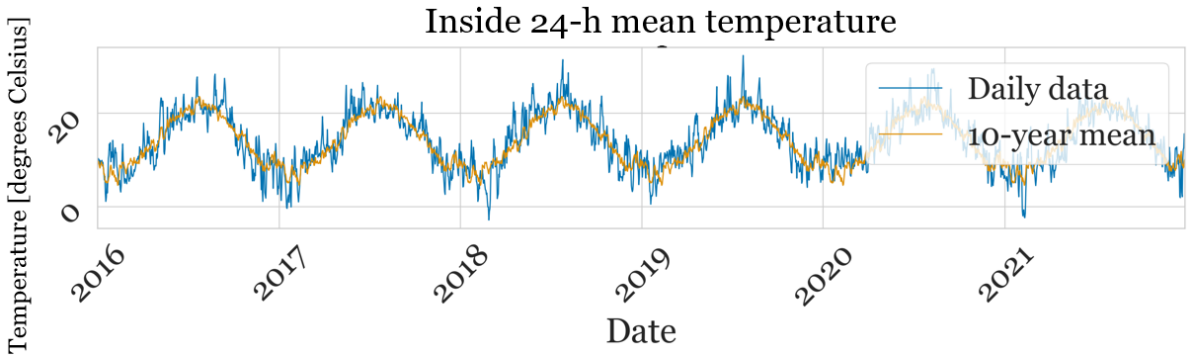


Figure 5.32: The input data on inside 24-h mean temperature for Rotterdam, based on KNMI data for station number 344.

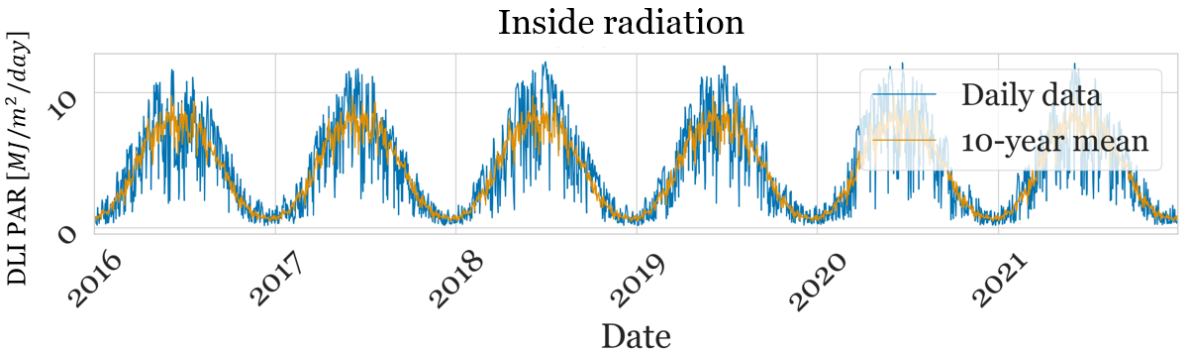


Figure 5.33: The input data on inside DLI PAR for Rotterdam, based on KNMI data for station number 344.

The conclusion drawn from this table is that the system performed well with the new inputs. As illustration, in Figure 5.34, the accumulation of cumulative production and turnover for run 0 are visualised over a year. The standard deviation is much significantly less than 1/20 of the mean value, for the KPIs occupancy rate greenhouse, production and turnover. This is considered small, which implies a robust behaviour. Only for lost plants, the standard deviation is bigger. It depends on the cost of a plant whether this will be a threat to the profitability, something which is beyond the scope of this research project. In this subsection, zero knowledge about the upcoming weather conditions was assumed. In real operations, a trend in weather can be forecast up to a couple of weeks in advance, which enables an improved sowing strategy. This can even enhance the production, and decrease the number of plants lost. In addition, the climate conditions per compartment could be slightly varied if the climate deviates from the foretasted, and therefore desired, values. Inspecting the final truss weight for this test substantiated the need for different climate conditions per compartment. The spread in truss weight is slightly stretched (Figure 5.35) due to a systematic difference in climate conditions in the growth period for certain batches. By changing the temperature, especially in the final growing phase, the ripening speed can be optimised for the desired truss weight. In addition, if there is an imbalance in

Table 5.5: Results for weather data of five years, using the curves of Figure 5.32 and 5.33. 'std' means standard deviation of the mean

Run	Year	occupancy rate greenhouse	Lost plants	Production kg/m^2	Turnover €$/m^2$
0	10-year mean	92.1	6.6	69.7	83.6
1	2016-2017	91.4	5.4	70.4	84.4
2	2017-2018	91.4	5.6	70.3	84.3
3	2018-2019	90.4	4.3	73.3	88.0
4	2019-2020	91.3	8.4	70.7	84.9
5	2020-2021	90.7	7.2	71.9	86.3
Mean ±std		91.0 ±0.4	6.2 ±2.1	71.3 ±1.1	85.6 ±1.4

temperature and light, one or both can be slightly adjusted. Therefore, smart operations can decrease the spread in truss weight.

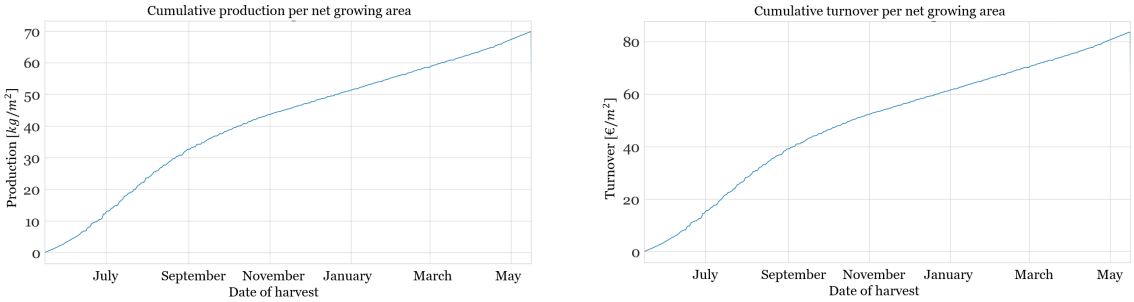


Figure 5.34: Case 1, improved design: cumulative production (left) and turnover (right) per m^2 net growing area over the runtime

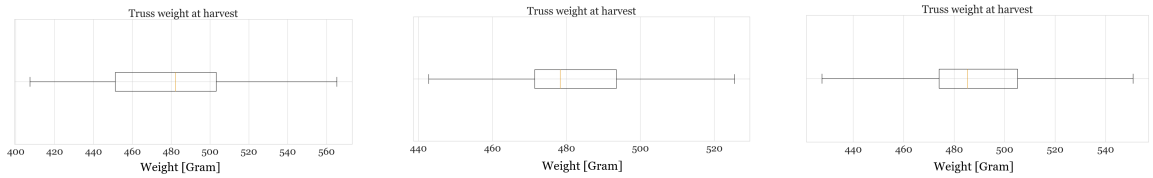


Figure 5.35: Case 1, improved design: distribution in final truss weights for runs of 2018-2019 (left), 2019-2020 (middle), and 2020-2021 (right)

5.3.3. Case study 2: Based on weather data for Darwin, Australia

For this case study, the same area of 300 x 150 [m] is used. In this case data for Darwin, Australia is used. The data is handled similarly as in the previous case. The dataset is retrieved from the Australian governmental bureau of meteorology and visualised in Figure 5.36. Again, this data was converted to inside data. The dataset differs much from Rotterdam. The two most important differences are that 1) the seasonal variation is less, and 2) the temperature and light intensity are substantially higher. In addtation to that, the winter and summer season are in the opposite period of the year. The hypothesis is that a higher light intensity and less variable conditions lead to a higher output of the greenhouse. The opposite seasons do not influence the performance.

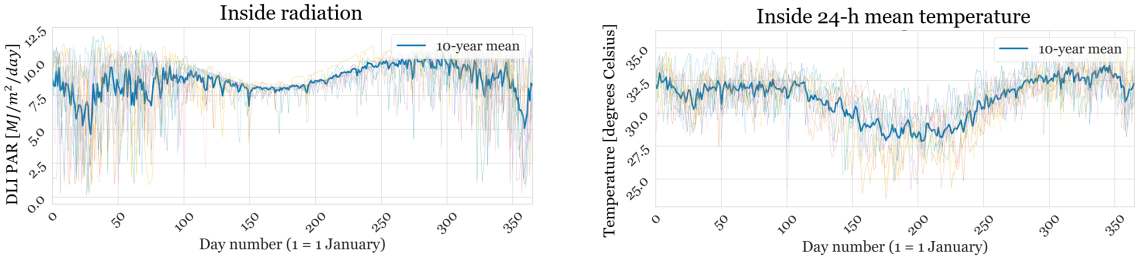


Figure 5.36: Data on inside DLI PAR (left) and inside 24-hour mean temperature (right) for the years 2012-2021, based on Australian weather data for Darwin.

For the 10-year mean values in Figure 5.36, a design is generated (Figure 5.37). The corresponding evelovment of the two relevant KPIs for the first evaluation step are in Figure 5.38. The capacity of the transport system and for the handling-tasks is doubled in respect to case 0 and 1, because it is expected that the higher temperatures and light intensities lead to a higher production. What can be observed is that Compartment 1 and Compartment 5 are too large. Therefore, it is suggested to remove one row from Compartment 1 and one row from Compartment 5. There is one row added to Compartment 3 and one to Compartment 2. These improvements lead to a similar design as for case study 1 (Figure 5.29). The sowing rate is enhanced by 8%. The outputs of the run for the enhanced sowing rate and improved design are in Figure 5.39.

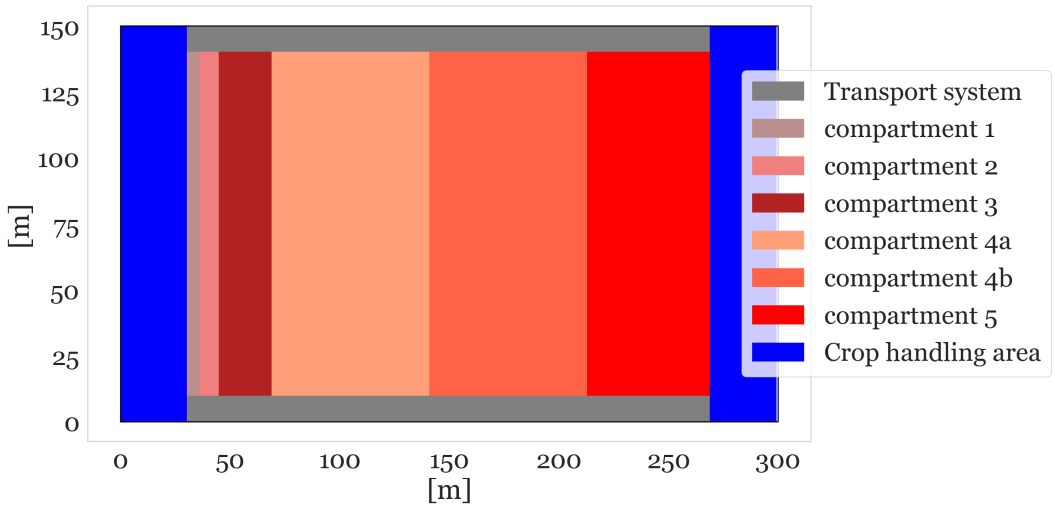


Figure 5.37: Case 2: The preliminary design for weather data in Darwin

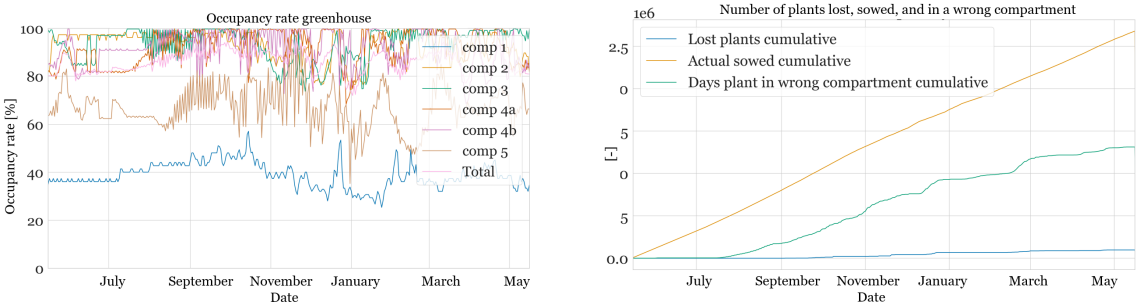


Figure 5.38: Case 2, preliminary design: simulated occupancy rate greenhouse (left) and plants lost (right) over the runtime

The improved sowing rate and design lead to a production which is around 75% higher for Darwin

than for Rotterdam. This is not only caused by a higher production during peak season; the more constant production plays a crucial role. The figures in Figure 5.39 are more straight than in Figure 5.34. Straight lines means a constant production. Furthermore, by improving the design and sowing rate, the thresholds on occupancy rate greenhouse and lost plants are met with respectively 91.7% and 5.2%.

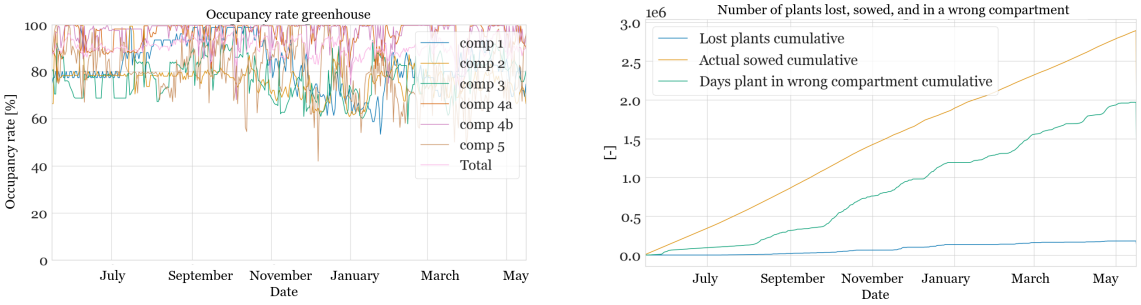


Figure 5.39: Case 2, improved design: production (left) and turnover (right) over the runtime

The next step is to assess the capacity for the transport system and the crop-handling tasks. In Figure 5.40, the maximum waiting time before pick-up from a compartment and the operational time per crop-handling tasks is given. Inspecting that figure gives lower peaks than expected. There was expected that due to the higher production, the capacites should be higher as well. However, there is only a slight increase in peak capacity. Therefore, there was simulated again with similar capacitities as for Rotterdam. The outputs of this simulation are in Figure 5.41. The plants lost due to over-occupation of the transport system are 0.3% which is between the thresholds as stated in case 0. Therefore, the design and capacities are fixed from now on, and there is simulated for 5 individual years. The inputs for the 5 individual years are in Figure 5.42 and 5.43, while the outputs are in Table 5.6.

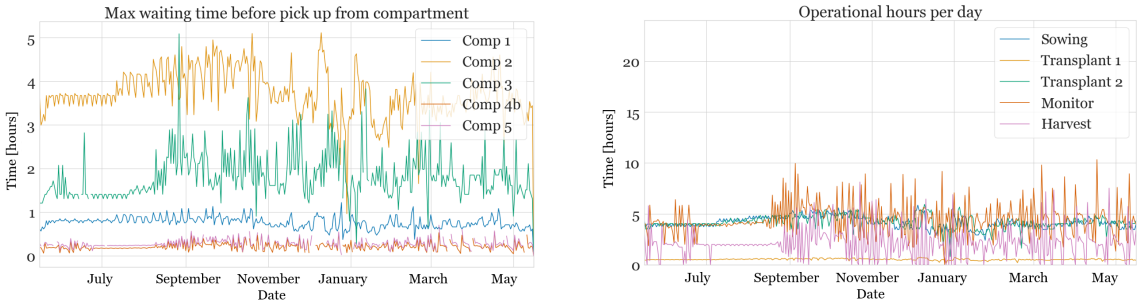


Figure 5.40: Case 2, improved design doubled capacities as for Rotterdam: the waiting time before pick-up from a compartment (left) and the operational time for machinery of a task per day (right) over the runtime

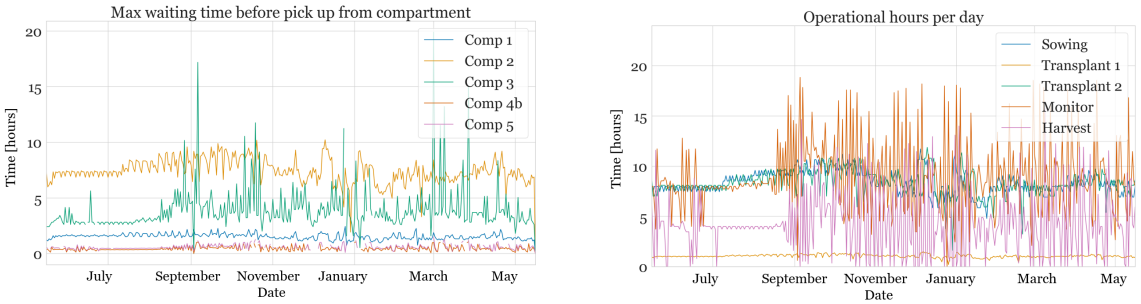


Figure 5.41: Case 2, improved design, similar capacities as for Rotterdam: the waiting time before pick-up from a compartment (left) and the operational time for machinery of a task per day (right) over the runtime

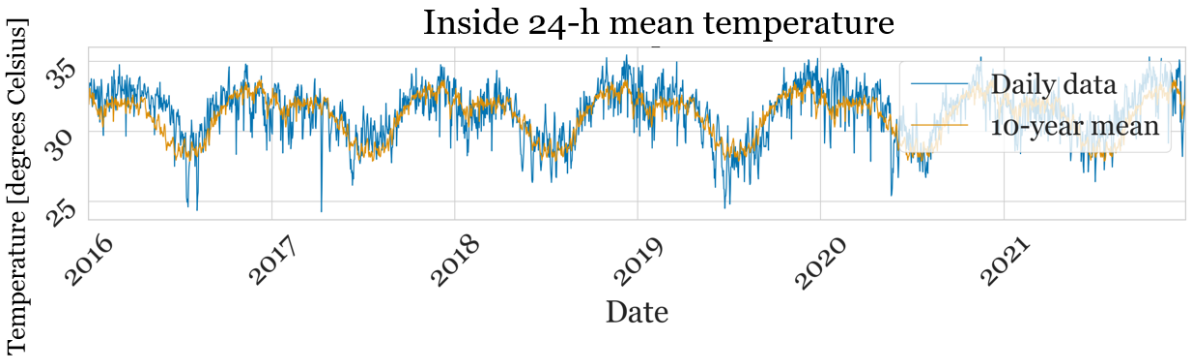


Figure 5.42: The input data on inside 24-h temperature for Darwin, based on data fro Australian governmental bureau of meteorology

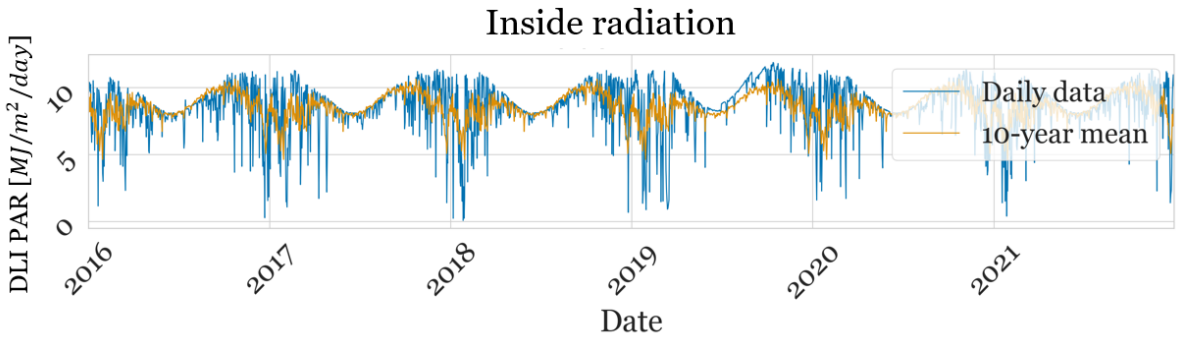


Figure 5.43: The input data on inside DLI PAR for Darwin, based on data fro Australian governmental bureau of meteorology

Table 5.6: Results for weather data of five years, using the curves of Figure 5.42 and 5.43. 'std' means standard deviation of the mean

Run	Year	Occupancy rate greenhouse	Lost plants	Production kg/m^2	Turnover €/m^2
0	10-year mean	91.0	5.2	125.8	151.0
1	2016-2017	91.0	5.6	120.1	144.1
2	2017-2018	90.8	4.8	125.5	150.7
3	2018-2019	90.3	6.3	124.6	149.6
4	2019-2020	91.0	5.2	125.8	151.0
5	2020-2021	89.4	5.5	119.7	143.7
Mean ±std		90.5 ±0.6	5.5 ±0.5	123.1 ±2.7	147.8 ±3.2

From Table 5.6 it can be concluded that the higher temperatures and light intensities are leading to a higher production. Figure 5.44 visualises the difference in length of stay per compartment. The lead time is shorter for Darwin than for Rotterdam. In addition to that, the length of stay is less variable. The smaller variability has the operational advantage that the stream of tomatoes is more constant. A disadvantage is that the impact of a change in lead time is fractionally larger. If the lead time for a compartment changes from 6 to 7 days makes fractionally a larger difference than from 10 to 11 days. Moreover, to the smaller total lead time, it makes a larger difference if the there is a week of relatively shady weather. This is because one week is a larger share of 40 days than of 60 days. This

difference leads to a larger spread in truss weight. The calculated variation in truss weight is for the last three simulated years visualised in Figure 5.45. This is considered too much, because the variation in truss weight is up to about 20% for the 99.7% confidence interval. Changing the temperatures per compartment to stir the growth in the right direction is the solution to deal with the changing weather conditions. The designed greenhouse has the power to regulate conditions per compartment, which can thigthen the the spread in truss weight. Another conclusion that can be drawn, is that also for this case the standard deviation is smaller than 1/20 of the mean for the KPIs occupancy rate greenhouse, production and turnover. Therefore, the design is considered robust.

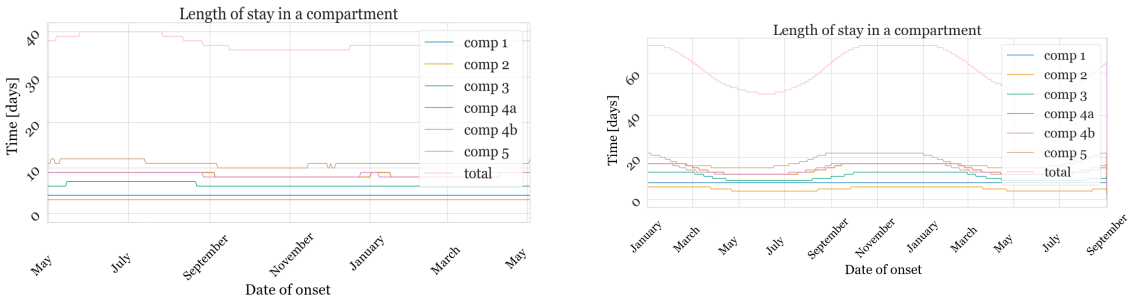


Figure 5.44: Lead time per compartment for the 10-year mean for Darwin (left) and Rotterdam (right), based on weather data.

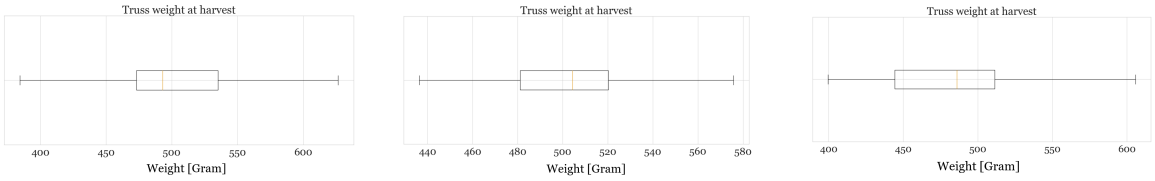


Figure 5.45: Case 2, improved design: distribution in final truss weights for runs of 2018-2019 (left), 2019-2020 (middle), and 2020-2021 (right).

5.3.4. Case study 3: Based on weather data for Melbourne, Australia

The final case study is Melbourne, Australia. For Melbourne, weather data is retrieved from the governmental institute of Meteorology as well. In Figure 5.46, data for 10 years is given, with also the 10-year mean indicated. Evident from this figure, is that there are large seasonal variations compared to the other case studies. In addition to that, the peaks in climate are higher. Moreover, in short periods of the year the temperature and light could drop beneath the thresholds of 3 $MJ/m^2/day$ and 17 degrees Celsius (substantiated in Section 4.1). Therefore, some artificial light and temperature is added when it drops below the thresholds.

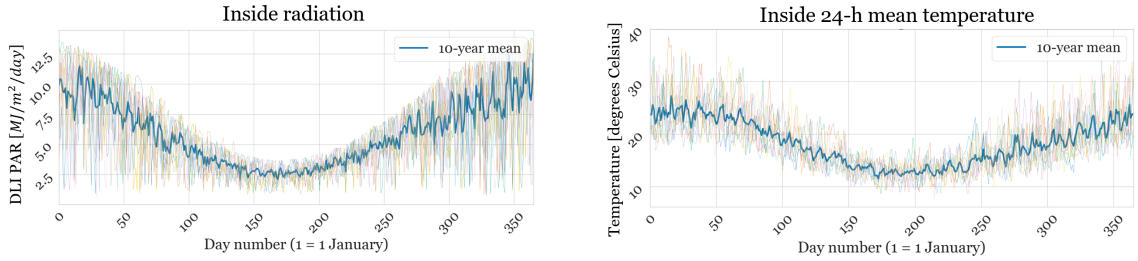


Figure 5.46: Data on inside DLI PAR (left) and inside 24-hour temperature (right) for the years 2012-2021, based on Australian weather data for Melbourne

The procedure is similar for this case as for the two previous cases. The first step is to generate and simulate a preliminary design. In Figure 5.47, this design is provided. In Figure 5.48, the corresponding

evolvement of the two KPIs for this evaluation step are visualised. The mean occupancy rate is 83.6% and the lost plants 6.0%. Most important observation from Figure 5.48 is that compartments 1, 2, and 5 seems to be large, while compartment 3 and 4b seems to be too small. Therefore, one row of compartments 1 and 2 is interchanged with one row of Compartment 3, while one row of Compartment 5 is dedicated to Compartment 4b. The sowing rate is enhanced with 8% and a factor of 1.6 is applied on the differential for increasing temperatures again (see case 1 for substantiation). As starting point, the same capacities are used as for case 2. At this stage the capacities are just high enough to handle peak season.

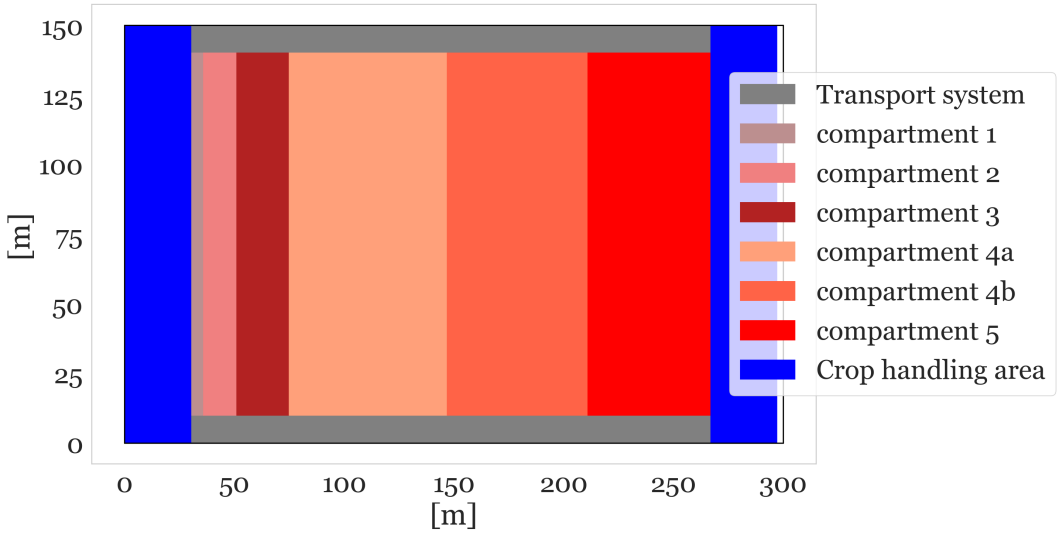


Figure 5.47: Case 3: The preliminary design for weather data in Melbourne

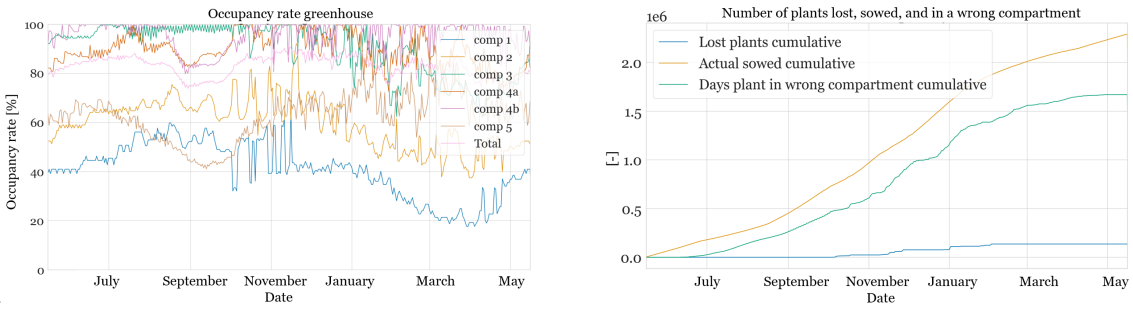


Figure 5.48: Case 3, preliminary design: simulated occupancy rate greenhouse (left) and plants lost (right) over the runtime.

Simulating with the proposed improvements implemented lead to an occupancy rate of 90.9%, but with a share of lost crops of 8.7%. This 8.7% is not respecting the threshold on lost plants. There is 3.5% of the crops lost due to over-occupation of the transport system. Therefore, the next step was to simulate again for the same design and sowing strategy, but with two transport lanes instead of one. For this new simulation, the design is the same as for case 1 and case 2, while the occupancy of the greenhouse and the share of lost plants are in Figure 5.49. The occupancy rate equals 91.8% and the share of lost plans decreased to 5.2%, which is within the thresholds.

The occupancy of the crop-handling system is high, because it is approaching 24-hours operational per day during peak season. The crops lost due to waiting too long before pick up is 0%. For a transport lane less, it is exceeding the threshold. Therefore, both the capacities of the transport system and for the crop-handling system is considered viable. The outputs are in Figure 5.50.

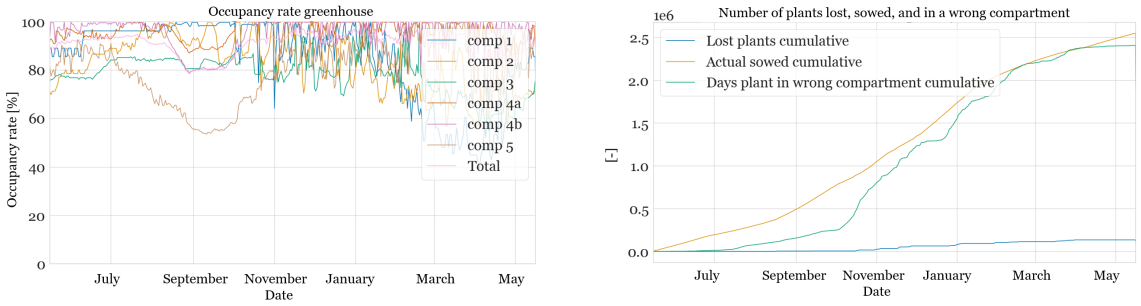


Figure 5.49: Case 3, improved design: simulated occupancy rate greenhouse (left) and plants lost (right) over the runtime.

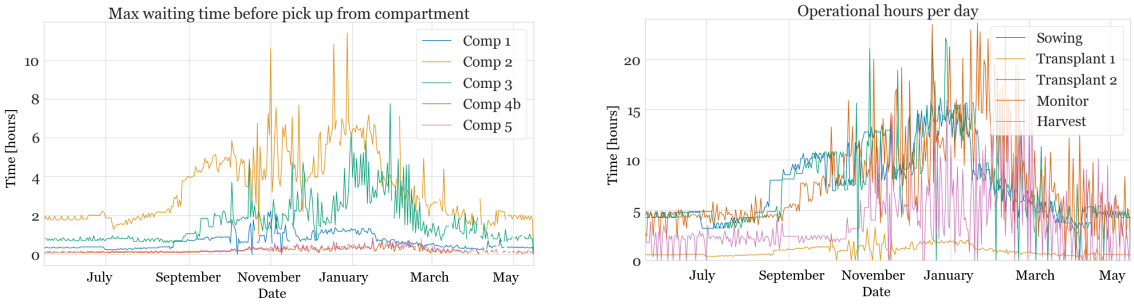


Figure 5.50: Case 3, improved design, similar capacities as for Melbourne: the waiting time before pick-up from a compartment (left) and the operational time for machinery of a task per day (right) over the runtime.

The next step is to simulate for 5 individual years. The input signals are in Figure 5.51 and 5.52. The results are in Table 5.7. The conclusion drawn from this table is that the system performed well with the new inputs. In Figure 5.53, the KPIs production and turnover are visualised for the 10-year mean. The standard deviation is less than 1/20 of the mean value, for the KPIs occupancy rate greenhouse, production and turnover. This is considered small, which implies a robust behaviour. Only for lost plants, the standard deviation is bigger. It depends on the cost of a plant whether this will be a threat to the profitability, something which is beyond the scope of this research project. In this subsection, zero knowledge about the upcoming weather conditions was assumed. In real operations, a trend in weather can be forecast up to a couple of weeks in advance, which enables an improved sowing strategy. This can even enhance the production, and decrease the number of plants lost. In addition, the climate conditions per compartment could be slightly varied if the climate deviates from the foretasted, and therefore desired, values. Inspecting the final truss weight shows that for this case the final truss weight is within the acceptable range (see Figure 5.54). The greenhouse show a similar behaviour as for case 1, only with a higher temperature and light intensity, the production and capacity, in terms of number of plants, is higher.

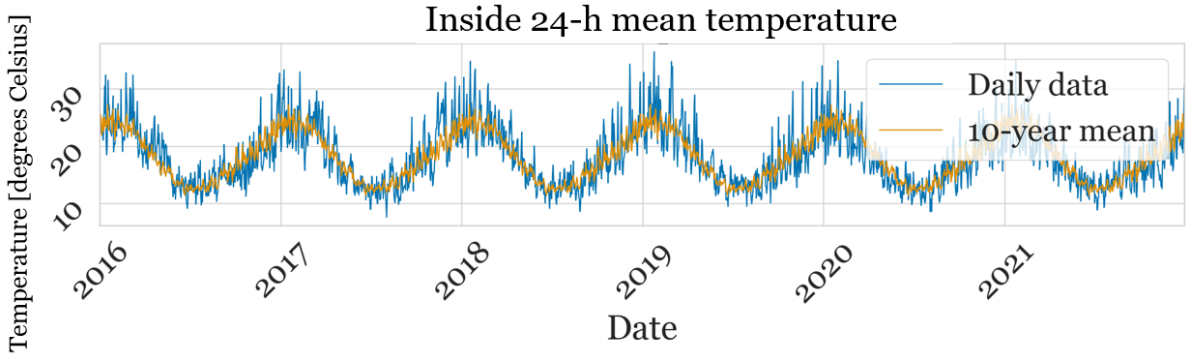


Figure 5.51: The input data on inside 24-h temperature for Melbourne, based on data fro Australian governmental bureau of meteorology. Note that values beneath the lower threshold for temperature are set equal to this threshold of 17 degrees Celsius

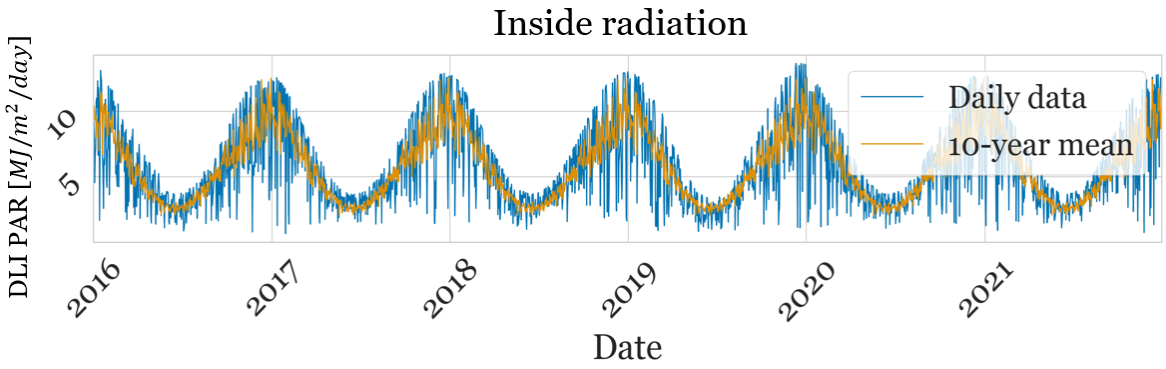


Figure 5.52: The input data on inside DLI PAR for Darwin, based on data fro Australian governmental bureau of meteorology. Note that values beneath the lower threshold for DLI PAR are set equal to this threshold of 3 $MJ/m^2/day$

Table 5.7: Results for weather data of five years, using the curves of Figure 5.51 and 5.52. 'std' means standard deviation of the mean.

Run	Year	occupancy rate greenhouse	Lost plants	Production kg/m^2	Turnover €$/m^2$
0	10-year mean	91.8	5.2	86.8	104.1
1	2016-2017	90.2	7.1	85.9	103.1
2	2017-2018	90.2	4.2	88.1	105.7
3	2018-2019	90.4	4.2	87.5	105.0
4	2019-2020	91.8	5.2	86.8	104.1
5	2020-2021	91.5	6.8	85.7	102.9
Mean ±std		90.8 ±0.7	5.5 ±1.5	86.8 ±0.9	104.1 ±1.1

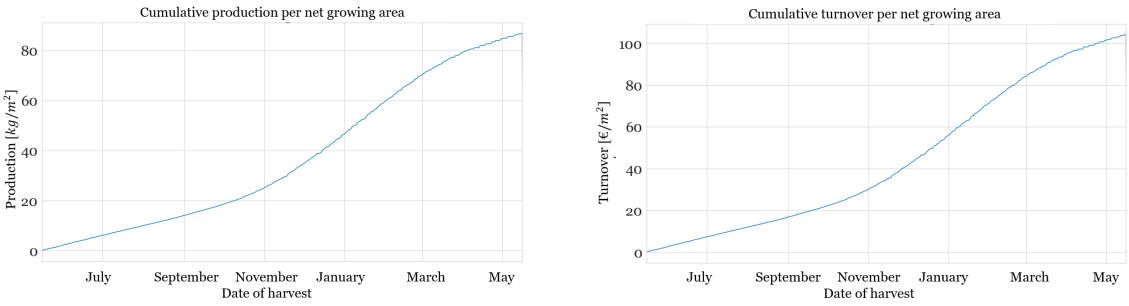


Figure 5.53: Case 3, improved design: cumulative production (left) and turnover (right) per m^2 net growing area over the runtime.

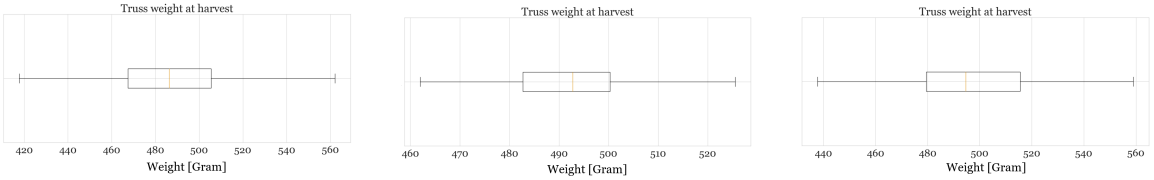


Figure 5.54: Case 3, improved design: distribution in final truss weights for runs of 2018-2019 (left), 2019-2020 (middle), and 2020-2021 (right).

5.4. Discussion

The main goals of this chapter were to answer the fourth sub-question and to prove that the design framework is generic. This was done by employing the evaluation protocol (Figure 5.1) on different case studies at geographic locations with very different climate conditions. To summarise the results of these case studies, Table 5.8 is provided. The table shows that, in accordance with the sensitivity analysis, the production and turnover are higher for locations with more light. The KPI lost plants is considered high, and the standard deviation is also high in respect to the mean. That implies that if weather conditions vary from expected this will fractionally sort most effect in the share of lost plants. This share of lost plants could be reduced by varying the weather conditions per compartment in the operational phase. There is simulated for zero knowledge about upcoming weather, and no regulating measures in summer. In real operations, there is some knowledge about the weather upcoming two weeks, and regulating measures can be taken. Therefore, the share of lost plants can be reduced.

Table 5.8: Simulation results for the case studies for simulating five individual years. 'std' means standard deviation of the mean

Case study	Location	occupancy rate greenhouse	Lost plants	Production kg/m^2	Turnover €$/m^2$
1	Rotterdam, the Netherlands,	91.0 ±0.4	6.2 ±2.1	71.3 ±1.1	85.6 ±1.4
2	Darwin, Australia,	90.5 ±0.6	5.5 ±0.5	123.1 ±2.7	147.8 ±3.2
3	Melbourne, Australia	90.8 ±0.7	5.5 ±1.5	86.8 ±0.9	104.1 ±1.1

An even more important application of these regulating measures is in achieving a more constant truss weight. Especially, in the last compartment, the growth can be steered in the desired direction by changing the temperature or the truss density. At that stage, there is knowledge about the plant development so far, and the upcoming weather for the last growing stage. By using that knowledge, the truss weight can be brought closer to the desired value. The implemented zero knowledge is therefore a conservative assumption. Releasing this assumption was beyond the scope of this research project, because that belongs to the operational model. Parts of the developed simulation model can be used for an operational phase, but that was not the purpose of this study.

For every case study, the area allocation of the conceptual design was similar. However, for the preliminary design there was a small variation. That the final design is similar is logical, because for different climate conditions, the lead time per compartment as fraction of the total lead time is always almost similar. Furthermore, the fraction of light per compartment is similar as well. The area allocation will be affected when the growth dependent parameters get an update when there is a validating study performed for the crop growth model. Another definition of growth stages, and variation of climate conditions between compartments will affect the area allocation as well.

Although, the new concept is totally different from the conventional concept, it is interesting to relate the simulated performance of the new concept to the performance of the conventional concept. The remark must be made that comparing two different concepts, of whom one is simulated and one is from literature contains arbitrary elements. Because these two concepts are so different, it was not possible to simulate the conventional version in the developed simulation model of this study. However, the

simulated performance should be at least competitive with the conventional concept to mark it as potentially economic viable.

In literature, there is found for the Netherlands that the reference production in greenhouses is within 50-70 kg/m^2 (Vermeulen, 2016). The Dutch governmental statistics institute (CBS) reports an average production of about 50 kg/m^2 for Dutch nurseries for the last three years (StatLine, 2022). Therefore, the production for Rotterdam of 71.3 kg/m^2 is considered competitive. Verheul et al. 2019 did a research to the effect of artificial light in Norway. In this study, a yield potential of 125-140 kg/m^2 for year round cultivation was discovered. For this experiment, much HPS light was added. The conditions were approaching optimal conditions for tomato yield production. The situation in Darwin shows similarities to this case. With 123 kg/m^2 it is approached the values found by Verheul. Therefore, this production is considered competitive as well. The Australian Department of Primary Industries of New South Wales did study to the potential yield of Northern Victoria and New South Wales (NSW Department of Primary Industries, 2020). Melbourne is located in Northern Victoria. The study of the local government reports that the current yield production is around 80 kg/m^2. The future target set equals 200 kg/m^2. There are quite some steps to take to reach that target. However, the new concept is competitive for Northern Victoria with a production of 104.1 kg/m^2. Moreover the report of the local government states that it is needed to enhance the production to remain profitable in times of water scarcity. The new concept is considered beneficial in terms of resource utilisation due to the high truss density and continuous production. Therefore, the new concept can contribute to reaching that target.

5.5. Conclusion

The generated results in this chapter answer the fourth sub-question: *How can the conceptual design be evaluated, and the simulated performance of the conceptual design be improved?* The evaluation protocol (Figure 5.1) is the guideline for evaluating and improving. The four KPIs are used in that protocol. The turnover per net m^2 growing area is used for the sensitivity analysis and the optimal dimension analysis. The conclusion is that the turnover is sensitive to the amount of light and the turnover increases when there are more rows. This trend of a higher turnover for more rows is stronger when the total amount of rows is lower.

After these general statements, case studies are employed for the next evaluation steps. The case studies proved multiple things. First, the developed design framework is generic. For all cases, with very different climate conditions, the design framework was applicable. A preliminary design was generated, an improved version was made iteratively, the capacities for the transport system and crop-handling tasks could be reviewed, and the designs with corresponding strategy were robust. The solutions are considered robust, because with simulating for the individual years, the standard deviation was smaller than 1/20 of the mean. Second, the different case studies proved that the turnover is sensitive to the light intensity. In discussion in the next chapter, there is discussed how the position of the new concept could be. In that chapter also conclusions are drawn and recommendations for follow-up studies are provided.

6

Discussion, conclusion & recommendations

This chapter starts with a discussion in Section 6.1. That section discusses in a broad sense the new conceptual designs and their simulated performance obtained by employing the developed generic design framework. Next, the conclusion in Section 6.2 reflects on the research and answers the main research question. The conclusion is followed by recommendations, in Section 6.3, for follow-up studies.

6.1. Discussion

In the discussion in Section 5.4, the simulated performance, of the new concept, obtained by employing the design framework, is related to the production in conventional concept. The production is competitive, but there are more aspects that determine whether the new concept could become the new standard in tomato cultivation. An important advantage of the new concept, is that it is considered an automation-facilitating concept. However, at this stage, the availability of automation solutions for crop-handling tasks on the market is limited. There are sample automation solutions discussed in Section 2.4.4 which can potentially implemented shortly. Although automation of crop-tasks is desired, the new concept does not need automation for a gain in efficiency compared to the conventional concept. Especially in the cultivation of orchids (Ter Laak Orchids, 2019), the advantages of a movable-crop concept are evident. At Ter Laak, important crop-handling tasks are performed by human labour ergonomically in the central crop-handling area. This ergonomically position leads to more efficiency and is more comfortable for the employee. Moreover, the climate where the employees work in is adjusted to the desires of the employees, while the climate inside the greenhouse fully serves crop growing. Another advantage of the new concept is that the cultivation cycle is about a factor 5 shorter, which reduces the severeness of illness. If a plant becomes ill, less tomatoes are lost for a short cultivation cycle than for a longer cycle.

Performing the crop-handling tasks in a central area has also the advantage that walking paths can be removed. This leads to a higher plant density, and a fall of required walking times by human labour, which is also a gain in efficiency. Another advantage is that the human - robot interaction is clear. If there had been chosen to implement robots in the conventional work environment, there would have been issues to resolve regarding the human - robot interaction. Because a robot can move more freely than standard equipment and could drive autonomously, one should also think about the questions in ethics about responsibility and liability for collisions. This is elaborately discussed for automated guided vehicles (AGVs) on, e.g. public roads, but there is no unambiguously answer formulated to these ethics questions (e.g. Santoni de Sio (2016), Matthias (2004), and Asaro (2011)). However, this responsibility gap can be mostly overcome by letting the AGVs always stop if there is something in the way. Letting an AGV always stop will be a threat to efficient operation of robots. This is an issue that could be resolved, but is a complication in the other approach of implementing robots in a conventional environment.

The simulated high production per m^2 compared to the conventional concept, indicates that the new concept makes efficient use of the area. The smaller plants, absence of walking paths, and having all growing stages present in the greenhouse at the same time, leads to a high truss density. This truss density is an important reason for the high production. In addition to the high production, a high truss density also makes efficient use of the area available. It depends on the geographical location on whether lots are scarce. However, there is a general trend, that sustainability becomes more important. In the context of tomato cultivation, efficient use of resources is an essential pillar for sustainable operations. A high truss density also leads to a high interception of light and heat. For the conventional concept, in the first growing stages, the truss density is lower than it could be, because all the plants in the greenhouse are small at this stage. Therefore, not all light and heat is intercepted by the plant and therefore lost. Energy systems in greenhouses are nowadays highly complex systems. It is another study to research the energy system in a greenhouse. Therefore, there cannot be quantified to what extent the new concept is more energy efficient. However, due to feature of that it intercepts light and temperature as efficient as possible, through the whole year, it is considered that the new concept makes more efficient use of the resources available and is therefore more sustainable than the conventional concept.

Furthermore, a cost study could be a great addition to the design framework. A price curve was implemented, to enable the case-base design generator to make choices based on maximising turnover. However, it could be more interesting to base the design on maximisation of profit. This could lead to more sophisticated strategies, including, for example, decisions in which both the unit selling price and costs as costs of seeds, heating and lighting are incorporated. To achieve this, all costs must be quantified. The simulation model is build in a way that this could easily be implemented. Implementing these costs could help in gaining more insight into the potential profitability of the new concept.

A threat to the new concept, related to the cost, can be the gas crisis that has impacted North-West Europe. The framework showed that for year-round production, there is often some artificial light and supplementary heat needed. For the Netherlands, it is expected that only the most energy efficient greenhouses will survive (Gersdorf, 2022); this new concept is considered such an energy efficient greenhouse. Furthermore gas crisis led to higher prices of steel and aluminium, which could also be a cost related threat to the new concept. On the other hand, there is trend observed that there is growing demand for sustainable and locally produced food, a trend accelerated by the COVID-19 crisis (Pedersen and Hansson, 2021). Undoubtedly, this trend will be strengthened due to the cut in wheat supply in 2022 because of the war in Ukraine (Carriquiry et al., 2022). Cultivation in efficient greenhouses can play an important role in producing food locally. The new concept researched in this study can contribute to that trend.

6.2. Conclusion

This section formulates and answer to the main research question: *How to develop a generic conceptual design framework for a tomato greenhouse with the limited-truss tomato on a movable-gutter concept?* To answer to the main research question, the sub-questions are answered first.

Sub-question one: What could be a viable path towards automation in tomato greenhouses? This question reflects on the last part of the main question. There are two paths towards automation: 1) designing robots for the conventional environment, 2) designing a concept which facilitates automation. Reviewing other crops, reveal that crops with a movable-crop concept appears at a higher level of automation. In these movable-crop concept, crops are transported to a central-crop handling area, where the handling tasks are performed. For the greenhouse tomato it is required to apply a limited-truss tomato on a movable concept for feasible transportation. This limited-truss concept, produces less tomatoes, but has more successive cycles and a higher plant density than the conventional concept. Te more successive cycles and higher plant density, ensure that the production for the new concept is at least competitive with the conventional concept. Because competitive production and

automation-facilitating aspect, this limited-truss tomato on a movable-gutter concept is a viable path towards automation in tomato greenhouses.

Sub-question two: What modelling methodologies can be used to generate and simulate conceptual designs? The first important step in answering this question was developing the structure of the design framework. Part of this framework is the simulation model with three modules. The first module models the growth of a greenhouse tomato. The existing model of De Koning (1994) is suitable for this module. The relationships of the De Koning model were used, but the approach in which the model was applied is different. Based on the output of the crop growth module part, a conceptual design was generated. The MILP approach was suitable to generate an optimal conceptual design. Finally, this design was simulated to examine the performance. The OO-DES protocol is applicable to fulfill that task.

Sub-question three: How should the new concept be implemented in a simulation model? The main building blocks implemented in the simulation model are summarised in the scheme of Figure 6.1. The implementation is done using the spiral workflow, which means that there is started relatively simple, and that complexities are added step by step. Verification is also part of the implementation task. The verification proves the model behaves as expected, whereas the validation and experimentation are left to follow-up studies, because there were no reference systems or test-setup available or feasible to build.

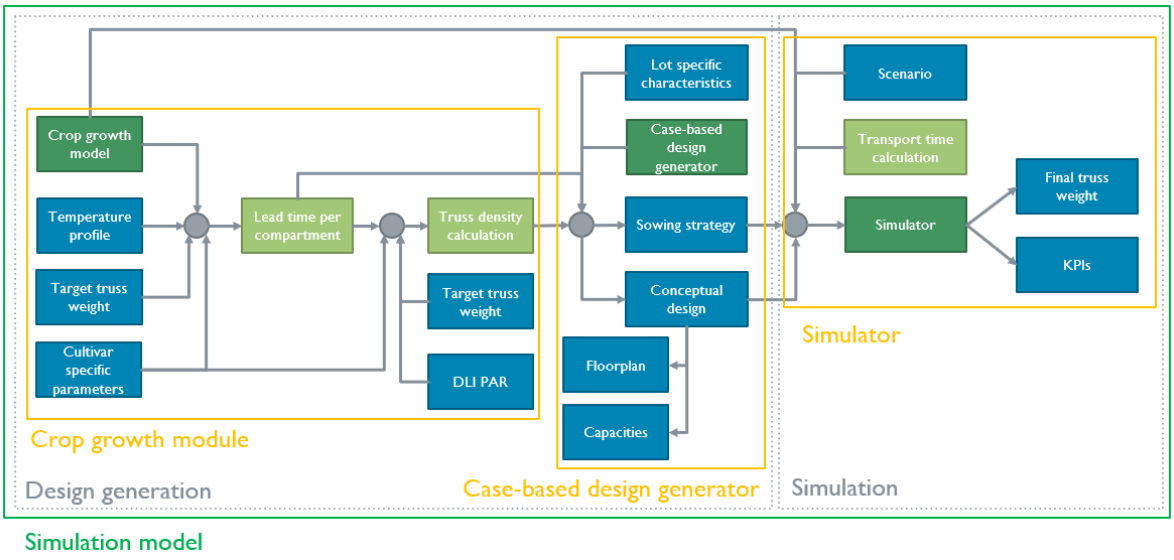


Figure 6.1: Flow of the simulation model. The inputs and outputs are in blue, while the three modules are in darker green.

Sub-question four: How can the conceptual design be evaluated, and the simulated performance of the conceptual design be improved? Central in the answer to this question is the evaluation protocol in Figure 6.2. First, the lower boundaries for light and temperature are examined and set equal to a minimum DLI PAR of 3 $MJ/m^2/day$ and a minimum 24-h temperature of 17 degrees Celsius. The turnover is sensitive to the amount of light and the turnover increases when there are more rows. This trend of a higher turnover for more rows is stronger when the total amount of rows is lower. For the next evaluation blocks in that scheme, case studies are employed for three different geographical locations on earth. Executing the evaluation protocol and the defined underlying procedures, which are using the KPIs, will lead to an improved design which meets the defined thresholds. Moreover, employing the evaluation protocol and using the case studies proves that the design framework developed in this study is applicable to multiple, very different, cases. Therefore, the developed framework is generic.

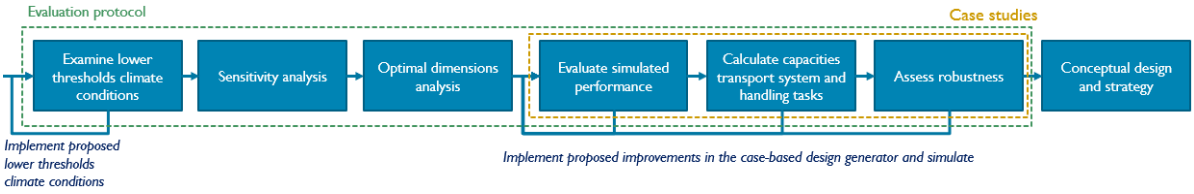


Figure 6.2: Protocol applied for evaluation of the conceptual design and improvements of the simulated performance.

Main research question: How to develop a generic conceptual design framework for a tomato greenhouse with the limited-truss tomato on a movable-gutter concept? The conceptual design framework developed in this study suits the problem well. The framework, schematically provided in Figure 6.3, is all-encompasing and generic. By following the steps in this study, conceptual designs can be generated and evaluated for multiple cases. The applied model-based simulation in the framework is capable to explore different design solutions and to identify bad design choices before significant resources are spent. Moreover, designs generated by the framework are automation-facilitating and show a simulated performance which is competitive with the conventional concept. The developed generic framework covers the conceptual design phase. Therefore the next design phase, in which greater detail is added and validations are done, can start.

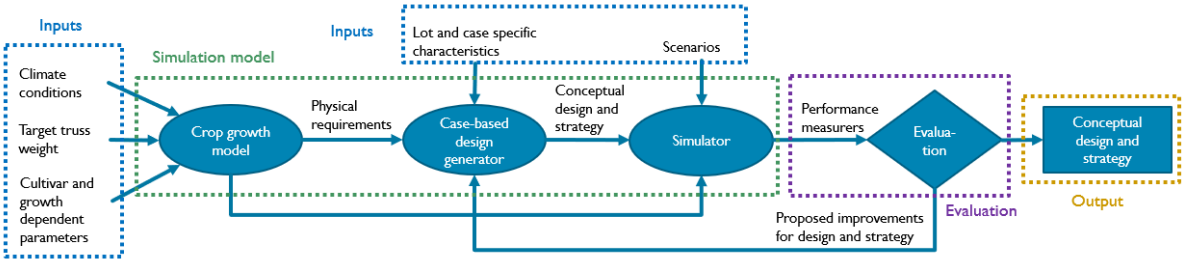


Figure 6.3: Structure of the generic conceptual design framework

6.3. Recommendations

The following is a list of recommendations for further steps in research:

- Validate the applied crop-growth model for the new cultivation concept. This validating study should research whether the relationships still hold. Additionally, these relationships should be specified per growing stage, indicating how they can differ. A substantiation of the definition of the different growing stages should also be included.
- Propose a reduced list of possible truss densities for a compartment. Generally, more discrete options in truss density lead to a truss density closer to the optimal calculated continuous value. However, this view implies the undesirable effect that more connection points to the watering system must be built. Further research should produce an optimal list of possible truss densities per compartment.
- Make a preliminary or even detailed design of the transport system. There is no design for the application of gutters with greenhouse tomatoes. There are systems available for transporting lettuce, which has overlap with transporting gutters for greenhouse tomatoes. Therefore, design knowledge can be gained from those systems, but they need to be redesigned. Once the design is developed, it can be implemented in the model. The current numbers regarding the capacity should be the starting point for the design of the transport system.
- Investigate and substantiate the performance of human labour and robots. In addition, an ongoing monitoring of the development of robotics for tomatoes should be maintained.

- Add greater detail to the model developed in this study. The simplifications and assumptions should be updated (e.g. the constant LUE or transmission rate). Furthermore, the operational times could be updated from 24/7 to more realistic times.
- Conduct a cost analysis and assess the new concept's financial feasibility. This could lead to more sophisticated strategies, including, for example, decisions in which both the unit selling price and costs as costs and costs of seeds are incorporated. To achieve this, all costs must be quantified and implemented in the model. Once this is done, decisions can be made based on profit maximisation.
- Conduct a study on the quantification of the energy efficiency of the new concept, and examine its sustainability.
- A validation of the overall model should be conducted. The first step is developing a small-scale test setup.

Bibliography

13002-2, N.-E. (2014). *Nederlandse norm: Veiligheid van hijskragen - algemeen ontwerp - deel 2: Belastingen*. Normcomissie 345002.

Accorsi, R., Cholette, S., Manzini, R., Pini, C., & Penazzi, S. (2016). The land-network problem: Ecosystem carbon balance in planning sustainable agro-food supply chains. *J. Clean. Prod.*, (112), 158–171.

Achour, Y., Ouammi, A., & Zejli, D. (2021). Technological progresses in modern sustainable greenhouses cultivation as the path towards precision agriculture. *Renewable and Sustainable Energy Reviews*, *147*(111251). https://doi.org/10.1016/j.rser.2021.111251

Ahumada, O., Villalobos, J., & Mason, A. (2012). Tactical planning of the production and distribution of fresh agricultural products under uncertainty. *Agric. Syst.*, (112), 17–26.

Asaro, P. (2011). A body to kick, but still no soul to damn: Legal perspectives on robotics.

Banasik, A., Kanellopoulos, A., Bloemhof-Ruwaard, J., & Claassen, G. (2019). Accounting for uncertainty in eco-efficient agri-food supply chains: A case study for mushroom production planning. *J. Clean. Prod.*, (216), 249–256.

Banasik, A., Kanellopoulos, A., Claassen, G., Bloemhof-Ruwaard, J., & van der Vorst, J. (2017). Closing loops in agricultural supply chains using multi-objective optimization: A case study of an industrial mushroom supply chain. *Int J. Prod. Econ*, (183), 409–420.

Baratam, S., Feng, J., Lely, L. v., Oltra, M. R., & Espinoza, M. (2022). Classification framework for autonomous greenhouses.

Bhuvanagiri, S., Pichika, S., Akkur, R., Chaganti, K., Madhusoodhanan, R., & Rusapati, S. (2018). Integrated approach for modeling coastal lagoons: A case for chilka lake, india. *Handbook of Statics*, *39*. https://doi.org/10.1016/bs.host.2018.06.005

Boulard, T., Roy, J.-C., Pouillard, J.-P., Fatnassi, H., & Grisey, A. (2017). Modelling of micrometeorology, canopy transpiration and photosynthesis in a closed greenhouse using computational fluid dynamics. *Biosystems Engineering*, *158*, 110–133. https://doi.org/10.1016/j.biosystemseng.2017.04.001

Carriquiry, M., Dumortier, J., & Elobeid, A. (2022). Food crisis driven by ukraine war could put wild lands to the plough. *609*.

CBS. (2017). *Vooral tomaten in de kas*. https://www.cbs.nl/nl-nl/nieuws/2017/32/vooral-tomaten-in-de-kas (accessed: 22-04-2022)

CBS. (2018). *Schaalvergroting groenteteelt in glastuinbouw*. https://www.cbs.nl/nl-nl/nieuws/2018/16/schaalvergroting-groententeelt-in-glastuinbouw/ (accessed: 16-12-2021)

Costa, A., dos Santos, L., Alem, D., & Santos, R. (2014). Sustainable vegetable crop supply problem with perishable stocks. *Ann. Oper. Res.*, (219), 265–283.

De Koning, A. (1994). Development and dry matter distribution in glasshouse tomato: A quantitative approach.

Deforche, F., & Deforche, O. (2015). *Teelsysteem* (BE1023221A1).

Dieleman, A., De Gelder, A., Eveleens, B., Elings, A., Janse, J., Lagas, P., Qian, T., Steenhuizen, J., & Meinen, E. (2009). Tomaten telen in een geconditioneerde kas: Groei, productie en onderliggende processen. (633).

European Commission. (2022). *Tomatoes statistics*. https://ec.europa.eu/info/food-farming-fisheries/farming/facts-and-figures/markets/overviews/market-observatories/fruit-and-vegetables/tomatoes-statistics_en (accessed: 31-03-2022)

Fernandez, J., & Hernandez, C. (2019). *Practical model-based systems engineering* [ISBN: 9781630815790]. ARTECH HOUSE.

Flores, H., & Villalobos, J. (2018). A modeling framework for the strategic design of local fresh-food systems. *Agric. Syst.*, (161), 1–15.

FromBoer. (2019). *Teelsystemen*. https://www.fromboer.nl/teeltsysteem/ (accessed: 23-12-2021)

Gersdorf, F. (2022). In sommige westlandse kassen blijft het komende winter donker. *Financieel Dagblad*. https://fd.nl/bedrijfsleven/1450937/in-sommige-westlandse-kassen-blijft-het-komende-winter-donker-quj2ca3T2omF. (accessed: 20-10-2021)

Giacomelli, G., Ting, K., & Mears, D. (1994). Design of a single truss tomato production system sttps. *ISHS Acta Horticulturae*, (361). https://doi.org/10.17660/ActaHortic.1994.361.6

Gong, L., Yu, M., Jian, S., Cutsuridis, V., Kollias, S., & Pearson, S. (2021). Studies of evolutionary algorithms for the reduced tomgro model calibration for modelling tomato yields. *Smart Agricultural Technology*, *1*. https://doi.org/10.1016/j.stech.2021.10011

GroenteKennisnet. (n.d.). *Steeds meer teelt op water*. https://www.groenkennisnet.nl/nieuwsitem/Steeds-meer-teelt-op-water-1/ (accessed: 20-12-2021)

Groentennieuws.nl. (2021). *Update uit australische glastuinbouw: "arbeid meest urgente probleem"*. https://www.groentennieuws.nl/article/9368121/update-uit-australische-glastuinbouw-arbeid-meest-urgente-probleem/ (accessed: 10-01-2022)

Grunow, M., Günther, H.-O., & Westinner, R. (2007). Supply optimization for the production of raw sugar. *Int. J. Prod. Econ.*, (110), 224–239. https://doi.org/10.1016/j.ijpe.2007.02.019

Gurobi Optimisation, LLC. (2021). Gurobi Optimizer Reference Manual. https://www.gurobi.com

Hajimirzajan, A., Vahdat, M., Sadegheih, A., Shadkam, E., & El Bilali, H. (2021). An integrated strategic framework for large-scale crop planning: Sustainable climate-smart crop planning and agri-food supply chain management. *Sustainable Production and Consumption*, (26), 709–732. https://doi.org/10.1016/j.spc.2020.12.016

Hemming, S., De Zwart, F., Elings, A., Righini, I., & Petropoulou, A. (2019). Remote control of greenhouse vegetable production with artificial intelligence - greenhouse climate, irrigation and crop production. *Sensors*, *19*(1807). https://doi.org/10.3390/s19081807

Hemming, S., De Zwart, F., Elings, A., Righini, I., & Petropoulou, A. (2020). Cherry tomato production in intelligent greenhouses - sensor and ai for control fo climate irrigation, crop yield, and quality. *Sensors*, *20*(6430). https://doi.org/10.3390/s20226430

Heuvelink, E. (1996). Tomato growth and yield: Quantitative analysis and synthesis.

Heuvelink, E., & Dorais, M. (2005). Crop growth and yield [ISBN: 0-85199-396-6]. *Tomatoes*. Wallingford: CABI Publishing.

Heuvelink, E., Marcelis, L., Eveleens, B., & Kierkels, T. (2021). More insight into cause of autumn drop in tomato production : Light use efficiency 14% lower in autumn than spring. *In Greenhouses*, *10*, 48–49.

Heuvelink, E., Van der Ploeg, A., & Van der Meer, M. (2007). Breeding for a more energy efficient greenhouse tomato: Past and future perspectives. *Euphytica*, *158*, 129–138. https://doi.org/10.1007/s10681-007-9437-z

Huang, K.-L., Yang, C.-L., & C.-M., K. (2020). Plant factory crop scheduling considering volume yield changes and multi-period harvests using lagrangian relaxation. *Biosystems Engineering*, (200), 328–337. https://doi.org/10.1016/j.biosystemseng.2020.10.012

Hussain, I., Knapen, L., Galland, S., Janssens, D., Bellemans, T., Yasar, A., & Wets, G. (2014). Organizational and agent-based automated negotiation model for carpooling. *Procedia Computer Science*, *37*, 396–403.

Jones, J., Allen, L., & Dayan, E. (1991). A dynamic tomato growth and yield model (tomgro).

Lin, D., Wei, R., & Xu, L. (2019). An integrated yield prediction model for greenhouse tomato. *Agronomy*, *9*(973). https://doi.org/10.3390/agronomy9120873

Liuping, W. (2009). Model predictive control system design and implementation using matlab. *Springer Sciences & Business Media*.

Logendra, L., Gianfagna, T., Specca, D., & Janes, H. (2001). Greenhouse tomato limited cluster production systems: Crop management practices affect yield. *5*(36), 893–896.

Logiqs. (2021). *Design the ideal benching system for your crop with just a few clicks*. https://greenhousebenching.logiqsken3d.nl/ (accessed: 28-04-2022)

Matthias, A. (2004). The responsibility gap: Ascribing responsibility for the actions of learning automata. *Ethics Inf Tech*, *6*(3), 175–183. https://doi.org/10.1007/s10676-004-3422-1

Mefteh, W. (2018). Simulation-based design: Overview about related works. *Mathematics and Computers in Simulation*, *15*, 81–97. https://doi.org/10.1016/j.matcom.2018.03.012

Mesoudi, K., Soudani, A., & Bournet, P. (2010). Determination of the inside air temperature of a greenhouse with tomato crop, under hot and arid climates. *Journal of Applied Sciences in Environmental Sanitation*, *5*(2), 117–129.

Mills, A. (2014). Thermal radiation [ISBN: 978-1-292-04248-0]. *Basic heat and mass transfer, second edition*. Essex, UK: Pearson Education Limited.

Mostafavi, S., & Resaei, A. (2019). Energy consumption in greenhouses and selection of an optimized heating system with minimum energy consumption. https://doi.org/10.1002/htj.21540

Motevalli-Taher, F., Paydar, M., & Emami, S. (2020). Wheat sustainable supply chain network design with forecasted demand by simulation. *Comput. Electron. Agric.*, (178). https://doi.org/10.1016/j.compag.2020.105763

Muaz, N., & Hussain, A. (2011). Agent-based computing from multi-agent systems to agent-based models: A visual survey. *Scientometrics*, *89*(2), 479–499. https://doi.org/10.1007/s11192-011-0468-9

Munoz, M., Guzman, J., Sanchez, J., Rodriquez, F., & Torres, M. (2019). Greenhouse models as a service (gmaas) for simulation and control. *IFAC PapersOnline*, *52*(30), 190–195. https://doi.org/10.1016/j.ifacol.2019.12.520

Nederhoff, E., & Marcelis, L. (2010). Calculating light & lighting. *Practical Hydroponics & Greenhouses*, (43).

Nir, A. (2021). *90% succes for robotic tomato harvester*. https://www.hortidaily.com/article/9321841/90-success-for-robotic-tomato-harvester/ (accessed: 16-12-2021)

Nir, A., & Nir, O. (2017). *Robotic devices for individually picking crops* (No. IL269211D0).

NSW Department of Primary Industries. (2020). Improving yields and yield stability in the australian processing tomato industry. *NSW Government*.

Pedersen, J., & Hansson, L. (2021). Consumers' attitude towards locally produced food products.

Piewthongngam, K., Pathumnakul, S., & Setthanan, K. (2009). Application of crop growth simulation and mathematical modeling to supply chain management in the thai sugar industry. *Agric. Syst.*, (102), 58–66. https://doi.org/10.1016/j.agsy.2009.07.002

Radzicki, M., & Taylor, R. (2008). Origin o fsystem dynamics: Jay w. forrester and the history of system dynamics. *U.S. Department of Energy's Introduction to System Dynamics*.

Santoni de Sio, F. (2016). Ethics and self-driving cars: A white paper on responsible innovation in automated driving systems. https://doi.org/10.1007/s10677-017-9780-7

Simske, S. (2019). Sensitivity analysis and big systems engineering. *Elsevier*. https://doi.org/10.1016/B978-0-12-814623-1.00005-8

StatLine, C. (2022). Groenteteelt; oogst en teeltoppervlakte per groentesoot. https://opendata.cbs.nl/statline/#/CBS/nl/dataset/37738/table?fromstatweb. (accessed: 20-10-2021)

Streit, M., & Gehlenborg, N. (2014). Visualizing samples with box plots: Use box plots to illustrate the spread differences of samples. *Nature Methods*, (11).

Tan, B., & Çömden, N. (2012). Agricultural planning of annual plants under demand, maturation, harvest, and yield risk. *Eur. J. Oper. Res.*, (220), 539–549. https://doi.org/10.1016/j.ejor.2012.02.005

Ter Laak Orchids. (2019). *Behind the scenes at ter laak orchids*. https://www.youtube.com/watch?v=TMInsF0EI6A

Testa, R., Di Trapani, A., Sgroi, F., & Tdusisca, S. (2014). Economic sustainability of italian greenhouse cherry tomato. *Sustainability*, *6*, 7967–7981. https://doi.org/10.3390/su6117967

Timmermans, G., Hemming, S., Baeza, E., Van Thoor, E., Schenning, A., & Debije, M. (2019). Ultra-thin glass: The way forward for the low-energy greenhouse? *In Greenhouses*, *8*.

Timmermans, G., Hemming, S., Baeza, E., Van Thoor, E., Schenning, A., & Debije, M. (2020). Advance optical materials for sunlight control in greenhouses. *Advanced Optical Materials*, *8*. https://doi.org/10.1002/adom.202000738

Urbanucci, L. (2018). Limits and potentials of mixed integer linear programming methods for optimization of polygeneration energy systems. *Energy Procedia*, *148*, 1199–1205. https://doi.org/10.1016/j.egypro.2018.08.021

Valera, D., Molina-Aiz, F., Belmonte, L., & López, A. (2017). The greenhouses of almería, spain: Technological analysis and profitability. *Acta Horticulturae*.

Van Henten, E., Van Tuijl, B., Hoogakker, G.-J., Van Der Weerd, M., Hemming, J., Kornet, J., & Bontsema, J. (2006). An autonomous robot for de-leafing cucumber plants grown in a high-

wire cultivation system. *Biosystems Engineering*, *94*(3), 317–323. https://doi.org/10.1016/biosystemseng.2006.03.005

Van Os, E., Blok, C., Van Salm, C., Baeza, E., & Janse, J. (2020). General cultivation manual jordan. https://doi.org/10.18174/507532

Vanthoor, B., De Visser, P., Stanghellini, C., & Van Henten, E. (2011). A model-based greenhouse design method [ISBN: 978-90-8585-919-2].

Verbraeck, A. (2021). *Ngi101x - discrete modelling i*. https://www.youtube.com/watch?v=0eAFmdPQKQk

Verheul, M., Maessen, H., Popanov, M., Pansoyan, A., Kechasov, A., Naseer, D., & Popanov, I. (2022). Artificial top-light is more efficient for tomato production than inter-light. *Scientia Horticulturea*, *291*. https://doi.org/10.1016/j.scienta.2021.110537

Verheul, M., Maessen, H., Popanov, M., & Pansoyan, I., A.and Popanov. (2019). Effects of supplemental light and temperature on summer production of tomato in norway.

Verheul, M., Maessen, H., & Grimstad, S. (2012). Optimizing a year-round cultivation system of tomato under artificial light. *ISHS Acta Horticulturae*, (956). https://doi.org/10.17660/ActaHortic.2012.956.45

Verma, A., & Range, V. (2020). Cosec-rpl: Detection of copycat attacks in rpl based 6lowpans using outlier analysis. *Telecommunication systems*, *75*(1), 43–61. https://doi.org/10.1007/s11235-020-00674-w

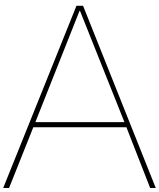
Vermeulen, P. (2016). Kwantitative informatie voor de glastuinbouw 2016 2017: Kengetallen voor groenten - snijbloemen - pot- en perklanten teelten. (GTB-5121).

Vitoriano, B., Ortuño, M., Recio, B., Rubio, F., & Alonso-Ayuso. (2003). Two alternative models for farm management: Discrete versus continuous time horizon. *Eur. J. Oper. Res.*, (144), 613–628.

Wishon, C., Villalobos, J., Mason, N., Flores, H., & Lujan, G. (2015). Use of mip for planning temporary immigrant farm labor force. *Int. J. Prod. Econ.*, (170), 25–33. https://doi.org/10.1016/j.ijpe.2015.09.004

Zeelen, R. (2009). *Method and device for the removal of a leaf from a crop* (US8666552B2).

Zeigler, P. (1991). Object oriented modelling and discrete event simulation. *Advances in Computers*, *33*, 67–114.



Scientific research paper

The scientific research paper begins in two pages.

This page is intentionally left blank

TU Delft report number: 2022.MME.8741
Horticulture Science
TU Delft X Ridder Growing Solutions

Conceptual design of an automation-facilitating tomato greenhouse concept

Author: Floris Bunnik,[1] supervised by: A.N.M. de Koning,[2] W. Oltheten,[2] Y. Pang,[1] & D.L. Schott[1]
[1]*Delft University of Technology, Faculty 3mE, Mekelweg 5, 2626CN, Delft, the Netherlands*
[2]*Ridder Growing Solutions, Honderdland 131, 2676LT Maasdijk, the Netherlands*

ARTICLE INFO

Submission date: 29-12-2022

Keywords:

Greenhouse
Tomato
Simulation-based design
Conceptual design
Optimisation

ABSTRACT

Automation in tomato greenhouses is lacking compared with some other crops. The efficiency and consistency of greenhouse operations can be boosted by implementing automation. Therefore, a new concept that facilitates the robotisation of crop-handling tasks is investigated. This concept combines the existing technology of a movable-gutter system with the cultivation concept of a limited-truss tomato. The combination of these two techniques has not been previously explored. For this paper, a generic conceptual design framework was developed which contains a simulation model and an evaluation protocol. In the evaluation protocol a sensitivity analysis and an optimal dimension analysis is performed. Case studies are conducted to evaluate the simulated performance, to examine the capacity of the transport system and crop-handling tasks and to assess the robustness. These case studies prove that the developed framework is generic and all-encompassing. The framework has proven its worth by eliminating erroneous design choices before significant resources were spent. The new concept is promising, as it facilitates automation and has a competitive simulated production. Therefore, it is advised to move on to the next design phase, which should mainly focus on validating the crop-growth model and the preliminary design of subsystems as the transport system and components as the movable-gutter itself. Furthermore, a cost analysis and economic assessment should be conducted.

1. Introduction

Automation can increase the efficiency and consistency of operations in horticultural greenhouses (Baratam, Feng, Lely, Oltra, M.R., & Espinoza, 2022). In addition, automation will reduce the amount of human labour needed, which could lead to operational cost savings. Although the tomato is the most cultivated crop in greenhouses in the Netherlands (CBS, 2022), the hypothesis that automation is lacking compared with other crops is affirmed in a literature study (Delft University of Technology report number 2022.MME.8597). Due to the jungle-like environment, there are hardly automation solutions operational in tomato greenhouses for crop-handling (De Zeeuw, 2019). Greenhouses for other crops have already achieved higher levels of automation: for example, with lettuce (FromBoer, 2019) and orchids (Ter Laak Orchids, 2019). These systems have in common that crops are transported to a central crop-handling area, where the handling tasks are performed. Orchids are grown in pots, whereas lettuces, like tomatoes, are often grown in gutters. Therefore, this study researched a similar movable-gutter system for greenhouse tomatoes.

One challenge is that, in the conventional cultivation concept, the stems of tomato plants reach lengths of up to 15 [m] in a cultivation cycle of about a year. The last 2.5 [m] of these stems grow vertically, and the remainder grow horizontally. Therefore, the plants are considered unfeasible for transport. In the 1990s and 2000s, a limited-truss concept, in which the plants grow up to 1.1

[m], was researched thoroughly (Giacomelli, Ting, & Mears, 1994; Logendra, Gianfagna, Specca, D.R., & Janes, 2001). The studies claim economic viability due to more successive growth cycles in a year and a higher plant density. This cultivation concept is not commercially operational. However, it could be part of the new movable-gutter concept.

The knowledge gap filled in this study is that there is no knowledge about how to design and evaluate a limited-truss tomato on a movable-gutter system. This research develops a generic conceptual design framework for that new concept. The scope is limited to the conceptual design phase, which includes an area-allocating floorplan and the required transport and machinery or human labour capacities.

This paper begins with an explanation of the new concept in more detail in Section 2, whereas Section 3 discusses the development of the generic conceptual design framework. The implementation is described in Section 4, with the corresponding simulation results in Section 5. Finally, conclusions are drawn in Section 6, and recommendations are made in Section 7.

2. The limited-truss tomato on a movable-gutter

The first important aspect of the new concept to discuss is the limited-truss cultivation concept. In this new concept, only up to three trusses are grown per plant, compared with the 35 trusses in the conventional concept. The cultivation cycle of the limited-truss concept only lasts 2-3 months, and has a higher plant density than the conventional concept. Because of the shorter cycle and higher plant density, there is claimed that a similar or higher production can be achieved with the limited-truss concept, compared to the conventional concept (Logendra, Gianfagna, Specca, D.R., & Janes, 2001). The cultivation cycle with the corresponding handling tasks is presented in Figure 1.

In Figure 1, the growth is divided into five growing stages. Between these stages, crop-handling action is needed. Therefore, the greenhouse is subdivided into five compartments, so there can be continuous year-round production with all growing stages present in the greenhouse. Year-round production is beneficial operational-wise and makes the use of the available space in a greenhouse as efficient as possible due to the different plant densities per growing phase. A top view of how the floorplan of such a concept could look like is presented in Figure 2. In Figure 2, the gutters are indicated with number 123. A front view of the proposed design of these gutters is illustrated in Figure 3.

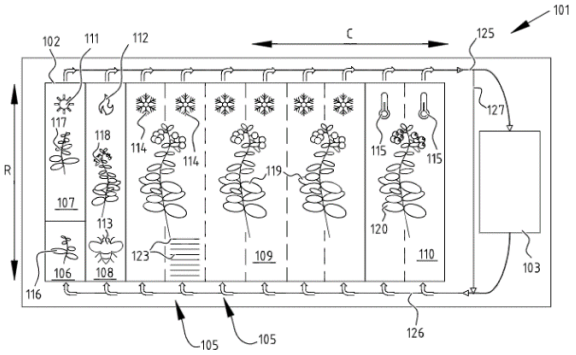


Figure 2: Top view of a tomato greenhouse with the new concept. There will be rows of trays and gutters, indicated with 123, that enter a row at 105. The trays and gutters leave the rows at the top of the figure, to be transported to a central crop-handling area (103). This area is conditioned to the tasks, which facilitates an optimal working environment for both robots and human labour. The underlined numbers discriminate the different growing compartments. Stage 1 will be at 106, 2 at 107, 3 at 108, 4 at 109, and 5 at 110. Furthermore, the climate conditions will differ for each compartment. The required space per stage has large variations, but the figure is not necessarily to scale. ©Ridder Growing Solutions.

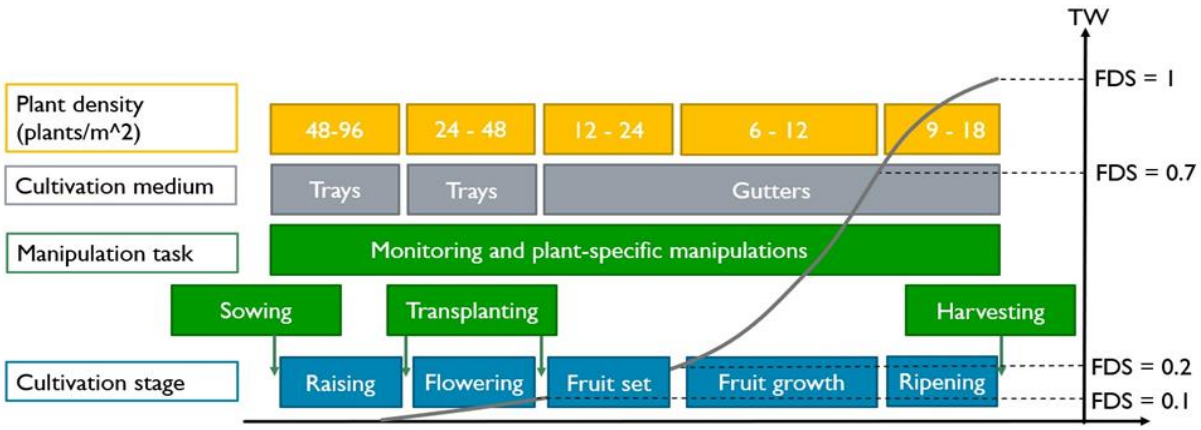


Figure 1: Schematic of the new cultivation concept. TW stands for Truss Weight in grams. FDS means Fruit Development State, a normalised measure for the ripeness of a truss. Days are an indication for cultivating in the Netherlands.

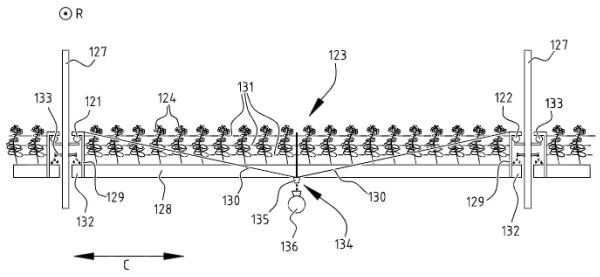


Figure 3: Front view of a gutter design, distance c is about 8 [m], whereas the width of a gutter (inside paper) is tens of centimetres. 121 is the hanging of the gutter, whereas 129 is the watering and nutrition system inflow, and 136 the drainage. ©Ridder Growing Solutions.

Figure 2 also indicates that climate conditions can vary per compartment. This variation is required to ensure a constant truss weight at the end of the cycle for different conditions during cultivation. Assuming there is sufficient water, humidity, nutrients, and CO_2 available, temperature and light are the two factors that mainly influence the growth process (see Figure 4). Light leads to the growth of the tomato (i.e. an increase in truss weight), whereas heat is the main influencing factor for ripeness, defined in the normalised measure of fruit development state (FDS). The light and temperature should be balanced to obtain a constant final product, which can be complex under varying conditions.

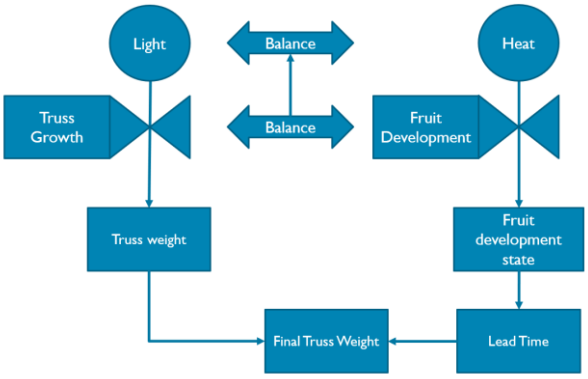


Figure 4: Schematic of the two most influencing factors in the cultivation concept

Figure 1 refers to the varying plant density. However, the important variable is the truss density. Plant density is one of the three variables which determines the truss density. The other two are the number of trusses per plant and gutters per m^2. A multiplication of these three variables determines the truss density. From the onset, the number of plants per gutter and trusses per plant is fixed, whereas the gutter distance can vary. These values at onset need not be constant over the year. Due to option of varying the gutter distance there is an extra degree of freedom in truss density in the new concept compared to the conventional concept with a fixed plant density. This implies there is an extra option to balance the light and heat. More trusses per m^2 imply there is less light per truss. There is a limit, however, when the truss density becomes very small, not all light is intercepted by the plants. The extra degree of freedom in truss density, combined with a year-round production makes the problem complex, but also enables an efficient use of the available area, light and heat.

3. Developed generic design framework

Simulation was used to address the complexities and develop a framework that applies to multiple cases. According to Mefteh (2018) First, there is the choice between a model-based and test-based simulation. Model-based is most applicable in this case, because for test-based simulation, the designer should have complete knowledge about the system. Designers want to simulate before building the real system, and with the model-based simulation, complete knowledge of the system is not required. The simulation is used to gain more knowledge about the system and to explore the different design possibilities.

The simulation model is part of the design framework and divided into three modules. These modules are all needed to generate and simulate the conceptual design and corresponding sowing strategy. The first step is to model the growth of the crops. Then, a conceptual design is generated. Finally, this design is simulated (see Figure 6).

In Figure 6, the climate conditions are the inside day light integral (DLI) of the photosynthetically active radiation (PAR) spectrum (i.e. the relevant spectrum for crop growth) for a geographical location. The cultivar and growth dependent parameters are retrieved from De Koning (1994). The physical design requirements are the expected lead time and truss densities for a specified day of onset, whereas the lot-specific characteristics are the available area and number of crop-handling areas. A

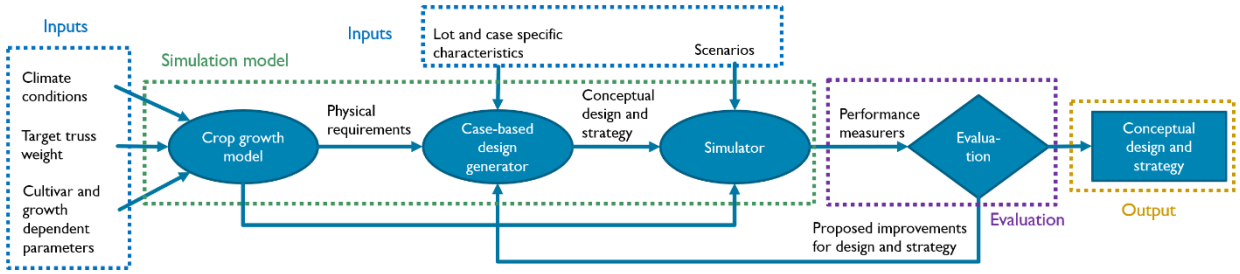


Figure 6: Structure of the developed generic conceptual design framework.

conceptual design (i.e. floorplan, capacity transport system, and capacity crop-handling tasks) is then generated, and a corresponding sowing strategy is calculated. The conceptual design together with the strategy is simulated in the simulator, into which different scenarios (i.e. light intensity, temperature, unit selling price, and lost plants due to illness) can be fed. The performance is assessed using KPIs, which are the occupancy of the greenhouse, the number of lost plants (i.e. due to over-occupation), production per m^2 net growing area, and turnover per m^2 net growing area. Additionally, the occupancy of the transport system and handling-tasks machinery, and the final truss weight are monitored.

In the first module, the crop-growth model of De Koning (1994) was implemented, which is a commercially used crop-growth model for the conventional cultivation concept. For this study, it was assumed the same relationships still hold as for the De Koning model. Moreover, it was assumed the growth behaviour is similar for all growing phases. However, in a follow-up study, this model should be validated for the new cultivation concept, and the relationships should be updated. Although there will probably be differences during the growth process, the hypothesis is that these updated relationships do not significantly influence the performance of the model itself or its outputs.

A mixed-integer liner programming (MILP) optimisation module was developed for the second module. For implementation, the Gurobi package for Python was used. The MILP approach is widely used in complex parallel systems, among which are horticultural systems. The MILP approach is claimed as a quick and state-of-the-art method by Urbanucci (2018). A drawback of this method is that MILP cannot handle the non-linearities in a system. In addition, the entire time horizon must be considered at once, and there is a risk of high dimensionality of the problem. However, there are methods to address these limitations. These risks can be avoided and/or resolved for this specific application. (Banasik, Kanellopoulos, Claassen, Bloemhof-Ruwaard, & Vorst, 2019).

The final step was to simulate the design. The four most widely used simulation approaches are 1) agent-based modelling (ABM), 2) discrete-event simulation (DES), 3) system dynamics modelling (SDM), and 4) model predictive control. Each approach has strengths for the specific application. The DES and ABM approaches seem most applicable to simulate the physical design and gain knowledge about parameter sensitivity. The DES approach tends to be more applicable in the field of operations research at the level of system analysis, whereas ABM is more applicable at a control theory level for complex systems. Therefore, DES is most applicable for the conceptual design phase, whereas ABM is more relevant in the later design phase. The DES approach can simulate how the system states will evolve. The principle is that the events that change the state will happen at discrete points in time. In recent decades, DES has been combined with OOP, as this combination (called OO-DES) is a powerful basis for studying the dynamics and capture the system's complexity (Zeigler, 1991).

4. Implementation of the simulation model

The calculation flow for the crop-growth model is presented in Figure 7. The most important outputs are the optimal truss density and lead time for a specified sowing date. The inputs are light and temperature. The growing stage is defined. The change in FDS over a growing stage is defined as dFDS, whereas light-use efficiency (LUE) is the amount of MJ DLI PAR light needed for 1 kg of fruit growth.

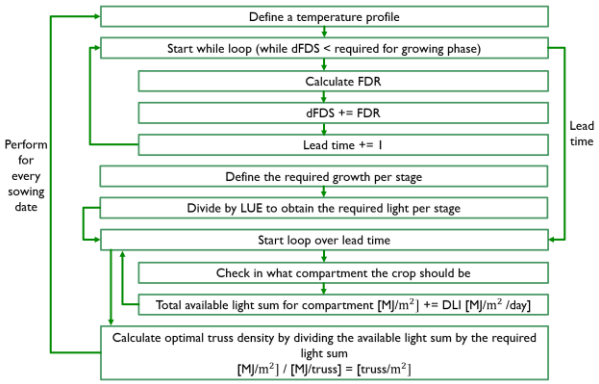


Figure 7: Schematic of the calculation flow for the crop-growth module

The second module, the case-based design generator, needs the outputs of the crop-growth model as input. Then, together with the lot-specific characteristics, this module was used to calculate the area allocation and sowing strategy. Based on this the required capacity of the transport system and the crop-handling system was calculated. This flow is presented in Figure 8.

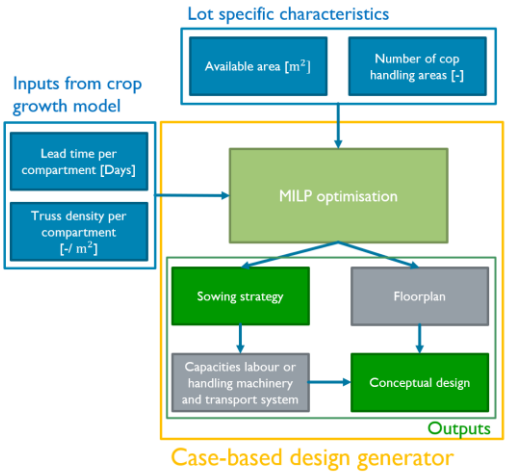


Figure 8: Schematic of the inputs and outputs for the case-based design generator

The simulator can be represented in multiple ways. The first option is using an activity diagram (Figure 9). This Figure presents a simplified overview of different activities but identifies the structure of the greenhouse operations. Next, the structure of an arbitrarily chosen piece of machinery, the sowing machine, a greenhouse compartment and a shuttle are presented in Figures 10, 11, 12, and 13, respectively.

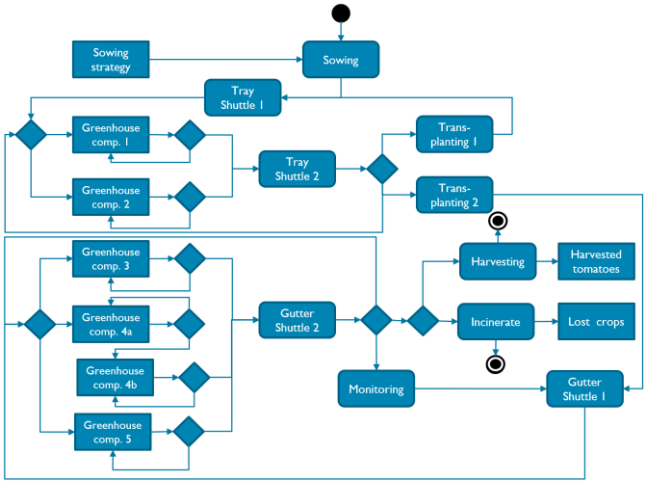


Figure 9: Activity diagram of the new concept (simplified)

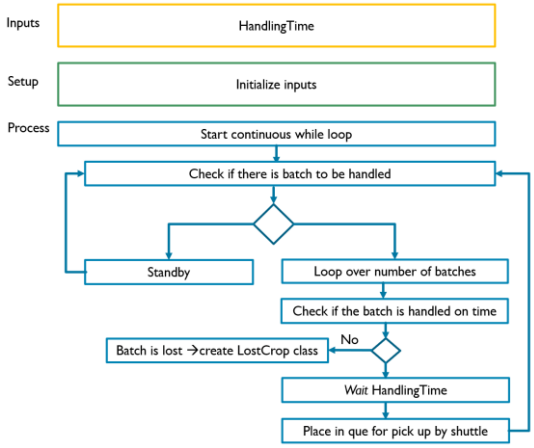


Figure 10: Basic structure implement for a handling machine (Transplant 1, Transplant 2, or monitoring)

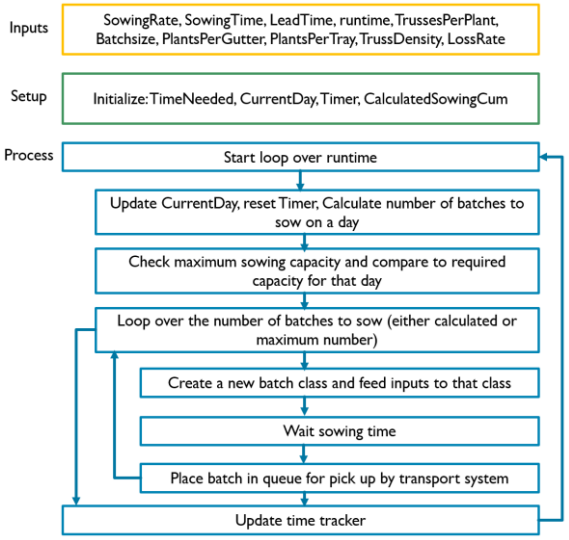


Figure 11: Definition of the sowing machine

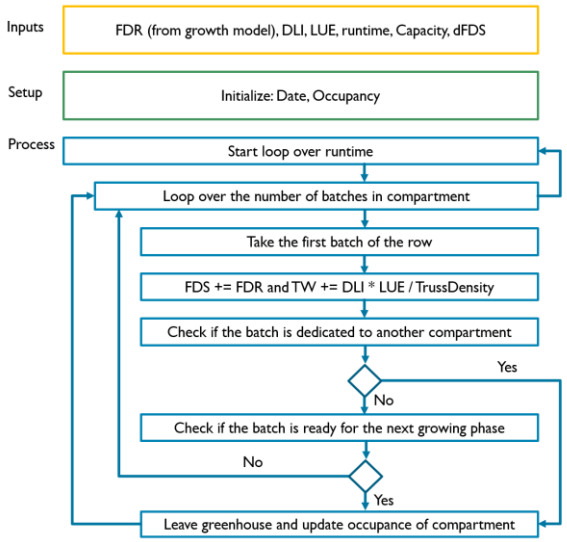


Figure 12: Definition of a greenhouse compartment

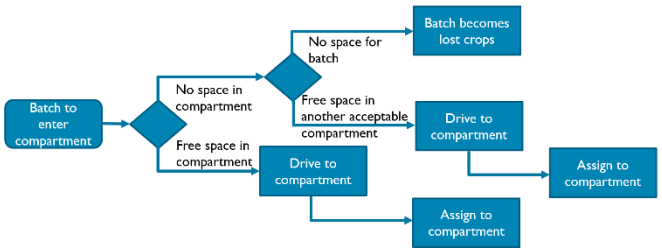


Figure 13: Decision tree for a shuttle that enters the greenhouse

5. Results

The evaluation protocol follows the scheme in Figure 14. First, a sensitivity analysis was performed. The sensitivities of the turnover regarding temperature, light intensity, LUE, αFDR (i.e. a cultivar-dependent parameter), and the target truss weight were assessed. The main result is that the system is proportionally sensitive to light intensity and LUE, as expected. The same results were found for light intensity and LUE, as both factors are multiplied in the calculation flow. A changing αFDR only causes disturbances. This is an empirically found parameter by De Koning (1994), which should be in line with the system and other cultivar-dependent parameters. The system is insensitive to temperature and target truss weight. These findings were expected, as a change in these values does not cause more or fewer kilograms of tomatoes to be produced. If these variables change, so do lead time and truss density, but the weights produced are similar. It is important that the 24-hour mean temperature remains above its feasible lower boundary of 17 degrees Celsius.

The second step involved assessing the dimensions at which the system performs at its maximum turnover. A new compartment can only start in a new row. Therefore, the area allocation is in discrete steps. The more rows there are, the more possibilities for area allocation. Generally, more options lead to a better choice in this type of model. More choices can be

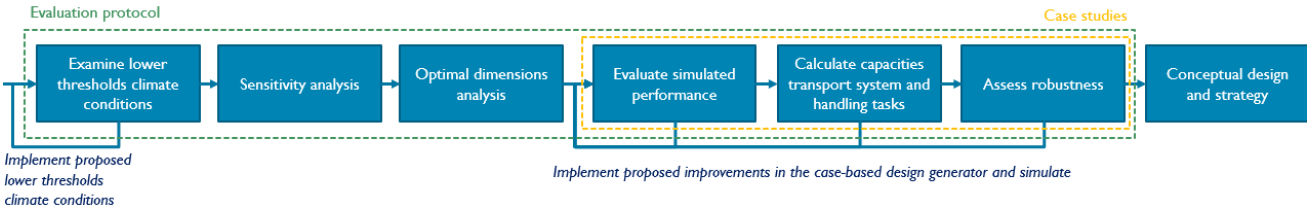


Figure 14: Protocol applied for evaluation of the conceptual design and improvements of the simulated performance.

obtained by having a larger greenhouse or changing the length:width ratio. In this context, for example, a length:width ratio of 1:2 means a 100 x 200 [m] greenhouse that has rows with a length of 100 [m] minus the width of the transport system. For a 1:2 length:width ratio, different options were simulated. The results are presented in Figure 15. Subsequently, for a 3-ha greenhouse, the length:width ratios were changed (see Figure 16).

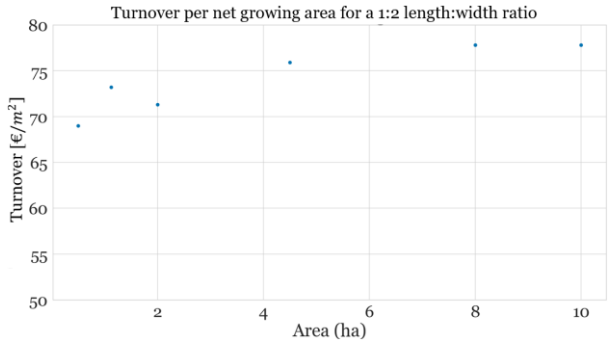


Figure 15: The data points for different length:width greenhouse sizes for a 1:2 length:width ratio.

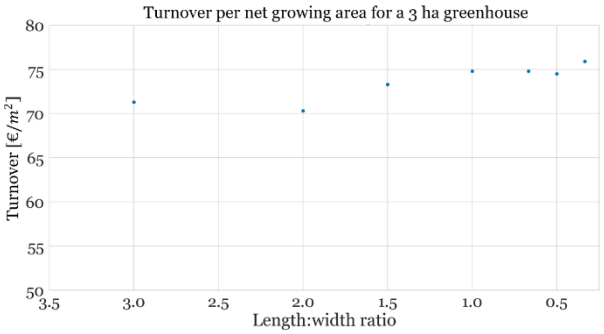


Figure 16: The data points for a different length:width ratios for a 3 ha greenhouse.

Based on the results in these last two figures, a bigger greenhouse and a smaller length:width ratio generally lead to higher turnover. However, from 4.5 ha and larger for a 1:2 ratio for a given input set, the turnover is considered to have converged to its maximum (based on the outputs of Figure 15). Next, a case of a 150 x 300 [m] lot, with typical climate conditions for Rotterdam, the Netherlands (Vermeulen, 2016) was simulated. First, the number of crop-handling areas was reviewed. The total travel time over the transport system was almost twice as low for two crop-handling areas at both sides than for a single crop-handling area. Having two crop-handling areas is not feasible for every lot, but for this one, it was considered workable. The generated floorplan is presented in Figure 17.

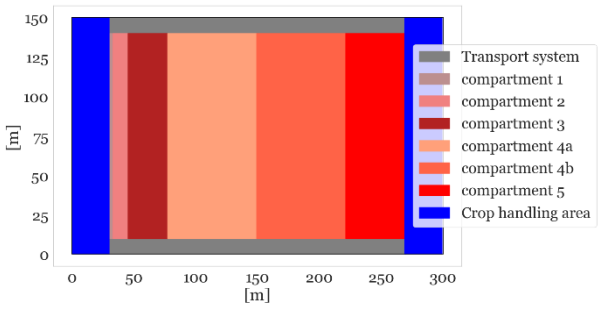


Figure 17: Generated floorplan with two crop handling areas

For this floorplan, the transport capacity and the capacity of crop-handling tasks were examined. The capacity of the system is minimised, as a system with a smaller capacity is often cheaper. At the same time the constraint that there may not form a significant bottleneck must be respected. A bottleneck is defined as that there are plants that cannot be picked up within a threshold of 24 hours and a threshold of six hours after pickup before crop-handling. In this specific case, only one transport lane was needed. Because it appears only one transport lane is needed to fulfil the minimum capacity constraint, in a future design phase, it is expected the required width of the transport system will be less than stated in Figure 16. Moreover, the model assumes 24/7 operations. Future studies could include other operational times and thresholds in the model.

Next, the three steps indicated with ‘case studies’ in Figure 14, are performed for weather data at three different geographical locations on earth. These locations are Rotterdam, Darwin and Melbourne. All these locations have very different climate conditions. Applying the evaluation protocol give similar final conceptual designs for all case studies, but the performance differs. The design depends on growth specific relationships, among which the growth stage definitions. In the sensitivity analysis, the turnover was about proportional sensitive to light. Therefore, the locations with higher light intensities over the year have a higher turnover. The capacities of the transport system and crop-handling system are reviewed, and the

Table 1: Output KPIs for case based on real weather data

Case study	Location	occupancy rate greenhouse	Lost plants	Production kg/m^2	Turnover €$/m^2$
1	Rotterdam, the Netherlands,	91.0 ±0.4	6.2 ±2.1	71.3 ±1.1	85.6 ±1.4
2	Darwin, Australia,	90.5 ±0.6	5.5 ±0.5	123.1 ±2.7	147.8 ±3.2
3	Melbourne, Australia	90.8 ±0.7	5.5 ±1.5	86.8 ±0.9	104.1 ±1.1

robustness is assessed. The sowing strategy and design are based on a 10-year mean, whereas the robustness is assessed by simulating for that design and strategy for weather data for 5 individual years. The simulated output KPIs for the case studies are in Table 1.

Table 1 indicates the new concept is promising as the production is competitive for all three cases (see for reference values Heuvelink & Dorais (2005), NSW Department of Primary Industries (2020), and Verheul, et al. (2019)). The occupancy rate of the greenhouse was high, because this simulated occupancy was for an entire year, higher than for the conventional concept. A 100% occupancy rate cannot be achieved due to the variable climate conditions over the year. The number of lost plants was due to overoccupancy is a trade-off with the other three KPIs. If there were aimed for less plants lost, the other KPIs would have been (slightly) lower as well. The turnover is a multiplication of the production with a price curve. This is kept constant due to the different geographic locations, but could also have been based on, for example, the EU five-year average tomato-price curve, with higher prices during winter and lower prices during summer. However, the results in Table 1 are competitive with the conventional concept. The main focus of the research is not an economic assessment, but the development of the generic conceptual design framework. Conducting the case studies proved that the developed design framework is applicable to multiple case studies with very different climate conditions. Therefore the design framework is generic.

6. Conclusion

The worth of the developed framework was proven in this study by demonstrating 1) that it can generate conceptual designs in a quick and smart manner, and 2) that multiple designs can be simulated, which makes it capable of eliminating design errors before significant resources are used. A first good example of proving its' worth is choosing the approach to employ an existing growth model. This choice involves whether to calculate the optimal truss density for a given temperature, or vice versa. The framework revealed the new approach of calculating truss density based on temperature led to a more constant truss weight at harvest than for the approach of calculating a temperature based on truss density. Another example of the power of using a model is that inspecting the figures regarding the occupancy of greenhouse compartments enables optimisation of the area allocation before using significant resources for designing. Simplified parameters remain in the current version of the simulation model. For example, the LUE and PAR ratio are constants, whereas they slightly vary (in the order of 5%) over the year. The developed simulation model can reveal how releasing these assumptions affects the design and performance of the concept and is, therefore, a major help in the design process. The framework is developed to apply to (almost) every case study, meaning it is generic. Therefore, the first step in filling the scientific knowledge gap – regarding the generation and evaluation of designs for the new tomato-greenhouse concept – is achieved.

The designs generated by the framework, thus employing the evaluation protocol, for different cases show a high production compared to the conventional system. In addition to this competitive production, the new concept contains other elements which makes it interesting. The concept facilitates automation, but even if no automation is implemented the new concept will gain the operational efficiency. In addition, the multiple successive cycles per year reduce the severeness of illnesses. If a plant becomes ill, there is less loss in a short cultivation cycle than for a long cultivation cycle. An advantage of the new concept is also the year-round production, which also enables more continuous operations. Moreover, it makes very efficient use of resources, which makes the new concept more sustainable than the conventional one.

7. Recommendations

The first major assumption is that the existing crop-growth model of De Koning (1994) still holds for the new cultivation process. Future research should determine whether this assumption is correct. Additionally, the exact relationships for fruit growth and development with the climate conditions for different growing stages should be investigated. The definition of growing stages should also be substantiated in greater depth. Ideally, a digital twin should be developed and

implemented in the simulation model of this study to simulate the growth.

Another necessary step is the physical design of individual subsystems as the transport system or components as the movable-gutter itself. This study developed a framework for a conceptual of the system design, but studies should research the designs of individual subsystems and components.

The simulation model developed in this study can be further detailed. The simplified parts, assuming them to be constant (e.g. LUE or PAR Ratio) or linear (e.g. crop development), can be replaced by time-series or fundamental exponential equations. Implementing these new parts would make the model more complex. However, the simplifications are considered to have had no substantial effect on the conceptual design or performance. Another desirable aspect for the model is quantifying all kinds of costs, for example, heating and lighting costs and the cost of a plant. By implementing all these costs, decision can be made on profit maximiastion.

Furthermore, validation and real-world experiments should be conducted. Since the system researched in this study does not exist, a thorough validation was impossible, let alone real-world experiments. The next step would be to develop a small-scale test setup to undertake the validation and experiments. However, it will be costly to build such a setup; therefore, it would be beneficial to develop the model in greater detail first.

Finally, regarding the viability further research should be conducted. This is in a broad sense. First, new automation solutions for individual that can be implemented in the new concept should be reviewed. Second, there could be examined till what extent the new concept is more sustainable than the conventional concept. The simulated performance of the new concept is promising, but it should be researched what the place in the fast-changing greenhouse market could be.

References

Banasik, A., Kanellopoulos, A., Claassen, G., Bloemhof-Ruwaard, J., & Vorst, J. v. (2019). *Accounting for uncertainty in eco-efficient agri-food supply chains: A case study for mushroom production planning.* J. Clean. Prod.

Baratam, S., Feng, J., Lely, L. v., Oltra, M.R., & Espinoza, M. (2022). *Classification Framework for Autonomous Greenhouses.* AgTech Institute.

CBS. (2022). *Groenteteelt; oogst en teeltoppervlakte per groentesoot*. Opgeroepen op 10 20, 2022, van CBS Statline: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/37738/table?fromstatweb

De Koning, A.N.M. (1994). *Development and dray matter distribution in glasshouse tomato: A quantitative approach.* Wageningen University of Agriculture.

De Zeeuw, R. (2019). Insection in de kas zijn de klos door jagende drone. *Algemeen Dagblad*. Opgeroepen op 12 28, 2021, van https://www.ad.nl/westland/insecten-in-kas-zijn-de-klos-door-jagende-drone~aaca12bc/

FromBoer. (2019). Teeltsystemen. Opgeroepen op 12 23, 2021, van https://www.fromboer.nl/teeltsysteem/

Giacomelli, G., Ting, K., & Mears, D. (1994). *Design of a single truss tomato production system STTPS.* ISHS Acta Horticulturae. doi:10.17660/ActaHortic.1994.361.6

Heuvelink, E., & Dorais, M. (2005). *Tomatoes* (Vol. Crop Growth and yield). Wallingford: CABI Publishing.

Logendra, L., Gianfagna, T., Specca, D.R., & Janes, H. (2001). *Greenhouse Tomato Limited Cluster Production Systems: Crop Management Practices Affect Yield.* Hortiscience.

Mefteh, W. (2018). *Simulation-Based Design: Overview about related works.* Mathematics and Computers in Simulation. doi:10.1016/j.matcom.2018.03.012

NSW Department of Primary Industries. (2020). *Improving yields and yield stability in Australian processing tomato industry*. NSW Government.

Smutkupt, U., & Wimonkasame, S. (2009). *Plant Layout Design with Simulation.* Hong Kong: IMECS: Proceedings of the International Multi-Conference of Engineers and Computer Scientists.

Ter Laak Orchids. (2019). Behind the scenes at Ter Laak Orchids. Opgehaald van https://www.youtube.com/watch?v=TMInsF0EI6A

Urbanucci, L. (2018). *Limits and potentials of Mixed Integer LInear Programming methods for optimization of polygeneration energy systems.* Energy Procedia. doi:10.1016/j.egypro.2018.08.021

Verbreack, A. (2021). *NGI101x - Discrete Modelling I.* Delft University of Technology. Opgehaald van https://www.youtube.com/watch?v=0eAFmdPQKQk

Verheul, M., Maessen, H., Popanov, M., Pansoyan, A., Kechasov, A., Naseer, D., & Popanov, I. (2019). *Effect of supplemental light and temeprature on summer production of tomato in Norway.* Norwegian Institute of Bioecnomy Research.

Vermeulen, P. (2016). *Kwantitative Informatio voor de Glastuinbouw 2016 2017: Kengetallen voor Groenten - Snijbloemen - Pot - en perklaplanten teelten.* Wageningen University \& Research: Businiess Unit Glastuinbouw.

Zeigler, P. (1991). *Object Oriented Modelling and Discrete Event Simulation.* Advances in Computers.

B

Pseudo code

In this appendix, the full code is represented as pseudo code. This is fully in line with the full code given in the next Appendix. Here the code is given in one file, but for different parts and functions, this could also be done in multiple files. The psuedo code is given in a python code text format, to be able to indicate indents.

```
# Set inputs for scenario
Temperature strategy: choice either a constant value, a curve with typical
    numbers, and possibly add noise or a constant term
Light intensity: Choose a typical number curve, add noise, or base on a given
    temperature
Cultivar: Only implemented choice is a single cultivar (possibly extend in a
    follow-up study)
LossRate: Choose a percentage of lost plants due to illness
Price curve: Set a constant price, EU-price curve, or block curve
Orientation: Choose either for relatively short or long rows
Number of crop handling areas: Choose either for 1 or 2 crop handling areas
Year: if one wants to simulate for weather data of a specific year indicate here
Base curve: if one wants to deviate from typical numbers as base to a 10-year
    average, indicate here
```

```
# import packages
import numpy
import pandas
import matplotlib.pyplot
import matplotlib.patches
import seaborn
import scipy
import gurobipy
import salabim
import sklearn.preprocessing
```

```
# Make definitions on growing phases and greenhouse conditions
Define fractional fruit development per growing stage
Define fruit growth in grams per stage (sum equals target truss weight)
   Note that for both these terms, a very small number is implemented, while in
       reality this is zero. If new relations ships are found this could be updated
Define length and width of greenhouse in [m]
Define upper and lower bound inside DLI [MJ/m^2/day]
Define runtime: default run for 2 years -> take 1 representative year for
    evaluation (due to start and stop phase)
Define batchsize: default 500 plants
```

```
Define plants per tray for stage 1 and 2: this will probably be constant over the
    year, might be overwritten later
Define transmission rate as average for greenhouse as whole: default 0.80
Define PAR ratio (fraction par spectrum in light beam): default 0.50 might
    slightly vary by varying refraction indices
Define LUE (Light Use Efficiency): default 46; might slightly vary over the year
    and per cultivar
Define number of compartments: default 6 (compartment 4 is split into 4a and 4b)
if cultivar == default cultivar:
   Define cultivar dependent parameters (M_acc, B_acc, aFDR, dm_pr, source_sink,
       TargetWeightPerFruit, FruitsPerTruss, FDMC0, FDMC1, FDMC, WeeklyFruitGrowth)
```

```
# Initialize and reformulate
Define Available area = length * width
Define crop handling area = 20% of total area
Calculate width of crop handling area
Calculate length of crop handling area
Define transport system width
Define width of a row of gutters
Define length of a gutter
Define width of a row of trays
Define length of a tray
if orientation == 'short rows':
   Calculate length of a row for orientation
   Calculate available width for rows for orientation
   Calculate area for transport system for orientation
   Calculate effective area for crop growth for orientation
if orientation == 'long rows':
   Calculate length of a row for orientation
   Calculate available width for rows for orientation
   Calculate area for transport system for orientation
   Calculate effective area for crop growth for orientation
Create an array with discrete steps for areas for rows of trays
Create an array with discrete steps for areas for rows of gutters
Merge these two arrays into one matrix
```

```
if base curve == '10-year average'
   Load KNMI Weather data
   Create data frames with information of the last 10 years
   Select and store information for 24-h average temperature and Global radiation
   For temperature:
      Divide by 10 to convert to Celsius
      Add 3 Celsius as conversion from outside to inside (might add a more
          complex relation with light here)
      Calculate 10-year mean
      Duplicate mean to have a data frame for simulation for 2 years
   For Light:
      Divide by 100 to convert to MJ/m^2
      Multiply by transmission rate and PAR ratio
      Calculate 10-year mean
      Duplicate mean to obatin a data frame for simulation for 2 years
   plot obatined data
```

```
if base curve == 'typical numbers'
   Define an array with monthly average typical numbers
   Define days per month
   Interpolate to obtain a daily value (use scipy)
   Multiply with transmission rate and PAR ratio
```

```
   divide by 100 to convert to MJ/m^2
```

```
Define DLI curve calculation function:
   Check what base curve is
   Take relevant base curve for light
   Loop over runtime
      If light is below threshold -> Add artificial light
      If light is above threshold -> filter by screens
      if light is within thresholds -> keep light curve
   if noise added == True
      Set random seed
      Use np.random.normal to generate noise
      Initialize arrays for noise curves
      Loop over runtime
         Add noise per datapoint
         Check if the thresholds are still respected
            if yes -> continue
            if not -> add/remove light or heat
   Return light curves
Calculate light curve with function
Plot light curves
```

```
Define Temperature curve calculation curve
   if scaled with typical numbers
      Use Sklearn MinMax scaler to scale with light within min and max temperature
   if scale with typical numbers + term
      Add term to scaled version
   if noise added
      Add noise to scaled curve by using same noise as for light
   if constant value
      Create an array with a constant value
   if 10-year mean base curve is chosen
      Load information from created data frame for temperature based on KNMI data
   return Temperature curve
Calculate temperature curve with function
plot temperature curve
```

```
# Calculate total lead time
# If the simulation is used for design, there is a preference to base it on
    typical numbers
# Else, this can be based on any input
Define max lead time: default = 120
Initialize lead time array
loop over runtime - max lead time (for i over range runtime)
   Initialize FDR (array) and FDS (integer)
   loop of max lead time (for j over range max lead time)
      Calculate current day (i + j)
      Calculate FDR (aFDR + log(T/20) * Empirical constant
      Update FDS (FDS += FDR)
      if FDS > 1
         Define Lead time for day of onset i = j
         break

# Calculate lead time per compartment
Set lead time compartment 1 is constant: default = 6 days
Define array with lead time for date of onset (constant value)

Initialize array for lead time for compartment 2
```

```
loop over runtime - max lead time (for i over range runtime)
   Initialize FDR (array) and FDS (integer)
   loop of max lead time (for j over range max lead time)
      Calculate current day (i + j + lead time previous compartments)
      Calculate FDR (aFDR + log(T/20) * Empirical constant
      Update FDS (FDS += FDR)
      if FDS > delta FDS for compartment
         Define Lead time for day of onset i = j + lead time previous
             compartments for date of onset i
         break
Do some protocol for other compartments as well
Now the lead time per compartment for a specified day of onset is calculated
Note that this lead time is calculated on beforehand and can differ for real
    conditions
Plot the lead times
```

```
# Calculate the light sums and plant density
Initialize array for needed light per plant per compartment
loop over number of compartments
   Calculate required light intensity (= growth per stage / LUE)

Initialize arrays for total available light per compartment
Loop over runtime - max lead time (for i over range runtime)
   Initialize light Sum (= 0)
   Loop over max lead time (for j over range max lead time)
      Define current lead time (= j)
      Calculate current day (day = i + j)
      Update light sum (Light sum += DLI)
      if current lead time == lead time compartment 1 for date of onset i
         Available light compartment 1 = Light Sum
      if current lead time == lead time compartment 1+2 for date of onset i
         Available light compartment 2 = Light Sum - Available light compartment 1
      if current lead time == lead time compartment 1+2+3 for date of onset i
         Available light compartment 3 = Light Sum - Available light compartment
             1 + 2
      ..
      ..
      if current lead time == lead time compartment 1+2+3+4a+4b+5 for date of
          onset i
         Available light compartment 5 = Light Sum - Available light compartment
             1+2+3+4a+4b
      break

Calculate continuous truss density over lead time for specified day of onset:
   Truss density compartment = Available light sum compartment / Required light
       intensity compartment
   #-> By applying this procedure the truss density is constant over the lead
       time for a compartment #for a batch that enters that compartment, while the
       density can vary over different batches in a #compartment. Moreover, this
       procedure is not that sensitive to daily fluctuations in light, is #uses
       the average over a lead time

Define function find_nearest
   inputs:  array with possible options
            value where the nearest should be found
   use python formulation: idx = (np.abs(array - value)).argmin()
   return array [idx] (nearest value)

Create an array of possible truss densities
Calculate closest discrete truss density by using fined_nearest function
```

```
Initialize array for number of trusses, plants per gutter and gutters per m2
Loop over runtime
   Find closest truss density for compartment 4 (largest compartment -> greatest
       influence on performance)
   Find the corresponding number of trusses per plant for that truss density
      -> If there are more options, choose the largest number of trusses
   Find the corresponding number of plants per gutter
      -> If there are more options, choose the largest number of trusses
   Fix the number of trusses per plant and plant per gutters over compartment 3,
       4a, 4b and 5 -> there is no transplant step
   Find the nearest from the reduced list of options (only free variable is
       gutters per m2)
   Fix the number of trusses per plant for compartment 1 and 2
   Find the nearest from the reduced list of options (free variables are plants
       per gutter and gutters per m2)

# Comments: one could implement a separate list for compartment 1 and 2 as well
# One could relax the constraint of a fixed number of plants per gutters over
    compartment 3 to 5
# There is a general list of truss densities, this could be reduced to smaller
    lists of options per compartment. For building the greenhouse it is better if
    there are less options, as per compartment the chosen options remain within a
    range.

Plot both the found continuous as the discrete values for truss densities
```

```
Define function GrowthModel
# this model might be extended with new found (validated) relationships
   Inputs: start day, runtime, max lead time, temperature, truss density, delta
       FDS per growing phasE)
   Intialize FDR, FDR, Fresh Fruit Weight (FFW), Absolute Daily Growth (ADG)

   loop over max lead time (for i over range max lead time)
      Select the growing phase a plant is in
      Calculate FDR for day i (aFDR + log(T/20) + Empirical constant)
      Calculate FDS (FDS[i+1] = FDS[i] + FDR[i])
      Calculate ADG (DLI * LUE / truss density)
      Calculate FFW (FFW[i+1] = FFW[i] + ADG[i])

      if FDS > 1
         lead time = i
         break
   return FFW, lead time, FDR, FDS

Define GrowthOverRun
   Make a dictionary for every date of onset for FFW, FDR, and FDS
   return Dictionary with information for expected growth for every date of onset
```

```
Define function PriceCurve
   if price curve == "EU-price"
      Load monthly prices and interpolate
   if price curve == "EU-block"
      create a block curve with prices similar to EU-curve
   if price curve == "constant"
      create a flat constant price curve
   return price curve
Calculate price curve based on strategy
plot curve
```

```
Define function Optimisation model
   # Sets, parameters, variables
   Define set K for number of compartments
   Define set I for runtime
   Load the lead time as calculated for typical numbers or 10-year mean
   Load price curve
   Initialize model
   Initialize variable for sowing rate
   Initialize variable for area allocation
   Initialize variables for discrete options in area allocation

   # objective function
   Set objective: Maximise (Sowing Rate * Corresponding expected final truss
       weight * price per kg)
   update model

   # Add constraints
   1) Allocated area may not be bigger than available effective area
   2) Sowing rate must respect available effective area
      -> Sowing Rate * lead time <= Area * Truss density / Trusses per plant
         -> Sowing rate is defined as an amount of sowed plants
   3) Non-negativity constraints (sowing rate and area may not be negative)
   4) Discrete value for area allocation constraint
      a) Assign discrete possible values to compartments areas
      b) Introduce binary help variable per compartment as array as long as
          discrete area options
      c) State that the sum of that help variable array equals 1 (1 area is
          chosen)
      d) Connect that binary variable to a value of the discrete list
   5) Constrain the start up and finish strategy (here with an empty greenhouse)
   6) State that after the representative year, there has not to be sowed anymore
       (a representative year is here defined from day 135 to day 500 out of 730

   #Perform optimisation; Use gurobi
   Set cut off condition for gain in performance (default 0)
   Set cut off time limit for optimisation procedure (default 30 seconds)
   Preform optimisation

   Save Sowing strategy
   Enhance sowing strategy (default 5%) as there is not accounted for buffering
       capacity
   Save Area allocation -> This is also the place where the area (design) is
       fixed for simulating other scenarios
```

```
# If one has no gurobi software license, an estimation can be made as well be
    using the next 'manual' procedure

Calculate median lead time per compartment
Calculate median m^2 per truss (1/truss density) per compartment
Multiply lead time * m^2 per truss per compartment
Sum over all compartments (=SUM WT)
Calculate 'area weight' = lead time * m^2 per truss / SUM WT * Effective area for
    growth

apply find_nearest function for available areas

Check if due to the offsets the total area is not exceeding the available area
   if yes -> remove a row from the area with the largest offset in the
       find_nearest function
```

```
Calculate sowing rate by Effective area for growth / (m^2 per truss for sowing
    date * lead time for sowing date)
Improve sowing rate by multiplying the differential in (m^2 per truss * lead
    time) by a factor 20
Enhance sowing rate by 5% to make efficient use of the buffering capacities that
    are not considered yet.
```

```
Define function Floorplan
   Here, the area allocation and orientation is converted to a plot of a floorplan
   return plot of floorplan
```

```
# Calculate required capacities etc.
Convert sowing rate to production rate, based on expected lead time
Calculate expected harvested tomatoes, based on expected climate conditions
Calculate expected occupancy by taking the difference in sowed and produced

Make initialization for transport, transport medium and handling tasks

Calculate minimum, maximum and mean capacities transport system and handling
    tasks, based on defined characteristics of trays and gutters, and based on
    expected lead times

Implement transport capacities in a throughput number
Calculate the time a handling takes

Define an unload and load time of the transport medium (default = 1 second)

Define function TransportTime
   Input: StartLocation, TargeLocation
   Define max acceleration (default = 0.5 m/s^2)
   Define max velocity (default = 3 m/s)
   Check whether there are 1 or 2 crop handling areas
      Check Start and Target Location
         Take transport distance
   Calculate time needed for acceleration and deceleration
   Convert that to a distance
   Calculate the distance for max velocity
   Calculate time for max velocity
   Calculate drive time (max velocity time + 2 * acceleration time)
   Convert from seconds to days (default time unit of simulation)

Load drive times for fixed compartments
# one can also make this function more complex and retrieve 'real time' drive
    times per transport action. Now there is implemented that there is a fixed
    distance for every compartment. Furthermore, by implementing the current way,
    a shuttle that takes one tray or gutter is implement. Potentially this could
    also be another transport mode. That mode should be implemented in this
    function.
```

```
if a there is chosen to simulate for data based on KNMI for a specific year
   Check what the period is
      Take data from dataframe created at the beginning of the script
         Check whether the boundaries on light intensity and temperature are
             respected
            Adjust by extra heat, artificial light or screening if needed

Update calculated FDR for the new values
```

```
# FDR can either be calculated real time, or feed as known value to the
    simulation, this gives the same results. Only in an operational phase, this
    should be done real time, as the values are not known on beforehand

Update Sowing Strategy, by applying the next procedure
   Calculate sowing rate by Effective area for growth / (m^2 per truss for sowing
       date * lead time for sowing date)
   Improve sowing rate by multiplying the differential in (m^2 per truss * lead
       time) by a factor 20
   Enhance sowing rate by 5% to make efficient use of the buffering capacities
       that are not considered yet.
   # The reason to do this, is that the 10-year mean differs systematically from
       the typical numbers. The design should remain the same as for the typical
       numbers!
```

```
# Define Classes for simulation

class TBatch = SimElement #passive class that consist trusses, plants and
    cultivation mediums
   def setup
      Inputs            Batchsize, LossRate
      Initialize as integer: FDS, TrussWeight, NumberOfTrussesPerPlant,
          NumberOfGutters, NumberOfTrays1, NumberOfTrays2, Timer, STartDate,
          CurrentTime, TrussDensity, TimeInWrongCompartment,
          TransportTimeCumulative, TransportDistanceCumulative,
          NumberOfTransportActionsCumulative
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      repeat
         Day = InterArrivalTime.sample
         Wait Day
         Timer += Day

class THarvestedTruss = SimElement #passive class that consist all harvested
    trusses
   def setup
      inputs:           None
      Initialize as integer: TrussWeight, LeadTime, Revenue,
          TransportTimeCumulative, TransportDistanceCumulative,
          NumberOfTransportActionsCumulative, TrussesPerPlant
   def process
      None

class TLostCrop = SimElement #passive class that consist all lost crops
   def setup
      inputs: None
      Initialize as integer: TimeLost, ReasonOfLoss, TransportTimeCumulative,
          TransportDistanceCumulative, NumberOfTransportActionsCumulative,
          TrussesPerPlant
   def process
      None

class TSowingGenerator = SimElement # Active class that generates the batches
   def setup
      inputs: SowingRate, TimePerSowingAction, LeadTime, runtime,
          NumberOfTrussesPerPlant, Batchsize, PlantsPerGutter, PlantsPerTray1,
          PlantsPerTray2, TrussDensity, LossRate
      Initialize as integer: TimeNedded, CurrentDay, Timer,
          CalculatedSowingCumulative, ActualSowed
      Initialize as array of length runtime: OperationalTime
```

```
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      Day = InterArrivalTime.sample
      loop over runtime
         CurrentDay += Day
         Timer = 0
         CalculatedSowingCumulative += SowinRate(day)
         SowedToday = CalculatedSowingCumulative - sum(Sowed till now)
         NumberOfPlants = Batchsize
         NumberOfBatches = round down (SowedToday / NumberOfPlants)
         SowingTimeForBatch = SowingTimePerTray * NumberOfPlants / PlantsPerTray
         Sowed += NumberOfBatches * NumberOfPlants
         if NumberOfBatches > 0:
            MaximumNumberOfBatches = integer(Batches/SowingTimeForBatch)
            if MaximumNumberOfBatchs > NumberOfBatches:
               NumberOfBatches = MaximumNumberOfBatches
               give warning
            Loop over NumberOfBatches
               Batch = Create TBatch(BatchSize, LossRate)
               Assign all attributes to TBatch
               Wait SowingTime * NumberOfTrays1
               OperationalTime += SowingTime * NumberOfTrays1
               Time += SowingTime * NumberOfBatches
               SowingDoneQueue.Enter(Batch)
         if Day - Timer > 10^-10 #very small
            Wait Day - Timer #wait till the next day

class TTrayCropHandlingToGreenhouseShuttle = SimElement #Active class that
    transports trays from crop handling to greenhouse
   def setup
      inputs: DriveTimes, Distances, runtime #Note DriveTime and Distance is an
          array with information for every compartment
      Initialize as UniformDistribution: InterArrivalTime (Constant (1 day))
   def process
      Day = InterArrivalTime.sample

      while SowingDoneQueue and Transplant1DoneQueue are empty: Standby

      while SowingDoneQueue is not empty:
         Batch = SowingDoneQueue.LeaveQueue
         if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
             pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
         elif GreenhouseC1.Capacity > (GreenhouseC1.Occupancy +
             AreaNeededForBatch) #Check if a batch can be added
            Wait LoadTime * NumberOfTrays1
            Wait DriveTime * NumberOfTrays1
            Wait UnloadTime * NumberOfTrays1
            Batch.TransportTimeCumulative += DriveTime * NumberOfTrays1
            Batch.TransportDistanceCumulative += Distance * NumberOfTrays1
            Batch.NumberOfTransportActionsCumulative += NumberOfTrays1
            GreenhouseC1Queue.add(Batch)
            GreenhouseC1.Occupancy += AreaNeededForBatch
         elif GreenhouseC2Capacity > (GreenhouseC2.Occpancy + AreaNeededForBatch)
             #Check the option to store in another compartment
```

```
            Wait LoadTime * NumberOfTrays1
            Wait DriveTime * NumberOfTrays1
            Wait UnloadTime * NumberOfTrays1
            Batch.TransportTimeCumulative += DriveTime * NumberOfTrays1
            Batch.TransportDistanceCumulative += Distance * NumberOfTrays1
            Batch.NumberOfTransportActionsCumulative += NumberOfTrays1
            Batch.PlacedInWrongCompartment = 'C1'
            GreenhouseC2Queue.add(Batch)
            GreenhouseC2.Occupancy += AreaNeededForBatch
        else #there is no space in neither compartment 1 or 2
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'waited too long after sowing'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                    LostCropsQueue.add(NewLostCrop)

    while Transplant1DoneQueue is not empty:
        Batch = Transplant1DoneQueue.LeaveQueue
        if Batch.Timer < CurrentTime - 1 #check if there is waiter too long for
            pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                    CurrentTime < 500
                LostCropsQueue.add(NewLostCrop)
        elif GreenhouseC2.Capacity > (GreenhouseC2.Occupancy +
            AreaNeededForBatch) #Check if a batch can be added
            Wait LoadTime * NumberOfTrays2
            Wait DriveTime * NumberOfTrays2
            Wait UnloadTime * NumberOfTrays2
            Batch.TransportTimeCumulative += DriveTime * NumberOfTrays2
            Batch.TransportDistanceCumulative += Distance * NumberOfTrays2
            Batch.NumberOfTransportActionsCumulative += NumberOfTrays2
            GreenhouseC2Queue.add(Batch)
            GreenhouseC2.Occupancy += AreaNeededForBatch
        elif GreenhouseC1Capacity > (GreenhouseC1.Occpancy + AreaNeededForBatch)
            #Check the option to store in another compartment
            Wait LoadTime * NumberOfTrays2
            Wait DriveTime * NumberOfTrays2
            Wait UnloadTime * NumberOfTrays2
            Batch.TransportTimeCumulative += DriveTime * NumberOfTrays2
            Batch.TransportDistanceCumulative += Distance * NumberOfTrays2
            Batch.NumberOfTransportActionsCumulative += NumberOfTrays2
            Batch.PlacedInWrongCompartment = 'C2'
            GreenhouseC1Queue.add(Batch)
            GreenhouseC1.Occupancy += AreaNeededForBatch
        else #there is no space in neither compartment 1 or 2
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'waited too long after sowing'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                    LostCropsQueue.add(NewLostCrop)
```

```
class TTrayGreenhouseToCropHandlingShuttle = SimElement #Active class that
    transports trays from greenhouse to crop handling area
   def setup
      inputs: DriveTimes, Distances, runtime
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      Day = InterArrivalTime.sample

      while Compartment1LeaveQueue and Compartment2LeaveQueue are empty: Standby

      while Compartment1LeaveQueue is not empty
         Batch = Compartment1LeaveQueue.LeaveQueue
         if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
             pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
         else
            if Batch.PlacedInWrongCompartment = '' #check if in right compartment
               Wait LoadTime * NumberOfTrays1
               Wait DriveTime * NumberOfTrays1
               Wait UnloadTime * NumberOfTrays1
               Batch.TransportTimeCumulative += DriveTime * NumberOfTrays1
               Batch.TransportDistanceCumulative += Distance * NumberOfTrays1
               Batch.NumberOfTransportActionsCumulative += NumberOfTrays1
               Batch.Timer = CurrentTime
               Transplant1Queue.add(Batch)
            else
               if BatchPlacedInWrongCompartment = 'C2'
                  if dFDS1 + dFDS2 < Batch.FDS #Check if ready for next
                      compartment (if ready for C2)
                     Wait LoadTime * NumberOfTrays1
                     Wait DriveTime * NumberOfTrays1
                     Wait UnloadTime * NumberOfTrays1
                     Batch.TransportTimeCumulative += DriveTime * NumberOfTrays1
                     Batch.TransportDistanceCumulative += Distance *
                         NumberOfTrays1
                     Batch.NumberOfTransportActionsCumulative += NumberOfTrays1
                     Batch.Timer = CurrentTime
                     Transplant1Queue.add(Batch)
                  elif GreenhouseC2Capacity > (GreenhouseC2.Occpancy +
                      AreaNeededForBatch) #Check if there is space in right
                      compartment 2
                     Wait LoadTime * NumberOfTrays2
                     Wait DriveTime * NumberOfTrays2
                     Wait UnloadTime * NumberOfTrays2
                     Batch.TransportTimeCumulative += DriveTime * NumberOfTrays2
                     Batch.TransportDistanceCumulative += Distance *
                         NumberOfTrays2
                     Batch.NumberOfTransportActionsCumulative += NumberOfTrays2
                     Batch.PlacedInWrongCompartment = ''
                     GreenhouseC2Queue.add(Batch)
                     GreenhouseC2.Occupancy += AreaNeededForBatch
                  elif GreenhouseC1Capacity > (GreenhouseC1.Occpancy +
                      AreaNeededForBatch) #Check if there is space in wrong
                      compartment 1
                     Wait LoadTime * NumberOfTrays1
```

```
                    Wait DriveTime * NumberOfTrays1
                    Wait UnloadTime * NumberOfTrays1
                    Batch.TransportTimeCumulative += DriveTime * NumberOfTrays1
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfTrays1
                    Batch.NumberOfTransportActionsCumulative += NumberOfTrays1
                    Batch.PlacedInWrongCompartment = 'C2'
                    GreenhouseC1Queue.add(Batch)
                    GreenhouseC1.Occupancy += AreaNeededForBatch
                else #there is no space in neither compartment 1 or 2
                    LostComponentQueue.add(Batch)
                    Loop over number of plants in Batch
                    NewLostCrop = TLostCrop
                    NewLostCrop.LossTime = CurrentTime
                    NewLostCrop.LossReason = 'waited too long after sowing'
                    if 135 < CurrentTime < 500 #only count lost crops for
                        representative year
                       LostCropsQueue.add(NewLostCrop)

class TGutterCropHandlingToGreenhouseShuttle = SimElement #Active class that
    transports gutters from crop handling area to greenhouse
   def setup
      inputs: DriveTimes, Distances, runtime #Note DriveTime and Distance is an
          array with information for every compartment
      Initialize as UniformDistribution: InterArrivalTime (Constant (1 day))
   def process
      Day = InterArrivalTime.sample

      while MonitoringDoneQueue and Transplant2DoneQueue are empty: Standby

      while Transplant2DoneQueue is not empty:
         Batch = Transplant2DoneQueue.LeaveQueue
         if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
             pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
         elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
             AreaNeededForBatch) #Check if a batch can be added
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            GreenhouseC3Queue.add(Batch)
            GreenhouseC3.Occupancy += AreaNeededForBatch
         elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
             AreaNeededForBatch) #Check if a batch can be added
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.PlacedInWrongCompartment = 'C3'
            GreenhouseC4aQueue.add(Batch)
```

```
            GreenhouseC4a.Occupancy += AreaNeededForBatch
        else #there is no space
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'No space in compartment'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                    LostCropsQueue.add(NewLostCrop)

    while MonitoringDoneQueue is not empty:
        Batch = MonitoringDoneQueue.LeaveQueue
        if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
            pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                    CurrentTime < 500
                LostCropsQueue.add(NewLostCrop)

        elif Batch.FDS < (dFDS1 + dFDS2 + dFDS3 + dFDS4a)

            if GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                AreaNeededForBatch) #Check if a batch can be added
                Wait LoadTime * NumberOfGutters
                Wait DriveTime * NumberOfGutters
                Wait UnloadTime * NumberOfGutters
                Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                Batch.TransportDistanceCumulative += Distance * NumberOfGutters
                Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                GreenhouseC4aQueue.add(Batch)
                GreenhouseC4a.Occupancy += AreaNeededForBatch
            elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                AreaNeededForBatch) #Check if a batch can be added
                Wait LoadTime * NumberOfGutters
                Wait DriveTime * NumberOfGutters
                Wait UnloadTime * NumberOfGutters
                Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                Batch.TransportDistanceCumulative += Distance * NumberOfGutters
                Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                GreenhouseC4bQueue.add(Batch)
                Batch.PlacedInWrongCompartment = 'C4a'
                GreenhouseC4b.Occupancy += AreaNeededForBatch
            elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
                AreaNeededForBatch) #Check if a batch can be added
                Wait LoadTime * NumberOfGutters
                Wait DriveTime * NumberOfGutters
                Wait UnloadTime * NumberOfGutters
                Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                Batch.TransportDistanceCumulative += Distance * NumberOfGutters
                Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                GreenhouseC3Queue.add(Batch)
                Batch.PlacedInWrongCompartment = 'C4a'
                GreenhouseC3.Occupancy += AreaNeededForBatch
            else #there is no space
                LostComponentQueue.add(Batch)
                Loop over number of plants in Batch
                    NewLostCrop = TLostCrop
```

```
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'No space in compartment'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                   LostCropsQueue.add(NewLostCrop)
         else
            if GreenhouseC5.Capacity > (GreenhouseC5.Occupancy +
                AreaNeededForBatch) #Check if a batch can be added
               Wait LoadTime * NumberOfGutters
               Wait DriveTime * NumberOfGutters
               Wait UnloadTime * NumberOfGutters
               Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
               Batch.TransportDistanceCumulative += Distance * NumberOfGutters
               Batch.NumberOfTransportActionsCumulative += NumberOfGutters
               GreenhouseC5Queue.add(Batch)
               GreenhouseC5.Occupancy += AreaNeededForBatch
            elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                AreaNeededForBatch) #Check if a batch can be added
               Wait LoadTime * NumberOfGutters
               Wait DriveTime * NumberOfGutters
               Wait UnloadTime * NumberOfGutters
               Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
               Batch.TransportDistanceCumulative += Distance * NumberOfGutters
               Batch.NumberOfTransportActionsCumulative += NumberOfGutters
               GreenhouseC4bQueue.add(Batch)
               Batch.PlacedInWrongCompartment = 'C5'
               GreenhouseC4b.Occupancy += AreaNeededForBatch
            else #there is no space
               LostComponentQueue.add(Batch)
               Loop over number of plants in Batch
                  NewLostCrop = TLostCrop
                  NewLostCrop.LossTime = CurrentTime
                  NewLostCrop.LossReason = 'No space in compartment'
                  if 135 < CurrentTime < 500 #only count lost crops for
                      representative year
                     LostCropsQueue.add(NewLostCrop)

class TGutterGreenhouseToCropHandlingShuttle = SimElement #Active class that
    transports gutters from greenhouse to crop handling area
   def setup
      inputs: DriveTimes, Distances, runtime
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      Day = InterArrivalTime.sample

      while Compartmen3LeaveQueue and Compartment4bLeaveQueue and
          Compartment5LeaveQueue are empty: Standby

      while Compartment3LeaveQueue is not empty
         Batch = Compartment3LeaveQueue.LeaveQueue
         if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
             pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
         else
            if Batch.PlacedInWrongCompartment = ''
```

```
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.Timer = CurrentTime
            MonitoringQueue.add(Batch)

        else
            if Batch.PlacedInWrongCompartment = 'C4a' #check if in right
                compartment
                Batch.PlacedInWrongCompartment = ''
                if (dFDS1 + dFDS2 + dFDS3 + dFDS4a) < Batch.FDS:
                    if GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                        AreaNeededForBatch
                        Wait LoadTime * NumberOfGutters
                        Wait DriveTime * NumberOfGutters
                        Wait UnloadTime * NumberOfGutters
                        Batch.TransportTimeCumulative += DriveTime *
                            NumberOfGutters
                        Batch.TransportDistanceCumulative += Distance *
                            NumberOfGutters
                        Batch.NumberOfTransportActionsCumulative +=
                            NumberOfGutters
                        Batch.Timer = CurrentTime
                        MonitoringQueue.add(Batch)
                    elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                        AreaNeededForBatch
                        Wait LoadTime * NumberOfGutters
                        Wait DriveTime * NumberOfGutters
                        Wait UnloadTime * NumberOfGutters
                        Batch.TransportTimeCumulative += DriveTime *
                            NumberOfGutters
                        Batch.TransportDistanceCumulative += Distance *
                            NumberOfGutters
                        Batch.NumberOfTransportActionsCumulative +=
                            NumberOfGutters
                        Batch.Timer = CurrentTime
                        Batch.PlaceInWrongCompartment = 'C4a'
                        GreenhouseC4b.Occupancy += AreaNeededForBatch
                        GreenhouseC4bQueue.add(Batch)
                    elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
                        AreaNeededForBatch
                        Wait LoadTime * NumberOfGutters
                        Wait DriveTime * NumberOfGutters
                        Wait UnloadTime * NumberOfGutters
                        Batch.TransportTimeCumulative += DriveTime *
                            NumberOfGutters
                        Batch.TransportDistanceCumulative += Distance *
                            NumberOfGutters
                        Batch.NumberOfTransportActionsCumulative +=
                            NumberOfGutters
                        Batch.Timer = CurrentTime
                        Batch.PlaceInWrongCompartment = 'C4a'
                        GreenhouseC3.Occupancy += AreaNeededForBatch
                        GreenhouseC3Queue.add(Batch)
                    else #there is no space in compartment
                        LostComponentQueue.add(Batch)
                        Loop over number of plants in Batch
                        NewLostCrop = TLostCrop
```

```
                    NewLostCrop.LossTime = CurrentTime
                    NewLostCrop.LossReason = 'No place in compartment'
                    if 135 < CurrentTime < 500 #only count lost crops for
                        representative year
                       LostCropsQueue.add(NewLostCrop)

while Compartment4aLeaveQueue is not empty
   Batch = Compartment4aLeaveQueue.LeaveQueue
   if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
       pick up
      LostComponentQueue.add(Batch)
      Loop over number of plants in Batch
         NewLostCrop = TLostCrop
         NewLostCrop.LossTime = CurrentTime
         NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
             CurrentTime < 500
         LostCropsQueue.add(NewLostCrop)
   else
      if Batch.PlacedInWrongCompartment = 'C3'
         Batch.PlacedInWrongCompartment = ''
         if (dFDS1 + dFDS2 + dFDS3) < Batch.FDS:
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.Timer = CurrentTime
            MonitoringQueue.add(Batch)
         elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
             AreaNeededForBatch
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.Timer = CurrentTime
            GreenhouseC3Queue.add(Batch)
         elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
             AreaNeededForBatch
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.Timer = CurrentTime
            Batch.PlacedInWrongCompartment = 'C3'
            GreenhouseC4aQueue.add(Batch)
         else #there is no space in compartment
               LostComponentQueue.add(Batch)
               Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'No place in compartment'
               if 135 < CurrentTime < 500 #only count lost crops for
                   representative year
                  LostCropsQueue.add(NewLostCrop)

      elif Batch.PlacedInWrongCompartment = 'C4b' #check if in right
```

```
            compartment
            Batch.PlacedInWrongCompartment = ''
            if (dFDS1 + dFDS2 + dFDS3 + dFDS4a + dFDS4b) < Batch.FDS:
               Wait LoadTime * NumberOfGutters
               Wait DriveTime * NumberOfGutters
               Wait UnloadTime * NumberOfGutters
               Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
               Batch.TransportDistanceCumulative += Distance * NumberOfGutters
               Batch.NumberOfTransportActionsCumulative += NumberOfGutters
               Batch.Timer = CurrentTime
               MonitoringQueue.add(Batch)
            elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                AreaNeededForBatch
               Wait LoadTime * NumberOfGutters
               Wait DriveTime * NumberOfGutters
               Wait UnloadTime * NumberOfGutters
               Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
               Batch.TransportDistanceCumulative += Distance * NumberOfGutters
               Batch.NumberOfTransportActionsCumulative += NumberOfGutters
               Batch.Timer = CurrentTime
               GreenhouseC4bQueue.add(Batch)
            elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                AreaNeededForBatch
               Wait LoadTime * NumberOfGutters
               Wait DriveTime * NumberOfGutters
               Wait UnloadTime * NumberOfGutters
               Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
               Batch.TransportDistanceCumulative += Distance * NumberOfGutters
               Batch.NumberOfTransportActionsCumulative += NumberOfGutters
               Batch.Timer = CurrentTime
               Batch.PlacedInWrongCompartment = 'C4b'
               GreenhouseC4aQueue.add(Batch)
            else #there is no space in compartment
                  LostComponentQueue.add(Batch)
                  Loop over number of plants in Batch
                  NewLostCrop = TLostCrop
                  NewLostCrop.LossTime = CurrentTime
                  NewLostCrop.LossReason = 'No place in compartment'
                  if 135 < CurrentTime < 500 #only count lost crops for
                      representative year
                     LostCropsQueue.add(NewLostCrop)
         else #there is no space in compartment
               LostComponentQueue.add(Batch)
               Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'No place in compartment'
               if 135 < CurrentTime < 500 #only count lost crops for
                   representative year
                  LostCropsQueue.add(NewLostCrop)

   while Compartment4bLeaveQueue is not empty
      Batch = Compartment3LeaveQueue.LeaveQueue
      if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
          pick up
         LostComponentQueue.add(Batch)
         Loop over number of plants in Batch
            NewLostCrop = TLostCrop
            NewLostCrop.LossTime = CurrentTime
            NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                CurrentTime < 500
```

```
            LostCropsQueue.add(NewLostCrop)
        else
            if Batch.PlacedInWrongCompartment = ''
                Wait LoadTime * NumberOfGutters
                Wait DriveTime * NumberOfGutters
                Wait UnloadTime * NumberOfGutters
                Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                Batch.TransportDistanceCumulative += Distance * NumberOfGutters
                Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                Batch.Timer = CurrentTime
                MonitoringQueue.add(Batch)

            else
                if Batch.PlacedInWrongCompartment = 'C4a' #check if in right
                    compartment
                    Batch.PlacedInWrongCompartment = ''
                    if (dFDS1 + dFDS2 + dFDS3 + dFDS4a) < Batch.FDS:
                        if GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                            AreaNeededForBatch
                            Wait LoadTime * NumberOfGutters
                            Wait DriveTime * NumberOfGutters
                            Wait UnloadTime * NumberOfGutters
                            Batch.TransportTimeCumulative += DriveTime *
                                NumberOfGutters
                            Batch.TransportDistanceCumulative += Distance *
                                NumberOfGutters
                            Batch.NumberOfTransportActionsCumulative +=
                                NumberOfGutters
                            Batch.Timer = CurrentTime
                            Monitoring.add(Batch)
                        elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                            AreaNeededForBatch
                            Wait LoadTime * NumberOfGutters
                            Wait DriveTime * NumberOfGutters
                            Wait UnloadTime * NumberOfGutters
                            Batch.TransportTimeCumulative += DriveTime *
                                NumberOfGutters
                            Batch.TransportDistanceCumulative += Distance *
                                NumberOfGutters
                            Batch.NumberOfTransportActionsCumulative +=
                                NumberOfGutters
                            Batch.Timer = CurrentTime
                            Batch.PlaceInWrongCompartment = ''
                            GreenhouseC4a.Occupancy += AreaNeededForBatch
                            GreenhouseC4aQueue.add(Batch)
                        elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                            AreaNeededForBatch
                            Wait LoadTime * NumberOfGutters
                            Wait DriveTime * NumberOfGutters
                            Wait UnloadTime * NumberOfGutters
                            Batch.TransportTimeCumulative += DriveTime *
                                NumberOfGutters
                            Batch.TransportDistanceCumulative += Distance *
                                NumberOfGutters
                            Batch.NumberOfTransportActionsCumulative +=
                                NumberOfGutters
                            Batch.Timer = CurrentTime
                            Batch.PlaceInWrongCompartment = 'C4a'
                            GreenhouseC4b.Occupancy += AreaNeededForBatch
                            GreenhouseC4bQueue.add(Batch)
                        elif GreenhouseC3.Capacity > (Greenhouse3.Occupancy +
```

```
            AreaNeededForBatch
             Wait LoadTime * NumberOfGutters
             Wait DriveTime * NumberOfGutters
             Wait UnloadTime * NumberOfGutters
             Batch.TransportTimeCumulative += DriveTime *
                 NumberOfGutters
             Batch.TransportDistanceCumulative += Distance *
                 NumberOfGutters
             Batch.NumberOfTransportActionsCumulative +=
                 NumberOfGutters
             Batch.Timer = CurrentTime
             Batch.PlaceInWrongCompartment = 'C4a'
             GreenhouseC4.Occupancy += AreaNeededForBatch
             GreenhouseC4Queue.add(Batch)
          else #there is no space in compartment
             LostComponentQueue.add(Batch)
             Loop over number of plants in Batch
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'No place in compartment'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                   LostCropsQueue.add(NewLostCrop)

elif Batch.PlacedInWrongCompartment = 'C5' #check if in right
    compartment
   Batch.PlacedInWrongCompartment = ''
   if 1 < Batch.FDS:
      Wait LoadTime * NumberOfGutters
      Wait DriveTime * NumberOfGutters
      Wait UnloadTime * NumberOfGutters
      Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
      Batch.TransportDistanceCumulative += Distance *
          NumberOfGutters
      Batch.NumberOfTransportActionsCumulative += NumberOfGutters
      Batch.Timer = CurrentTime
      HarvestingQueue.add(Batch)
    elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
        AreaNeededForBatch
      Wait LoadTime * NumberOfGutters
      Wait DriveTime * NumberOfGutters
      Wait UnloadTime * NumberOfGutters
      Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
      Batch.TransportDistanceCumulative += Distance *
          NumberOfGutters
      Batch.NumberOfTransportActionsCumulative += NumberOfGutters
      Batch.Timer = CurrentTime
      HarvestingQueue.add(Batch)
   elif GreenhouseC5.Capacity > (GreenhouseC5.Occupancy +
       AreaNeededForBatch
      Wait LoadTime * NumberOfGutters
      Wait DriveTime * NumberOfGutters
      Wait UnloadTime * NumberOfGutters
      Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
      Batch.TransportDistanceCumulative += Distance *
          NumberOfGutters
      Batch.NumberOfTransportActionsCumulative += NumberOfGutters
      Batch.Timer = CurrentTime
      Batch.PlaceInWrongCompartment = ''
      GreenhouseC5.Occupancy += AreaNeededForBatch
      GreenhouseC5Queue.add(Batch)
```

```
                elif GreenhouseC4b.Capacity > (GreenhouseC5.Occupancy +
                    AreaNeededForBatch
                    Wait LoadTime * NumberOfGutters
                    Wait DriveTime * NumberOfGutters
                    Wait UnloadTime * NumberOfGutters
                    Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfGutters
                    Batth.NumberOfTransportActionsCumulative += NumberOfGutters
                    Batch.Timer = CurrentTime
                    Batch.PlaceInWrongCompartment = 'C5'
                    GreenhouseC4b.Occupancy += AreaNeededForBatch
                    GreenhouseC4bQueue.add(Batch)
                else #there is no space in compartment
                    LostComponentQueue.add(Batch)
                    Loop over number of plants in Batch
                        NewLostCrop = TLostCrop
                        NewLostCrop.LossTime = CurrentTime
                        NewLostCrop.LossReason = 'No place in compartment'
                        if 135 < CurrentTime < 500 #only count lost crops for
                            representative year
                            LostCropsQueue.add(NewLostCrop)

while Compartment5LeaveQueue is not empty
    Batch = Compartment3LeaveQueue.LeaveQueue
    if Batch.Timer < CurrentTime - 1 #check if there is waited too long for
        pick up
        LostComponentQueue.add(Batch)
        Loop over number of plants in Batch
            NewLostCrop = TLostCrop
            NewLostCrop.LossTime = CurrentTime
            NewLostCrop.LossReason = 'waited too long after sowing' if 135 <
                CurrentTime < 500
            LostCropsQueue.add(NewLostCrop)
    else
        if Batch.PlacedInWrongCompartment = ''
            Wait LoadTime * NumberOfGutters
            Wait DriveTime * NumberOfGutters
            Wait UnloadTime * NumberOfGutters
            Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
            Batch.TransportDistanceCumulative += Distance * NumberOfGutters
            Batch.NumberOfTransportActionsCumulative += NumberOfGutters
            Batch.Timer = CurrentTime
            HarvestingQueue.add(Batch)

        else
            if Batch.PlacedInWrongCompartment = 'C4b' #check if in right
                compartment
                Batch.PlacedInWrongCompartment = ''
                if 1 < Batch.FDS:
                    Wait LoadTime * NumberOfGutters
                    Wait DriveTime * NumberOfGutters
                    Wait UnloadTime * NumberOfGutters
                    Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfGutters
                    Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                    Batch.Timer = CurrentTime
                    HarvestingQueue.add(Batch)

                elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
```

```
                    AreaNeededForBatch
                    Wait LoadTime * NumberOfGutters
                    Wait DriveTime * NumberOfGutters
                    Wait UnloadTime * NumberOfGutters
                    Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfGutters
                    Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                    Batch.Timer = CurrentTime
                    GreenhouseC4b.Occupancy += AreaNeededForBatch
                    GreenhouseC4bQueue.add(Batch)
                elif GreenhouseC5.Capacity > (GreenhouseC5.Occupancy +
                    AreaNeededForBatch
                    Wait LoadTime * NumberOfGutters
                    Wait DriveTime * NumberOfGutters
                    Wait UnloadTime * NumberOfGutters
                    Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfGutters
                    Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                    Batch.Timer = CurrentTime
                    Batch.PlaceInWrongCompartment = 'C4b'
                    GreenhouseC5.Occupancy += AreaNeededForBatch
                    GreenhouseC5Queue.add(Batch)
                elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                    AreaNeededForBatch
                    Wait LoadTime * NumberOfGutters
                    Wait DriveTime * NumberOfGutters
                    Wait UnloadTime * NumberOfGutters
                    Batch.TransportTimeCumulative += DriveTime * NumberOfGutters
                    Batch.TransportDistanceCumulative += Distance *
                        NumberOfGutters
                    Batch.NumberOfTransportActionsCumulative += NumberOfGutters
                    Batch.Timer = CurrentTime
                    Batch.PlaceInWrongCompartment = 'C4b'
                    GreenhouseC4a.Occupancy += AreaNeededForBatch
                    GreenhouseC4aueue.add(Batch)
                else #there is no space in compartment
                    LostComponentQueue.add(Batch)
                    Loop over number of plants in Batch
                        NewLostCrop = TLostCrop
                        NewLostCrop.LossTime = CurrentTime
                        NewLostCrop.LossReason = 'No place in compartment'
                        if 135 < CurrentTime < 500 #only count lost crops for
                            representative year
                            LostCropsQueue.add(NewLostCrop)

class TTransplant1Machine = SimElement # Active class that tranplants from tray 1
    to tray 2
    def setup
        inputs: HandlingTime, LossRate # LossRate is currently only zero
        Initialize as zeros array of length runtime: OperationalTime
    def process
        while Transplant1Queue is empty: Standby

        while Transplant1Queue is not empty
            Batch = Transplant1Queue.LeaveQueue
            if Batch.Timer < CurrentTime - 1/6 #check if there is waited too long
                for pick up
                LostComponentQueue.add(Batch)
                Loop over number of plants in Batch
```

```
                NewLostCrop = TLostCrop
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'waited too long for Transplanting 1' if
                    135 < CurrentTime < 500
                LostCropsQueue.add(NewLostCrop)
            else
                Wait HandlingTime * Batch.NumberOfTrays1
                OperationalTime[integer(CurrentTime)] += HandlingTime *
                    Batch.NumberOfTrays1
                Batch.Timer = CurrentTime
                Transplant1DoneQueue.add(Batch)

class TTransplant2Machine = SimElement # Active class that transplant from tray 2
    and gutter
    def setup
        inputs: HandlingTime, LossRate # LossRate is currently only zero
        Initialize as zeros array of length runtime: OperationalTime
    def process
        while Transplant2Queue is empty: Standby

        while Transplant2Queue is not empty
            Batch = Transplant2Queue.LeaveQueue
            if Batch.Timer < CurrentTime - 1/6 #check if there is waited too long
                for pick up
                LostComponentQueue.add(Batch)
                Loop over number of plants in Batch
                    NewLostCrop = TLostCrop
                    NewLostCrop.LossTime = CurrentTime
                    NewLostCrop.LossReason = 'waited too long for Transplanting 2' if
                        135 < CurrentTime < 500
                    LostCropsQueue.add(NewLostCrop)
            else
                Wait HandlingTime * Batch.NumberOfTrays2
                OperationalTime[integer(CurrentTime)] += HandlingTime *
                    Batch.NumberOfTrays2
                Batch.Timer = CurrentTime
                Transplant2DoneQueue.add(Batch)

class TMonitoringMachien = SimElement # Active class that monitors between the
    gutters to go to a new compartment (possibly a handling task can be performed
    as well)
    def setup
        inputs: HandlingTime, LossRate # LossRate is currently only zero
        Initialize as zeros array of length runtime: OperationalTime
    def process
        while MonitoringQueue is empty: Standby

        while MonitoringQueue is not empty
            Batch = MonitoringQueue.LeaveQueue
            if Batch.Timer < CurrentTime - 1/6 #check if there is waited too long
                for pick up
                LostComponentQueue.add(Batch)
                Loop over number of plants in Batch
                    NewLostCrop = TLostCrop
                    NewLostCrop.LossTime = CurrentTime
                    NewLostCrop.LossReason = 'waited too long for Monitoring' if 135 <
                        CurrentTime < 500
                    LostCropsQueue.add(NewLostCrop)
            else
                Wait HandlingTime * Batch.NumberOfGutters
                OperationalTime[integer(CurrentTime)] += HandlingTime *
```

```
                Batch.NumberOfGutters
            Batch.Timer = CurrentTime
            MonitoringDoneQueue.add(Batch)

class THarvestingMachine = SimElement # Active class that harvest the trusses
   def setup
      inputs: HandlingTime, LossRate # LossRate here is not for machine, but for
          entire process
      Initialize as zeros array of length runtime: OperationalTime
   def process
      while HarvestingQueue is empty: standy

      while HarvestingQueue is not empty
         Batch = HarvestingQueue.LeaveQueue
         if Batch.Timer < CurrentTime - 1/6 #check if there is waited too long
             for pick up
            LostComponentQueue.add(Batch)
            Loop over number of plants in Batch
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'waited too long for harvesting' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
         else
            Wait HandlingTime * NumberOfGutters
            OperationalTime[integer(CurrentTime)] += HandlingTime *
                Batch.NumberOfGutters
            Batch.Timer = CurrentTime
            # Apply Loss Rate
            NumberOfLostPlants = LostRate * Batch.NumberOfPlants
            Batch.NumberOfPlants -= NumberOfLostPlants
            Loop over NumberOfLostPlants
               NewLostCrop = TLostCrop
               NewLostCrop.LossTime = CurrentTime
               NewLostCrop.LossReason = 'Lost due to applied LossRate' if 135 <
                   CurrentTime < 500
               LostCropsQueue.add(NewLostCrop)
            Loop over Batch.NumberOfPlants
               Loop of Batch.TrussesPerPlant
                  newHarvestedTruss = THarvestedTruss()
                  Assign TW, CurrentTime, and Revenue to HarvestedTruss
                  HarvestedTrussQueue.add(newHarvestedTruss)


class TGreenhouseCompartment1 = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment1Queue.LeaveQueue
         Batch.Timer = CurrentTime
         Batch.CurrentTime = CurrentTime
         Batch.FDS += FDR
         if BatchPlacedInWrongComprtment = 'C2' # if batch desires compartment 1
             instead of 2
```

```
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment1.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment1LeaveQueue.add(Batch)
         elif Batch.FDS >= dFDS # check if batch is ready for next step
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            GreenhouseCompartment1.Occupancy -= AreaNeededForBatch
            Compartment1LeaveQueue.add(Batch)
         else
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Compartment1Queue.add(Batch)
      Wait day

class TGreenhouseCompartment2 = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment2Queue.LeaveQueue
         Batch.Timer = CurrentTime
         Batch.CurrentTime = CurrentTime
         Batch.FDS += FDR
         if BatchPlacedInWrongComprtment = 'C1' # if batch desires compartment 2
             instead of 1
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment2.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment2LeaveQueue.add(Batch)
         elif Batch.FDS >= dFDS # check if batch is ready for next step
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            GreenhouseCompartment2.Occupancy -= AreaNeededForBatch
            Compartment1LeaveQueue.add(Batch)
         else
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Compartment2Queue.add(Batch)
      Wait day

class TGreenhouseCompartment3 = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment3Queue.LeaveQueue
         Batch.Timer = CurrentTime
         Batch.CurrentTime = CurrentTime
         Batch.FDS += FDR
         if BatchPlacedInWrongComprtment = 'C4a' # if batch desires compartment
```

```
            4a instead of 3
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment3.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment3LeaveQueue.add(Batch)
         elif Batch.FDS >= dFDS # check if batch is ready for next step
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            GreenhouseCompartment3.Occupancy -= AreaNeededForBatch
            Compartment3LeaveQueue.add(Batch)
         else
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Compartment3Queue.add(Batch)
      Wait day

class TGreenhouseCompartment4a = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment3Queue.LeaveQueue
         Batch.Timer = CurrentTime
         Batch.CurrentTime = CurrentTime
         Batch.FDS += FDR
         if BatchPlacedInWrongComprtment = 'C3' # if batch desires compartment 3
             instead of 4a
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment4a.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment4aLeaveQueue.add(Batch)
         elif BatchPlacedInWrongComprtment = 'C4b' # if batch desires compartment
             4b instead of 4a
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment4a.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment4aLeaveQueue.add(Batch)
         elif Batch.FDS >= dFDS # check if batch is ready for next step
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            GreenhouseCompartment4a.Occupancy -= AreaNeededForBatch
                Batch.PlacedInWrongCompartment = ''
            Compartment4bLeaveQueue.add(Batch)

            if GreenhouseC4b.Capacity > (Greenhouse4b.Occupancy +
                AreaNeededForBatch)
               Compartment4bQueue.add(Batch)
            elif GreenhouseC4a.Capacity > (Greenhouse4a.Occupancy +
                AreaNeededForBatch)
               Batch.PlacedInWrongCompartment = 'C4b'
               Compartment4aQueue.add(Batch)
            else #there is no space in neither compartment 4a or 4b
               LostComponentQueue.add(Batch)
               Loop over number of plants in Batch
               NewLostCrop = TLostCrop
```

```
                NewLostCrop.LossTime = CurrentTime
                NewLostCrop.LossReason = 'No place in 4a or 4b'
                if 135 < CurrentTime < 500 #only count lost crops for
                    representative year
                   LostCropsQueue.add(NewLostCrop)
          else
             Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
             Batch.Compartment4aQueue.add(Batch)
       Wait day

class TGreenhouseCompartment4b = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment3Queue.LeaveQueue
         Batch.Timer = CurrentTime
         Batch.CurrentTime = CurrentTime
         Batch.FDS += FDR
         if BatchPlacedInWrongComprtment = 'C4a' # if batch desires compartment
             4a instead of 4b
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment4b.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment4bLeaveQueue.add(Batch)
         elif BatchPlacedInWrongComprtment = 'C5' # if batch desires compartment
             5 instead of 4b
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Batch.TimeInWrongCompartment += 1
            GreenhouseCompartment4b.Occupancy -= AreaNeededForBatch
            Batch.PlacedInWrongCompartment = ''
            Compartment4bLeaveQueue.add(Batch)
         elif Batch.FDS >= dFDS # check if batch is ready for next step
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            GreenhouseCompartment3.Occupancy -= AreaNeededForBatch
            Compartment4bLeaveQueue.add(Batch)
         else
            Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
            Compartment4bQueue.add(Batch)
      Wait day


class TGreenhouseCompartment5 = SimElement #Active class where the cultivation
    takes place
   def setup
      Inputs: FDR, DLI, LUE, runtime, Capacity, dFDS # note that by these inputs
          both the FDR can be calculated realtime as on beforehand
      Initialize as integer: Occupancy
      Initialize as Uniform Distribution: InterArrivalTime (Constant (1 day))
   def process
      day = InterArrivalTime.sample
      CurrentTime += day
      Loop over length of Compartment1Queue
         Batch = Compartment3Queue.LeaveQueue
```

```
        Batch.Timer = CurrentTime
        Batch.CurrentTime = CurrentTime
        Batch.FDS += FDR
        if BatchPlacedInWrongComprtment = 'C4b' # if batch desires compartment
            4b instead of 5
           Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
           Batch.TimeInWrongCompartment += 1
           GreenhouseCompartment5.Occupancy -= AreaNeededForBatch
           Batch.PlacedInWrongCompartment = ''
           Compartment5LeaveQueue.add(Batch)
        elif Batch.FDS >= dFDS # check if batch is ready for next step
           Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
           GreenhouseCompartment5.Occupancy -= AreaNeededForBatch
           Compartment5LeaveQueue.add(Batch)
        else
           Batch.TrussWeight += DLI * LUE / Batch.TrussDensity
           Compartment5Queue.add(Batch)
     Wait day
```

```
# Perform simulation, use the salabim package
env = salabim.Environment (trace = False, time_unit = 'days', random_seed =
    1234567)

# Initialize Queues that represents greenhouse compartments
Compartment1Queue = sim.Queue
Compartment2Queue = sim.Queue
Compartment3Queue = sim.Queue
Compartment4aQueue = sim.Queue
Compartment4bQueue = sim.Queue
Compartment5Queue = sim.Queue

# Initialize Queues that hold batches that need to leave a compartment
Compartment1LeaveQueue = sim.Queue
Compartment2LeaveQueue = sim.Queue
Compartment3LeaveQueue = sim.Queue
Compartment4aLeaveQueue = sim.Queue
Compartment4bLeaveQueue = sim.Queue
Compartment5LeaveQueue = sim.Queue

# Initialize Queues that define machinery / labour
SowingMachineQueue = sim.Queue
Transplant1Queue = sim.Queue
Transplant2Queue = sim.Queue
MonitoringQueue = sim.Queue
HarvestingQueue = sim.Queue

#Initialize Queues that are used for post analysis
HarvestedTrussQueue = sim.Queue
LostcropsQueue = sim.Queue
LostComponentQueue = sim.Queue

# Initialize machinery and compartments
SowingGenerator = TSwoingGenerator
TrayCropGreenhouseShuttle = TTrayCropHandlingAreaToGreenhouseShuttle
TrayGreenhouseCropShuttle = TTrayGreenhouseToCropHandlingAreaShuttle
GutterGreenhouseCropShuttle = TGutterGreenhouseToCropHandlingAreaShuttle
GutterCropGreenhouseShuttle = TGreenhouseToCropHandlingAreaShuttle

GreenhouseC1 = TGreenhouseCompartment1
GreenhouseC2 = TGreenhouseCompartment2
```

```
GreenhouseC3 = TGreenhouseCompartment3
GreenhouseC4a = TGreenhouseCompartment4a
GreenhouseC4b = TGreenhosuecompartment4b
GreenhouseC5 = TGreenhouseCompartment5

Define Monitor to keep track over changes

# Simulated
Define model name
Define use of monitor True or False
Define speed of monitor

Simulate over runtime

Fill arrays with all kinds of relevant information from simulation
Plot information
print information
```

C

Full code

In this appendix the full code as it is implemented for the simulation model is given. There is worked with one single file here, while one can also choice to handle different parts or functions in seperate files.

```
# ======================================================================
# Floris Bunnik
# Ridder Growing Solutions / TU Delft
# August 2022
# ======================================================================
# =========== Load packages ============================================
# ======================================================================
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
from scipy import signal
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import seaborn as sns
import scipy as sp
from gurobipy import *
import salabim as sim
from sklearn.preprocessing import MinMaxScaler
import timeit as timeit
# ======================================================================
# ======================================================================
# ========== Input Settings ============================================
# ======================================================================
Temperature_strategy = 'range' # Should be 'range', 'range + 1', 'range + noise',
    '20', '21', '22'
Light = 'DLI curve' # Should be 'DLI curve', DLI curve + noise
plant_density = 'free' # Should be 'free'
Cultivar = 'single' # Now only 'single' is implemented
LossRate = 0 # Can be any integer (interpreted as a percentage)
price_curve = 'constant' # 'constant', 'block', 'EUprice' curve are allowed
orientation = 'short rows' # should be 'short rows' or 'long rows'
no_crop_handling = 'double' # should be 'single' or 'double'
Base_data = 'Melbourne' #should either be 'typical values', 'Rotterdam',
    'Darwin', or 'Melbourne'
year = '2020-2021' # should either be '2016-2017', '2017-2018', '2018-2019',
    '2019-2020', '2020-2021' or '10-year mean'
Number_of_Transport_Lanes = 2 #should be a positive integer
Add_row_to_compartment = np.array(['C3',
    'C4b'])#np.array(['C2','C3'])#np.array(['C4b', 'C3'])#np.array(['C3', 'C4b'])
```

```
    #Fill with as many as wanted; respect maximum area
Remove_row_from_compartment = np.array(['C1', 'C2', 'C5'])#np.array(['C1', 'C5'])
#np.array(['C1', 'C2', 'C5'])#np.array(['C1', 'C2', 'C5']) #Fill with as many as
    wanted; respect maximum area
Factor_to_sowing_rate = 1.08#1.07 # default = 1; apply factor to sowing rate
Factor_to_diff_sowing_rate = 1.6 #default = 1; apply factor to increasing or
    decreasing periods
Length = 300 #available area length [m] -> over this length there are rows
Width = 150 #availabe area width [m] -> this is the length of a row
# ======================================================================
# ============= Make definitions ======================================
# ======================================================================
dFDS1, dFDS2, dFDS3, dFDS4a, dFDS4b, dFDS5 = 0.03, 0.09, 0.18, 0.24, 0.24, 0.30
    #[-] delta fruit development state for compartment
Growth_per_stage = [10, 20, 50 , 155, 155, 120] # Grams
DLI_upper = 12 # upper bound DLI curve [MJ/m^2/day]
DLI_lower = 3 # lower bound DLI curve [MJ/m^2/day]
runtime = 730 #[days]; 2 years to cut off the start and stop effects
batchsize = 500
plants_per_tray_1 = np.ones((runtime)) * 16
plants_per_tray_2 = np.ones((runtime)) * 8
Transmission_rate = 0.8 #Average transmission greenhouse cover (average over year
    and day)
PAR_ratio = 0.5
Tmin, Tmax = 17, 23 #[Celsius] Feasible temeprature range
#Target_Truss_weight = 510 #[grams]
LUE = 46 #[gram/MJ]
number_of_compartment = 6

Sowed = np.zeros((runtime))
OccupancyRates = np.zeros((number_of_compartment, runtime))
WrongCompartment = np.zeros((number_of_compartment, runtime))
TW = np.zeros((runtime))
if Cultivar == 'single':
   M_acc = 0.397
   B_acc = 4.38
   aFDR = 0.017
   dm_per = 6
   source_sink = 0.5
   target_fresh_weight = 90
   target_fruits_truss = 6
   trusses_per_plant = 3
   weekly_fruit_growth = 2
   FDMC0, FDMC1, FDMC = 10, 5.5, 10

Area = Width * Length #[m^2]
Crop_handling_area = 0.2 * Area #[m^2]
Crop_handling_width = Crop_handling_area / Width
Transport_system_width = 10 #[m]
Gutter_width = 8 #[m]
Gutter_length = 0.25 #[m]
Tray_width = 3 #[m]
Tray_length = 0.4 #[m]
crop_handling_length = Width
start_day = 0

if orientation == 'short rows':
   Row_length = Width - 2 * Transport_system_width
   Width_for_rows = Length - (Crop_handling_area) / crop_handling_length
   Transport_area = 0.8 * Length * Transport_system_width * 2
   Effective_area = Area - Crop_handling_area - Transport_area
```

```
if orientation == 'long rows':
   Row_length = Length - 2 * Transport_system_width
   Width_for_rows = Length - 2* Transport_system_width
   Transport_area = Row_length * Transport_system_width * 2
   Effective_area = Area - Crop_handling_area - Transport_area
Areas_trays =
    np.arange(Tray_width*Row_length,Effective_area/10,Tray_width*Row_length)
    #discrete step compartment trays
Areas_gutters =
    np.arange(Gutter_width*Row_length,Effective_area,Gutter_width*Row_length)
    #discrete step compartmetn tray
Areas = [Areas_trays, Areas_gutters]
Tray1Capacity = 16
Tray2Capacity = 8 # factor 1/2 of tray 1 capacity
GutterCapacity = 16 # factor 1/2 of gutter capacity
s = pd.date_range('2020-01-01', periods=runtime, freq='D').to_series()

if Base_data == 'Rotterdam': #load 10 year average curve
   WeatherDataRotterdam = pd.read_excel("Weer Rotterdam.xlsx", skiprows=[0],
       sheet_name="etmgeg_344")

   W2012 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20120000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20130000)]
   W2013 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20130000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20140000)]
   W2014 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20140000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20150000)]
   W2015 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20150000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20160000)]
   W2016 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20160000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20170000)]
   W2017 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20170000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20180000)]
   W2018 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20180000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20190000)]
   W2019 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20190000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20200000)]
   W2020 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20200000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20210000)]
   W2021 = WeatherDataRotterdam.loc[
         (WeatherDataRotterdam["YYYYMMDD"] > 20210000) &
             (WeatherDataRotterdam["YYYYMMDD"] < 20220000)]
   W2012 = pd.DataFrame.reset_index(W2012)
   W2013 = pd.DataFrame.reset_index(W2013)
   W2014 = pd.DataFrame.reset_index(W2014)
   W2015 = pd.DataFrame.reset_index(W2015)
   W2016 = pd.DataFrame.reset_index(W2016)
   W2017 = pd.DataFrame.reset_index(W2017)
   W2018 = pd.DataFrame.reset_index(W2018)
   W2019 = pd.DataFrame.reset_index(W2019)
   W2020 = pd.DataFrame.reset_index(W2020)
   W2021 = pd.DataFrame.reset_index(W2021)
```

```
dTG = {'2012': W2012[" TG"], '2013': W2013[" TG"], '2014': W2014[" TG"],
    '2015': W2015[" TG"], '2016': W2016[" TG"],
      '2017': W2017[" TG"], '2018': W2018[" TG"], '2019': W2019[" TG"], '2020':
          W2020[" TG"], '2021': W2021[" TG"]}
dGQ = {'2012': W2012[" Q"], '2013': W2013[" Q"], '2014': W2014[" Q"], '2015':
    W2015[" Q"], '2016': W2016[" Q"],
      '2017': W2017[" Q"], '2018': W2018[" Q"], '2019': W2019[" Q"], '2020':
          W2020[" Q"], '2021': W2021[" Q"]}

df_weather_TG = pd.DataFrame(data = dTG)
df_weather_GQ = pd.DataFrame(data = dGQ)

df_weather_TG['2012'] = df_weather_TG['2012'].astype(float)
df_weather_TG['2013'] = df_weather_TG['2013'].astype(float)
df_weather_TG['2014'] = df_weather_TG['2014'].astype(float)
df_weather_TG['2015'] = df_weather_TG['2015'].astype(float)
df_weather_TG['2016'] = df_weather_TG['2016'].astype(float)
df_weather_TG['2017'] = df_weather_TG['2017'].astype(float)
df_weather_TG['2018'] = df_weather_TG['2018'].astype(float)
df_weather_TG['2019'] = df_weather_TG['2019'].astype(float)
df_weather_TG['2020'] = df_weather_TG['2020'].astype(float)
df_weather_TG['2021'] = df_weather_TG['2021'].astype(float)

df_weather_GQ['2012'] = df_weather_GQ['2012'].astype(float)
df_weather_GQ['2013'] = df_weather_GQ['2013'].astype(float)
df_weather_GQ['2014'] = df_weather_GQ['2014'].astype(float)
df_weather_GQ['2015'] = df_weather_GQ['2015'].astype(float)
df_weather_GQ['2016'] = df_weather_GQ['2016'].astype(float)
df_weather_GQ['2017'] = df_weather_GQ['2017'].astype(float)
df_weather_GQ['2018'] = df_weather_GQ['2018'].astype(float)
df_weather_GQ['2019'] = df_weather_GQ['2019'].astype(float)
df_weather_GQ['2020'] = df_weather_GQ['2020'].astype(float)
df_weather_GQ['2021'] = df_weather_GQ['2021'].astype(float)

df_weather_TG['mean'] = df_weather_TG.mean(axis=1, skipna = True)
df_weather_GQ['mean'] = df_weather_GQ.mean(axis=1, skipna = True)
TG = np.zeros((runtime))
GQ = np.zeros((runtime))

for i in range(runtime):
   if i < 365:
      df_weather_TG['mean'][i] = df_weather_TG['mean'][i] / 10 + 3
      df_weather_TG['2012'][i] = df_weather_TG['2012'][i] / 10 + 3
      df_weather_TG['2013'][i] = df_weather_TG['2013'][i] / 10 + 3
      df_weather_TG['2014'][i] = df_weather_TG['2014'][i] / 10 + 3
      df_weather_TG['2015'][i] = df_weather_TG['2015'][i] / 10 + 3
      df_weather_TG['2016'][i] = df_weather_TG['2016'][i] / 10 + 3
      df_weather_TG['2017'][i] = df_weather_TG['2017'][i] / 10 + 3
      df_weather_TG['2018'][i] = df_weather_TG['2018'][i] / 10 + 3
      df_weather_TG['2019'][i] = df_weather_TG['2019'][i] / 10 + 3
      df_weather_TG['2020'][i] = df_weather_TG['2020'][i] / 10 + 3
      df_weather_TG['2021'][i] = df_weather_TG['2021'][i] / 10 + 3

      df_weather_GQ['2012'][i] = df_weather_GQ['2012'][i] * Transmission_rate
          * PAR_ratio / 100
      df_weather_GQ['2013'][i] = df_weather_GQ['2013'][i] * Transmission_rate
          * PAR_ratio / 100
      df_weather_GQ['2014'][i] = df_weather_GQ['2014'][i] * Transmission_rate
          * PAR_ratio / 100
      df_weather_GQ['2015'][i] = df_weather_GQ['2015'][i] * Transmission_rate
```

```
            * PAR_ratio / 100
        df_weather_GQ['2016'][i] = df_weather_GQ['2016'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['2017'][i] = df_weather_GQ['2017'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['2018'][i] = df_weather_GQ['2018'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['2019'][i] = df_weather_GQ['2019'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['2020'][i] = df_weather_GQ['2020'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['2021'][i] = df_weather_GQ['2021'][i] * Transmission_rate
            * PAR_ratio / 100
        df_weather_GQ['mean'][i] = df_weather_GQ['mean'][i] * Transmission_rate
            * PAR_ratio / 100

        TG[i] = df_weather_TG['mean'][i]#outside temprature + 2 degC as
            correction for greenhouse
        GQ[i] = df_weather_GQ['mean'][i] #DLI PAR inside [MJ/m^2]

    else:
        TG[i] = df_weather_TG['mean'][i-365] #outside temprature + 2 degC as
            correction for greenhouse
        GQ[i] = df_weather_GQ['mean'][i-365] #DLI PAR inside [MJ/m^2]
sns.set_theme(context='paper', palette='colorblind', font='Georgia',
    font_scale=3, style="whitegrid")
plt.figure(21, figsize=[16,8])
days = np.linspace(0,365,365)
plt.plot(days, df_weather_TG['mean'][0:365], linewidth = 3)
plt.plot(days, df_weather_TG['2012'][0:365] , days,
    df_weather_TG['2013'][0:365] , days, df_weather_TG['2014'][0:365],
        days, df_weather_TG['2015'][0:365] , days, df_weather_TG['2016'][0:365]
            , days, df_weather_TG['2017'][0:365],
        days, df_weather_TG['2018'][0:365] , days, df_weather_TG['2019'][0:365]
            , days, df_weather_TG['2020'][0:365],
        days, df_weather_TG['2021'][0:365] , lw = 0.3)
#plt.plot(s, DLI_inside_primar)
plt.xlim([0,365])
plt.tick_params(labelrotation=45)
plt.xlabel("Day number (1 = 1 January)")
plt.ylabel("24h Celsius")
plt.legend(["10-year mean"], loc='upper right')
plt.title("Inside Temperature")

plt.figure(22, figsize=[16,8])
days = np.linspace(0,365,365)
plt.plot(days, df_weather_GQ['mean'][0:365], linewidth = 3)
plt.plot(days, df_weather_GQ['2012'][0:365] , days,
    df_weather_GQ['2013'][0:365] , days, df_weather_GQ['2014'][0:365],
        days, df_weather_GQ['2015'][0:365] , days, df_weather_GQ['2016'][0:365]
            , days, df_weather_GQ['2017'][0:365],
        days, df_weather_GQ['2018'][0:365] , days, df_weather_GQ['2019'][0:365]
            , days, df_weather_GQ['2020'][0:365],
        days, df_weather_GQ['2021'][0:365] , lw = 0.3)
#plt.plot(s, DLI_inside_primar)
plt.xlim([0,365])
plt.tick_params(labelrotation=45)
plt.xlabel("Day number (1 = 1 January)")
plt.ylabel("MJ/m^2/day")
plt.legend(["10-year mean"], loc='upper right')
plt.title("Inside DLI PAR")
```

```
if (Base_data == 'Darwin') or (Base_data == 'Melbourne'): #load 10 year average
    curve
   if Base_data == 'Darwin':
      SolarDarwin = pd.read_excel("data_australie.xlsx", sheet_name="solar
          darwin")
      MaxDarwin = pd.read_excel("data_australie.xlsx", sheet_name="max temp
          darwin")
      MinDarwin = pd.read_excel("data_australie.xlsx", sheet_name="min temp
          darwin")
   if Base_data == 'Melbourne':
      SolarDarwin = pd.read_excel("data_australie.xlsx", sheet_name="solar
          melbourne")
      MaxDarwin = pd.read_excel("data_australie.xlsx", sheet_name="max temp
          melbourne")
      MinDarwin = pd.read_excel("data_australie.xlsx", sheet_name="min temp
          melbourne")

   S2012 = SolarDarwin.loc[(SolarDarwin["Year"] == 2012)]["Daily global solar
       exposure (MJ/m*m)"]
   S2013 = SolarDarwin.loc[(SolarDarwin["Year"] == 2013)]["Daily global solar
       exposure (MJ/m*m)"]
   S2014 = SolarDarwin.loc[(SolarDarwin["Year"] == 2014)]["Daily global solar
       exposure (MJ/m*m)"]
   S2015 = SolarDarwin.loc[(SolarDarwin["Year"] == 2015)]["Daily global solar
       exposure (MJ/m*m)"]
   S2016 = SolarDarwin.loc[(SolarDarwin["Year"] == 2016)]["Daily global solar
       exposure (MJ/m*m)"]
   S2017 = SolarDarwin.loc[(SolarDarwin["Year"] == 2017)]["Daily global solar
       exposure (MJ/m*m)"]
   S2018 = SolarDarwin.loc[(SolarDarwin["Year"] == 2018)]["Daily global solar
       exposure (MJ/m*m)"]
   S2019 = SolarDarwin.loc[(SolarDarwin["Year"] == 2019)]["Daily global solar
       exposure (MJ/m*m)"]
   S2020 = SolarDarwin.loc[(SolarDarwin["Year"] == 2020)]["Daily global solar
       exposure (MJ/m*m)"]
   S2021 = SolarDarwin.loc[(SolarDarwin["Year"] == 2021)]["Daily global solar
       exposure (MJ/m*m)"]
   S2012 = pd.Series.reset_index(S2012)
   S2013 = pd.Series.reset_index(S2013)
   S2014 = pd.Series.reset_index(S2014)
   S2015 = pd.Series.reset_index(S2015)
   S2016 = pd.Series.reset_index(S2016)
   S2017 = pd.Series.reset_index(S2017)
   S2018 = pd.Series.reset_index(S2018)
   S2019 = pd.Series.reset_index(S2019)
   S2020 = pd.Series.reset_index(S2020)
   S2021 = pd.Series.reset_index(S2021)

   Max2012 = MaxDarwin.loc[(MaxDarwin["Year"] == 2012)]["Maximum temperature
       (Degree C)"]
   Max2013 = MaxDarwin.loc[(MaxDarwin["Year"] == 2013)]["Maximum temperature
       (Degree C)"]
   Max2014 = MaxDarwin.loc[(MaxDarwin["Year"] == 2014)]["Maximum temperature
       (Degree C)"]
   Max2015 = MaxDarwin.loc[(MaxDarwin["Year"] == 2015)]["Maximum temperature
       (Degree C)"]
   Max2016 = MaxDarwin.loc[(MaxDarwin["Year"] == 2016)]["Maximum temperature
       (Degree C)"]
   Max2017 = MaxDarwin.loc[(MaxDarwin["Year"] == 2017)]["Maximum temperature
       (Degree C)"]
```

```
Max2018 = MaxDarwin.loc[(MaxDarwin["Year"] == 2018)]["Maximum temperature
    (Degree C)"]
Max2019 = MaxDarwin.loc[(MaxDarwin["Year"] == 2019)]["Maximum temperature
    (Degree C)"]
Max2020 = MaxDarwin.loc[(MaxDarwin["Year"] == 2020)]["Maximum temperature
    (Degree C)"]
Max2021 = MaxDarwin.loc[(MaxDarwin["Year"] == 2021)]["Maximum temperature
    (Degree C)"]
Max2012 = pd.Series.reset_index(Max2012)
Max2013 = pd.Series.reset_index(Max2013)
Max2014 = pd.Series.reset_index(Max2014)
Max2015 = pd.Series.reset_index(Max2015)
Max2016 = pd.Series.reset_index(Max2016)
Max2017 = pd.Series.reset_index(Max2017)
Max2018 = pd.Series.reset_index(Max2018)
Max2019 = pd.Series.reset_index(Max2019)
Max2020 = pd.Series.reset_index(Max2020)
Max2021 = pd.Series.reset_index(Max2021)

Min2012 = MinDarwin.loc[(MinDarwin["Year"] == 2012)]["Minimum temperature
    (Degree C)"]
Min2013 = MinDarwin.loc[(MinDarwin["Year"] == 2013)]["Minimum temperature
    (Degree C)"]
Min2014 = MinDarwin.loc[(MinDarwin["Year"] == 2014)]["Minimum temperature
    (Degree C)"]
Min2015 = MinDarwin.loc[(MinDarwin["Year"] == 2015)]["Minimum temperature
    (Degree C)"]
Min2016 = MinDarwin.loc[(MinDarwin["Year"] == 2016)]["Minimum temperature
    (Degree C)"]
Min2017 = MinDarwin.loc[(MinDarwin["Year"] == 2017)]["Minimum temperature
    (Degree C)"]
Min2018 = MinDarwin.loc[(MinDarwin["Year"] == 2018)]["Minimum temperature
    (Degree C)"]
Min2019 = MinDarwin.loc[(MinDarwin["Year"] == 2019)]["Minimum temperature
    (Degree C)"]
Min2020 = MinDarwin.loc[(MinDarwin["Year"] == 2020)]["Minimum temperature
    (Degree C)"]
Min2021 = MinDarwin.loc[(MinDarwin["Year"] == 2021)]["Minimum temperature
    (Degree C)"]
Min2012 = pd.Series.reset_index(Min2012)
Min2013 = pd.Series.reset_index(Min2013)
Min2014 = pd.Series.reset_index(Min2014)
Min2015 = pd.Series.reset_index(Min2015)
Min2016 = pd.Series.reset_index(Min2016)
Min2017 = pd.Series.reset_index(Min2017)
Min2018 = pd.Series.reset_index(Min2018)
Min2019 = pd.Series.reset_index(Min2019)
Min2020 = pd.Series.reset_index(Min2020)
Min2021 = pd.Series.reset_index(Min2021)

T2012 = np.zeros((365))
T2013 = np.zeros((365))
T2014 = np.zeros((365))
T2015 = np.zeros((365))
T2016 = np.zeros((365))
T2017 = np.zeros((365))
T2018 = np.zeros((365))
T2019 = np.zeros((365))
T2020 = np.zeros((365))
T2021 = np.zeros((365))
```

```
Solar2012 = np.zeros((365))
Solar2013 = np.zeros((365))
Solar2014 = np.zeros((365))
Solar2015 = np.zeros((365))
Solar2016 = np.zeros((365))
Solar2017 = np.zeros((365))
Solar2018 = np.zeros((365))
Solar2019 = np.zeros((365))
Solar2020 = np.zeros((365))
Solar2021 = np.zeros((365))

TMean = np.zeros((runtime))
Smean = np.zeros((runtime))

for i in range(runtime):
   if i < 365:
      Solar2012[i] = S2012["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2013[i] = S2013["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2014[i] = S2014["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2015[i] = S2015["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2016[i] = S2016["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2017[i] = S2017["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2018[i] = S2018["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2019[i] = S2012["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2020[i] = S2020["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio
      Solar2021[i] = S2021["Daily global solar exposure (MJ/m*m)"][i] / 10 *
          Transmission_rate * PAR_ratio

      T2012[i] = (Min2012["Minimum temperature (Degree C)"][i] +
          Max2012["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2013[i] = (Min2013["Minimum temperature (Degree C)"][i] +
          Max2013["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2014[i] = (Min2014["Minimum temperature (Degree C)"][i] +
          Max2014["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2015[i] = (Min2015["Minimum temperature (Degree C)"][i] +
          Max2015["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2016[i] = (Min2016["Minimum temperature (Degree C)"][i] +
          Max2016["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2017[i] = (Min2017["Minimum temperature (Degree C)"][i] +
          Max2017["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2018[i] = (Min2018["Minimum temperature (Degree C)"][i] +
          Max2018["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2019[i] = (Min2019["Minimum temperature (Degree C)"][i] +
          Max2019["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2020[i] = (Min2020["Minimum temperature (Degree C)"][i] +
          Max2020["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3
      T2021[i] = (Min2021["Minimum temperature (Degree C)"][i] +
          Max2021["Maximum temperature (Degree C)"][i]) / 2 / 10 + 3

      TMean[i] = np.mean([T2012[i], T2013[i], T2014[i], T2015[i], T2016[i],
          T2017[i], T2018[i], T2019[i], T2020[i], T2021[i]])
      if ((TMean[i] > 10) or (TMean[i] < 40)) == False:
```

```
                TMean[i] = TMean[i-1]
            Smean[i] = np.mean([Solar2012[i], Solar2013[i], Solar2014[i],
                Solar2015[i], Solar2016[i], Solar2017[i], Solar2018[i], Solar2019[i],
                Solar2020[i], Solar2021[i]])
            Smean = np.nan_to_num(Smean)
            if Smean[i] < 0.1:
               if Smean[i-1] < 0.1:
                  Smean[i] = Smean[i - 2]
               else:
                  Smean[i] = Smean[i-1]

         else:
            TMean[i] = TMean[i-365] #outside temprature + 2 degC as correction for
                greenhouse
            Smean[i] = Smean[i-365] #DLI PAR inside [MJ/m^2]
            if Smean[i] < 0.1:
               if Smean[i-1] < 0.1:
                  Smean[i] = Smean[i - 2]
               else:
                  Smean[i] = Smean[i-1]

      sns.set_theme(context='paper', palette='colorblind', font='Georgia',
          font_scale=3, style="whitegrid")
      plt.figure(21, figsize=[16,8])
      days = np.linspace(0,365,365)
      plt.plot(days, Smean[0:365], linewidth = 3)
      plt.plot(days, Solar2012[0:365] , days, Solar2013[0:365] , days,
          Solar2014[0:365],
             days, Solar2015[0:365] , days, Solar2016[0:365] , days, Solar2017[0:365],
             days, Solar2018[0:365] , days, Solar2019[0:365] , days, Solar2020[0:365],
             days, Solar2021[0:365] , lw = 0.3)
      #plt.plot(s, DLI_inside_primar)
      plt.xlim([0,365])
      plt.tick_params(labelrotation=45)
      plt.xlabel("Day number (1 = 1 January)")
      plt.ylabel("MJ/m^2/day")
      plt.legend(["10-year mean"], loc='upper right')
      plt.title("Inside DLI PAR")

      plt.figure(22, figsize=[16,8])
      days = np.linspace(0,365,365)
      plt.plot(days, TMean[0:365], linewidth = 3)
      plt.plot(days, T2012[0:365] , days, T2013[0:365] , days, T2014[0:365],
             days, T2015[0:365] , days, T2016[0:365] , days, T2017[0:365],
             days, T2018[0:365] , days, T2019[0:365] , days, T2020[0:365],
             days, T2021[0:365] , lw = 0.3)
      #plt.plot(s, DLI_inside_primar)
      plt.xlim([0,365])
      plt.tick_params(labelrotation=45)
      plt.xlabel("Day number (1 = 1 January)")
      plt.ylabel("Celsius")
      plt.legend(["10-year mean"], loc='upper right')
      plt.title("Inside temperature")

# ========================================================================
# ====== Determine Light for DLI curve ==================================
# ========================================================================
def DLI_curve(DLI_upper, DLI_lower, Transmission_rate, PAR_ratio):
      # Load curve for Rotterdam [MJ/m^2/day]
      if Base_data == 'Rotterdam':
         DLI_inside = GQ
```

```
if (Base_data == 'Darwin') or (Base_data == 'Melbourne'):
   DLI_inside = Smean

else:
   LightIntensity = [247, 441, 922, 1550, 1777, 1988, 1805, 1478, 1094, 640,
       284, 180,
                 247, 441, 992, 1550, 1777, 1988, 1805, 1478, 1094, 640, 284,
                     180]

   # Interpolate
   Days = np.arange(0, runtime, 1)
   DaysMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31, 28, 31,
       30, 31, 30, 31, 31, 30, 31, 30, 31]
   y = np.zeros((25))
   x = np.zeros((25))
   y[0] = LightIntensity[23]
   j = 0

   for i in range(len(DaysMonth)):
      j += DaysMonth[i]
      y[i + 1] = LightIntensity[i]
      x[i + 1] = j

   f = sp.interpolate.interp1d(x, y, 'quadratic')
   LightIntensity_inter = f(Days) / 100

   DLI_inside = LightIntensity_inter * Transmission_rate * PAR_ratio
DLI = np.zeros((runtime))
Screening_lighting = np.zeros((runtime))
for i in range(runtime):
   if DLI_inside[i] < DLI_lower:
      DLI[i] = DLI_lower
      Screening_lighting[i] = DLI_lower - DLI_inside[i]
   if DLI_inside[i] > DLI_upper:
      DLI[i] = DLI_upper
      Screening_lighting[i] = DLI_upper - DLI_inside[i]
   if (DLI_inside[i] >= DLI_lower) and (DLI_inside[i] <= DLI_upper):
      DLI[i] = DLI_inside[i]
rs = np.random.RandomState(1234566)
#Noise = np.random.normal(0,0.269,runtime) #+/- 1 MJ
Noise = np.random.normal(0,0.649,runtime) #+/- 2 MJ
DLI_inside_noise = np.zeros((runtime))
DLI_noise = np.zeros((runtime))
Screening_lighting_noise = np.zeros((runtime))
for i in range(runtime):
   DLI_inside_noise[i] = DLI_inside[i] + Noise[i]
   Screening_lighting_noise = np.zeros((runtime))
for i in range(runtime):
   if DLI_inside_noise[i] < DLI_lower:
      DLI_noise[i] = DLI_lower
      Screening_lighting_noise[i] = DLI_lower - DLI_inside_noise[i]
   if DLI_inside_noise[i] > DLI_upper:
      DLI_noise[i] = DLI_upper
      Screening_lighting_noise[i] = DLI_upper - DLI_inside_noise[i]
   if (DLI_inside_noise[i] >= DLI_lower) and (DLI_inside_noise[i] <=
       DLI_upper):
      DLI_noise[i] = DLI_inside_noise[i]

return DLI, DLI_inside, DLI_inside_noise, DLI_noise, Screening_lighting,
    Screening_lighting_noise, Noise
```

```
DLI, DLI_inside_primar, DLI_inside_noise, DLI_noise, Screening_lighting,
    Screening_lighting_noise, Noise = DLI_curve(DLI_upper, DLI_lower,
                                                                    Transmission_rate,
                                                                        PAR_ratio)
if Light == 'DLI curve' and Base_data == 'typical values':
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
        font_scale=3, style="whitegrid")
    plt.figure(1, figsize=[16, 8])
    plt.plot(s, DLI_inside_primar, s, Screening_lighting, s, DLI)
    #plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.legend(["Natural DLI inside", "Added / filtered light", "Final DLI
        inside"], loc='upper right')
    plt.title("DLI PAR")


if Light == 'DLI curve + noise' and Base_data == 'typical values':
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
        font_scale=3, style="whitegrid")
    plt.figure(1, figsize=[16, 8])
    plt.plot(s, DLI_inside_noise, s, Screening_lighting_noise, s, DLI_noise)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.legend(["Natural DLI inside", "Added / filtered light", "Final DLI
        inside"], loc='upper right')
    plt.title("DLI PAR")
if (Light != 'DLI curve') and (Light != 'DLI curve + noise'):
    DLI = np.zeros((runtime))
# ============================================================================
# ============ Determine temperature profile =============================
# ============================================================================
def Temperature(Tmin, Tmax, DLI, Temperature_strategy):
    if Temperature_strategy == 'range':
        Scaler = MinMaxScaler((Tmin, Tmax))
        T = Scaler.fit_transform(DLI.reshape(-1, 1))
    if Temperature_strategy == 'range + 1':
        Scaler = MinMaxScaler((Tmin+1, Tmax+1))
        T = Scaler.fit_transform(DLI.reshape(-1, 1))
    if Temperature_strategy == 'range + noise':
        Scaler = MinMaxScaler((Tmin + np.min(Noise), Tmax + np.max(Noise)))
        Tt = Scaler.fit_transform(DLI_inside_noise.reshape(-1, 1))
        #noise = np.random.normal(0,1,runtime)
        #T = np.zeros((runtime))
        #for i in range(runtime):
        #   T[i] = noise[i] + Tt[i]
        T = Tt
    if Temperature_strategy == '20':
        T = np.ones((runtime)) * 20
    if Temperature_strategy == '21':
        T = np.ones((runtime)) * 21
    if Temperature_strategy == '22':
        T = np.ones((runtime)) * 22
    if Base_data == 'Rotterdam':
        T = np.zeros((runtime))
        for i in range(runtime):
```

```
            T[i] = TG[i]
            if T[i] < Tmin:
               T[i] = Tmin
    if (Base_data == 'Darwin') or (Base_data == 'Melbourne'):
       T = np.zeros((runtime))
       for i in range(runtime):
          T[i] = TMean[i]
          if T[i] < Tmin:
             T[i] = Tmin
    return T

T = Temperature(Tmin, Tmax, DLI, Temperature_strategy)
sns.set_theme(context='paper', palette='colorblind', font='Georgia',
    font_scale=3, style="whitegrid")
plt.figure(2, figsize=[16, 8])
plt.plot(s, T)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date")
plt.ylabel("Celsius")
plt.title("Temperature profile")
# Now the expected curves without noise are not needed anymore

if Light == 'DLI curve + noise':
    DLI = DLI_noise
    Screening_lighting = Screening_lighting_noise

# ================================================================================
# ======== Calculate lead time per compartment ==================================
# ================================================================================
max_length = 120
LOST = np.zeros((runtime)) # Length of Stay Total; calculated from day of sowing
for i in range(runtime-max_length):
    FDR = np.zeros((max_length))
    FDS = 0
    for j in range(max_length):
       l = i + j
       FDR = aFDR + np.log(T[l]/20) * 0.02131
       FDS += FDR
       if FDS >= 1:
          LOST[i] = j
          break

LOS1 = np.ones((runtime)) * 4 # For every entrance day length of stay equals 8
    days

LOS2 = np.zeros((runtime))
for i in range(runtime-max_length):
    FDR = np.zeros((max_length))
    FDS = 0
    for j in range(max_length):
       l = int(LOS1[i] + i + j)
       FDR = aFDR + np.log(T[l]/20) * 0.02131
       FDS += FDR
       if FDS >= (dFDS2):
          LOS2[i] = j
          break

LOS3 = np.zeros((runtime)) # initialize
for i in range(runtime-max_length):
    FDR = np.zeros((max_length))
```

```
   FDS = 0
   for j in range(max_length):
      l = int(LOS1[i] + LOS2[i] + i + j)
      FDR = aFDR + np.log(T[l]/20) * 0.02131
      FDS += FDR
      if FDS >= (dFDS3):
         LOS3[i] = j
         break

LOS4a = np.zeros((runtime)) # initialize
for i in range(runtime-max_length):
   FDR = np.zeros((max_length))
   FDS = 0
   for j in range(max_length):
      l = int(LOS1[i] + LOS2[i] + LOS3[i] + i + j)
      FDR = aFDR + np.log(T[l]/20) * 0.02131
      FDS += FDR
      if FDS >= (dFDS4a):
         LOS4a[i] = j
         break

LOS4b = np.zeros((runtime)) # initialize
for i in range(runtime-max_length):
   FDR = np.zeros((max_length))
   FDS = 0
   for j in range(max_length):
      l = int(LOS1[i] + LOS2[i] + LOS3[i] + LOS4a[i] + i + j)
      FDR = aFDR + np.log(T[l]/20) * 0.02131
      FDS += FDR
      if FDS >= (dFDS4b):
         LOS4b[i] = j
         break

LOS5 = np.zeros((runtime)) # initialize
for i in range(runtime-max_length):
   FDR = np.zeros((max_length))
   FDS = 0
   for j in range(max_length):
      l = int(LOS1[i] + LOS2[i] + LOS3[i] + LOS4a[i] + LOS4b[i] + i + j)
      FDR = aFDR + np.log(T[l]/20) * 0.02131
      FDS += FDR
      if FDS >= (dFDS5):
         LOS5[i] = j
         break


LOS = [LOS1, LOS2, LOS3, LOS4a, LOS4b, LOS5]

plt.figure(3, figsize=[16, 8])
plt.plot(s, LOS1, s, LOS2, s, LOS3, s, LOS4a, s, LOS4b, s, LOS5, s, LOST)
plt.xlim([s[0], s[runtime - max_length]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date of onset")
plt.ylabel("Days")
plt.title("Length of stay in a compartment")
plt.legend(["comp 1", "comp 2", "comp 3", "comp 4a", "comp 4b", "comp 5",
    "total"], loc='upper right')

# ========================================================================
# ============ Determine truss density ===================================
# ========================================================================
```

```
#RLIS = Target_Truss_weight / LUE # Required light sum truss
# Take fraction per compartment
LI = np.zeros((number_of_compartment))
for j in range(number_of_compartment):
   LI[j] = Growth_per_stage[j] / LUE

# Calculate total available light sum for a day of onset per compartment
LIA1 = np.zeros((runtime)) #Intitialze available light
LIA2 = np.zeros((runtime)) #Intitialze available light
LIA3 = np.zeros((runtime)) #Intitialze available light
LIA4a = np.zeros((runtime)) #Intitialze available light
LIA4b = np.zeros((runtime)) #Intitialze available light
LIA5 = np.zeros((runtime)) #Intitialze available light
LIAT = np.zeros((runtime)) #Initialize available light

for i in range(runtime-max_length):
   Light_SUM = 0
   for j in range(max_length):
      l = i + j
      Light_SUM += DLI[l]
      if j == LOS1[i]:
         LIA1[i] = Light_SUM
      if j == (int(LOS1[i] + LOS2[i])):
         LIA2[i] = Light_SUM - LIA1[i]
      if j == (int(LOS1[i] + LOS2[i] + LOS3[i])):
         LIA3[i] = Light_SUM - (LIA1[i] + LIA2[i])
      if j == (int(LOS1[i] + LOS2[i] + LOS3[i] + LOS4a[i])):
         LIA4a[i] = Light_SUM - (LIA1[i] + LIA2[i] + LIA3[i])
      if j == (int(LOS1[i] + LOS2[i] + LOS3[i] + LOS4a[i] + LOS4b[i])):
         LIA4b[i] = Light_SUM - (LIA1[i] + LIA2[i] + LIA3[i] + LIA4a[i])
      if j == (int(LOS1[i] + LOS2[i] + LOS3[i] + LOS4a[i] + LOS4b[i] + LOS5[i])):
         LIA5[i] = Light_SUM - (LIA1[i] + LIA2[i] + LIA3[i] + LIA4a[i] + LIA4b[i])
         LIAT[i] = Light_SUM
         break

# Truss density is required available light sum / required light sum
R1 = LIA1 / LI[0]
R2 = LIA2 / LI[1]
R3 = LIA3 / LI[2]
R4a = LIA4a / LI[3]
R4b = LIA4b / LI[4]
R5 = LIA5 / LI[5]
truss_density_cont = [R1, R2, R3, R4a, R4b, R5]

#plt.figure(4, figsize=[16, 8])
#plt.plot(s, R1, s, R2, s, R3, s, R4a, s, R4b, s, R5)
#plt.xlim([s[0], s[runtime - max_length]])
#plt.tick_params(labelrotation=45)
#plt.xlabel("Date of onset")
#plt.ylabel("Trusses/m^2")
#plt.title("Calculated truss density")
#plt.legend(["comp 1", "comp 2", "comp 3", "comp 4a", "comp 4b", "comp 5"], loc =
    'upper right')
#plt.ylim([0,140])
# Convert to discrete step ================================================
Discrete_steps = pd.read_excel("ConfigurationsTrussDensity.xlsx", sheet_name =
    "Blad1", header = 0)

def find_nearest(array, value):
   array = np.asarray(array)
   idx = (np.abs(array - value)).argmin()
```

```
    return array[idx]

truss_density = np.zeros((number_of_compartment, runtime))
number_of_discrete_steps = len(Discrete_steps["TrussDensity"])

number_of_trusses = np.zeros((runtime))
plants_per_gutter = np.zeros((runtime))
Gutter_per_meter = np.zeros((runtime))
for i in range(runtime):
    j = 3 # Find optimal for compartment 4a -> biggest compartment, greatest
        influence
    truss_density[j,i] = find_nearest(Discrete_steps["TrussDensity"],
        truss_density_cont[j][i])
    if truss_density[j,i] in
        Discrete_steps["TrussDensity"][int(number_of_discrete_steps/2):int(number_of_discrete_steps)]
        number_of_trusses[i] = 3
    elif truss_density[j,i] in
        Discrete_steps["TrussDensity"][0:int(number_of_discrete_steps/2)].values:
        number_of_trusses[i] = 2
    else:
        print("Error in truss density selection")

for i in range(runtime):
    if number_of_trusses[i] == 3:
        j = 3
        CropsPerGutterOption =
            Discrete_steps[int(number_of_discrete_steps/2):int(number_of_discrete_steps)]
        CropsPerGutterOption =
            CropsPerGutterOption.loc[(CropsPerGutterOption["TrussDensity"] ==
            truss_density[j,i])]
        plants_per_gutter[i] = np.max(CropsPerGutterOption["CropsGutter"])
    elif number_of_trusses[i] == 2:
        j = 3
        CropsPerGutterOption = Discrete_steps[0:int(number_of_discrete_steps/2)]
        CropsPerGutterOption =
            CropsPerGutterOption.loc[(CropsPerGutterOption["TrussDensity"] ==
            truss_density[j,i])]
        plants_per_gutter[i] = np.max(CropsPerGutterOption["CropsGutter"])

for i in range(runtime):
    GutterMeterOption = Discrete_steps.loc[(Discrete_steps["TrussesPP"] ==
        number_of_trusses[i])]
    GutterMeterOption = GutterMeterOption.loc[(GutterMeterOption["CropsGutter"] ==
        plants_per_gutter[i])]
    for j in [2, 4, 5]:
        truss_density[j, i] = find_nearest(GutterMeterOption["TrussDensity"],
            truss_density_cont[j][i])

for i in range(runtime):
    for j in [0,1]:
        truss_density[j,i] = find_nearest(Discrete_steps["TrussDensity"],
            truss_density_cont[j][i])

# Uncomment this for an optimal truss density per compartment, not limited to
    plants per gutter
for i in range(runtime):
    for j in range(number_of_compartment):
        truss_density[j,i] = find_nearest(Discrete_steps["TrussDensity"],
            truss_density_cont[j][i])

#for i in range(runtime): # uncomment for test with continuous truss density
```

```
#   for j in range(number_of_compartment):
#      truss_density[j,i] = truss_density_cont[j][i]
#      if truss_density[j,i] == 0:
#         truss_density[j,i] = 10

sns.set_theme(context='paper', palette='colorblind', font='Georgia',
    font_scale=3, style="whitegrid")
#plt.figure(5, figsize=[16, 8])
#plt.plot(s, truss_density_cont[:][2], s, truss_density[:][2], s,
    truss_density_cont[:][3], s, truss_density[:][3])
#plt.xlim([s[0], s[runtime - max_length]])
#plt.tick_params(labelrotation=45)
#plt.xlabel("Date of onset")
#plt.ylabel("Trusses/m^2")
#plt.legend(["comp 3 calc", "comp 3 discrete", "comp 4a calc", "comp 4a
    discrete"], loc = 'upper right')
#plt.title("Calculated and actual truss density for comp 3 and comp 4a")

#plt.figure(51, figsize=[16, 8])
#plt.plot(s, truss_density_cont[:][4], s, truss_density[:][4], s,
    truss_density_cont[:][5], s, truss_density[5])
#plt.xlim([s[0], s[runtime - max_length]])
#plt.tick_params(labelrotation=45)
#plt.xlabel("Date of onset")
#plt.ylabel("Trusses/m^2")
#plt.legend(["comp 4b calc", "comp 4b discrete", "comp 5 calc", "comp 5
    discrete"], loc = 'upper right')
#plt.title("Calculated and actual truss density for comp 4b and comp 5")


plt.figure(6, figsize=[16, 8])
plt.plot(s, truss_density[:][2], s, truss_density[:][3], s, truss_density[:][4],
    s, truss_density[:][5])
plt.xlim([s[0], s[runtime - max_length]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date of onset")
plt.ylabel("Trusses/m^2")
plt.legend(["comp 3", 'comp 4a', 'comp 4b', 'comp 5'], loc = 'upper right')
plt.title("Discrete truss density for gutter compartments")
# ============================================================================
# ======== Define growth model ==============================================
# ============================================================================
def Growth_Model(start_day, runtime, max_length, T, tr_m2,
             dFDS1, dFDS2, dFDS3, dFDS4a, dFDS4b, dFDS5):
   FDR = np.zeros((max_length))
   FDS = np.zeros((max_length))
   FFW = np.zeros((max_length))
   TW = np.zeros((max_length))
   ADG = np.zeros((max_length))

   length_of_stay = 0
   for i in range(max_length-1):
      l = int(start_day + i)
      FDS_state = FDS[i]
      z = 0
      if FDS_state > dFDS1:
         z = 1
         if FDS_state > (dFDS1 + dFDS2):
            z = 2
            if FDS_state > (dFDS1 + dFDS2 + dFDS3):
               z = 3
```

```
                if FDS_state > (dFDS1 + dFDS2 + dFDS3 + dFDS4a):
                    z = 4
                    if FDS_state > (dFDS1 + dFDS2 + dFDS3 + dFDS4a + dFDS4b):
                        z = 5

        if l > runtime - 1:
            break
        FDR[i] = aFDR + np.log(T[l]/20) * 0.02131
        FDS[i+1] = FDR[i] + FDS[i]

        ADG[i] = DLI[l] * LUE / tr_m2[int(z)][int(start_day)]
        FFW[i+1] = FFW[i] + ADG[i]

        if FDS[i+1] > 1:
            length_of_stay = i
            break

    return FFW, length_of_stay, FDR, FDS

def Growth_over_run(start_day, runtime, max_length, T, tr_m2,
                dFDS1, dFDS2, dFDS3, dFDS4a, dFDS4b, dFDS5):

    FFW_final = np.zeros((runtime))
    length_of_stay = np.zeros((runtime))
    FFW = {}
    FDR = {}
    FDS = {}

    for i in range(runtime):
        FFW[i], length_of_stay[i], FDR[i], FDS[i] = Growth_Model(i, runtime,
            max_length, T, tr_m2,
                dFDS1, dFDS2, dFDS3, dFDS4a, dFDS4b, dFDS5)
        FFW_final[i] = np.max(FFW[i])
    return FFW_final, FFW, length_of_stay, FDR, FDS

FFW_final, FFW, length_of_stay, FDR, FDS = Growth_over_run(0, runtime,
    max_length, T, truss_density,
                dFDS1, dFDS2, dFDS3, dFDS4a, dFDS4b, dFDS5)

FDR = aFDR + np.log(T/20) * 0.02131
FDR = [FDR, FDR, FDR, FDR, FDR, FDR]
plt.figure(50, figsize=[16,8])
plt.plot(s,FDR[0][:])
plt.title("FDR for temperature curve")
plt.xlabel("Date")
plt.ylabel("-")

#plt.figure(7, figsize=[16, 8])
#plt.plot(s, length_of_stay)
#plt.xlim([s[0], s[runtime - max_length]])
#plt.tick_params(labelrotation=45)
#plt.xlabel("Date of onset")
#plt.ylabel("Days")
#plt.title("Length of stay")

plt.figure(8, figsize=[16, 8])
plt.plot(s, FFW_final)
plt.plot([s[0], s[-1]], [np.average(FFW_final[150:500]),
    np.average(FFW_final[150:500])])
plt.xlim([s[0], s[runtime - max_length -2]])
plt.tick_params(labelrotation=45)
```

```
plt.xlabel("Date of onset")
plt.ylabel("gram")
plt.legend(['Actual values', 'Average'], loc = 'upper right')
plt.title("Truss weight at harvest")
plt.ylim([0, 1000])

plt.figure(9, figsize=[16, 8])
plt.plot(s[150:150+max_length], FFW[150], s[175:175+max_length], FFW[175],
    s[200:200+max_length], FFW[200])
plt.xlim([s[140], s[275]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date")
plt.ylabel("gram")
plt.title("Three Growth curves")
# ================================================================================
# ========== Determine price curve ===============================================
# ================================================================================
def Price_currve(price_curve):
   Days = np.arange(0, runtime, 1)
   if price_curve == "EUprice":
      MonthlyPrices = [154, 140, 143, 136, 95, 97, 106, 109, 122, 136, 134, 145,
          154, 140, 143, 136, 95, 97, 106, 109,
                   122, 136, 134, 145]
      LowerBound = [129, 126, 116, 102, 91, 84, 91, 91, 105, 116, 117, 129, 129,
          126, 116, 102, 91, 84, 91, 91, 105,
                 116,
                 117, 129]
      UpperBound = [187, 164, 167, 153, 99, 114, 132, 144, 142, 154, 149, 165,
          187, 164, 167, 153, 99, 114, 132, 144,
                 142,
                 154, 149, 165]

      DaysMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31, 28, 31,
          30, 31, 30, 31, 31, 30, 31, 30, 31]
      y = np.zeros((25))
      y_upper = np.zeros((25))
      y_lower = np.zeros((25))
      x = np.zeros((25))
      x[0] = 0
      y[0] = MonthlyPrices[23]
      y_lower[0] = LowerBound[23]
      y_upper[0] = UpperBound[23]
      j = 0
      for i in range(len(DaysMonth)):
         j += DaysMonth[i]
         y[i + 1] = MonthlyPrices[i]
         y_upper[i + 1] = UpperBound[i]
         y_lower[i + 1] = LowerBound[i]
         x[i + 1] = j

      f = sp.interpolate.interp1d(x, y, 'cubic')
      f_L = sp.interpolate.interp1d(x, y_lower, 'cubic')
      f_U = sp.interpolate.interp1d(x, y_upper, 'cubic')
      price = f(Days)

   if price_curve == "block":
      price = np.zeros(runtime)
      price[0:int(365 / 5)] = 140
      price[int(365 / 5):int(365 * 2 / 5)] = 120
      price[int(365 * 2 / 5):int(365 * 3 / 5)] = 100
      price[int(365 * 3 / 5):int(365 * 4 / 5)] = 120
```

```
      price[int(365 * 4 / 5):int(365 * 5 / 5)] = 140

      price[int(365 * 5 / 5):int(365 * 6 / 5)] = 140
      price[int(365 * 6 / 5):int(365 * 7 / 5)] = 120
      price[int(365 * 7 / 5):int(365 * 8 / 5)] = 100
      price[int(365 * 8 / 5):int(365 * 9 / 5)] = 120
      price[int(365 * 9 / 5):int(365 * 10 / 5)] = 140

   if price_curve == 'constant':
      price = np.ones(runtime) * 120

   return price
price = Price_currve(price_curve)
plt.figure(10, figsize=[16, 8])
plt.plot(s, price)
# plt.plot(f_U(Days))
# plt.plot(f_L(Days))
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date")
plt.ylabel("€/100 kg")
plt.title("Implemented price curve")
# ============================================================================
# ======= Define planning model ==============================================
# ============================================================================
def Planning_Model(FFW_final, price, number_of_compartment, runtime, LOS,
    Effective_area, Areas, CapacityWeight):
   # ========================================================================
   # Sets, parameters, other inputs
   K = [i for i in range(number_of_compartment)]
   I = [i for i in range(runtime)]

   for k in K:
      for i in I:
         LOS[k][i] = int(LOS[k][i])
         if LOS[k][i] == 0:
            LOS[k][i] = 1 # prevent zero division
   P = price
   # ========================================================================
   # Variables
   m = Model("Production plan")
   X = {} #sowing rate
   A = {} #area
   C = {} #compartment

   for i in I:
      X[i] = m.addVar(vtype=GRB.CONTINUOUS, lb=0)
   for k in K:
      A[k] = m.addVar(vtype=GRB.INTEGER, lb=0)

   comp1_steps = m.addVars(len(Areas[0][:]), vtype=GRB.BINARY)
   comp2_steps = m.addVars(len(Areas[0][:]), vtype=GRB.BINARY)
   comp3_steps = m.addVars(len(Areas[1][:]), vtype=GRB.BINARY)
   comp4a_steps = m.addVars(len(Areas[1][:]), vtype=GRB.BINARY)
   comp4b_steps = m.addVars(len(Areas[1][:]), vtype=GRB.BINARY)
   comp5_steps = m.addVars(len(Areas[1][:]), vtype=GRB.BINARY)

   # ====================================================================
   # Objective function
   m.update()
   m.setObjective(quicksum(X[i] * (FFW_final[i] / 1000) * (P[i] / 100) for i in
```

```
    I))
# m.setObjective(quicksum(X[i] for i in I))
m.modelSense = GRB.MAXIMIZE
m.update()
# ======================================================================
# constraints

# Constraint 1: Overall capacity constraint
m.addConstr(quicksum(A[k] for k in K) <= Effective_area)

# Constraint 2: Capacity per compartment
for i in I[56:502]:
   for k in K:
      m.addConstr(X[i] * LOS[k][i] <= A[k] * (truss_density[k][i] /
          number_of_trusses[i]))


# Constraint 3: Non-negative constraint
for i in I:
   m.addConstr(X[i] >= 0)
for k in K:
   m.addConstr(A[k] >= 0)

# Constraint 4: Add steps for allocating areas
m.addConstr(comp1_steps.sum() == 1)
m.addConstr(comp2_steps.sum() == 1)
m.addConstr(comp3_steps.sum() == 1)
m.addConstr(comp4a_steps.sum() == 1)
m.addConstr(comp4b_steps.sum() == 1)
m.addConstr(comp5_steps.sum() == 1)

for i, val in enumerate(Areas[0][:]):
   m.addConstr((comp1_steps[i] == 1) >> (A[0] == val))
for i, val in enumerate(Areas[0][:]):
   m.addConstr((comp2_steps[i] == 1) >> (A[1] == val))
for i, val in enumerate(Areas[1][:]):
   m.addConstr((comp3_steps[i] == 1) >> (A[2] == val))
for i, val in enumerate(Areas[1][:]):
   m.addConstr((comp4a_steps[i] == 1) >> (A[3] == val))
for i, val in enumerate(Areas[1][:]):
   m.addConstr((comp4b_steps[i] == 1) >> (A[4] == val))
for i, val in enumerate(Areas[1][:]):
   m.addConstr((comp5_steps[i] == 1) >> (A[5] == val))

# Constraint 5: Use an average number at beginning and end to account for
    filling and emptying
Z = Effective_area / CapacityWeight / 2
for i in I:
   if (i > 501 and i <= 600):
      m.addConstr((X[i] == Z[i-365]))
   if i > 600:
      m.addConstr((X[i] == 0))
for i in I:
   if (i > 10 and i <= 55):
      m.addConstr((X[i] == Z[i]))
   if i <= 10:
      m.addConstr((X[i] == 0))
#
    ================================================================================
# Optimization
m.Params.MIPGap = 0
```

```
   m.Params.TimeLimit = 30
   m.optimize()

   H = np.zeros((runtime))
   for i in I:
      H[i] = X[i].X
   Allocation = np.zeros((number_of_compartment))
   for k in K:
      Allocation[k] = A[k].X
   print('Allocated area by Gurobi = ', Allocation)
   print("Possible areas =", Areas[0], Areas[1])
   return H, Allocation
CapacityWeight = 1/R1 * LOS1 + 1/R2 * LOS2 + 1/R3 * LOS3 + 1/R4a * LOS4a + 1/R4b
    * LOS4b + 1/R5 * LOS5
SowingRate_Gurobi, Area = Planning_Model(FFW_final, price, number_of_compartment,
    runtime, LOS, Effective_area, Areas, CapacityWeight)
for i in range(501,730):
   SowingRate_Gurobi[i] = 0
for i in range(0,30):
   SowingRate_Gurobi[i] = 0

LOS1_median = np.median(LOS1[135:500])
LOS2_median = np.median(LOS2[135:500])
LOS3_median = np.median(LOS3[135:500])
LOS4a_median = np.median(LOS4a[135:500])
LOS4b_median = np.median(LOS4b[135:500])
LOS5_median = np.median(LOS5[135:500])

A1_median = 1/np.median(R1[135:500])
A2_median = 1/np.median(R2[135:500])
A3_median = 1/np.median(R3[135:500])
A4a_median = 1/np.median(R4a[135:500])
A4b_median = 1/np.median(R4b[135:500])
A5_median = 1/np.median(R5[135:500])

W1 = LOS1_median * A1_median
W2 = LOS2_median * A2_median
W3 = LOS3_median * A3_median
W4a = LOS4a_median * A4a_median
W4b = LOS4b_median * A4b_median
W5 = LOS5_median * A5_median
WT = W1 + W2 + W3 + W4a + W4b + W5

A1_p = W1 / WT * Effective_area
A2_p = W2 / WT * Effective_area
A3_p = W3 / WT * Effective_area
A4b_p = W4a / WT * Effective_area
A4a_p = W4b / WT * Effective_area
A5_p = W5 / WT * Effective_area

def find_nearest(array, value):
   array = np.asarray(array)
   idx = (np.abs(array - value)).argmin()
   offset = value - array[idx]
   return array[idx], offset

A1 = find_nearest(Areas[0], A1_p)
A2 = find_nearest(Areas[0], A2_p)
A3 = find_nearest(Areas[1], A3_p)
A4a = find_nearest(Areas[1], A4a_p)
A4b = find_nearest(Areas[1], A4b_p)
```

```
A5 = find_nearest(Areas[1], A5_p)

# Uncomment for an area selection without using gurobi
#Area = [A1[0], A2[0], A3[0], A4a[0], A4b[0], A5[0]]
#Area_offset = [A1[1], A2[1], A3[1], A4a[1], A4b[1], A5[1]]
#print(Area_offset, 'Offset continuous vs discrete by guessed area (manually)')
#print(Area, 'Manually guessed optimal area (manually)')
#while np.sum(Area) > Effective_area:
#   max_offset = np.min(Area_offset)
#   for i in range(len(Area_offset)):
#      if Area_offset[i] == max_offset:
#         Reduced = i
#         break
#   if Reduced < 2:
#      Area[Reduced] -= Areas[0][0]
#      Area_offset[Reduced] += Areas[0][0]
#   else:
#      Area[Reduced] -= Areas[1][0]
#      Area_offset[Reduced] += Areas[1][0]
#   print(Area, "Area manually adjusted in loop (reduced)")

#while np.sum(Area) < (Effective_area - Areas[0][0]):
#   max_offset = np.max([Area_offset[0], Area_offset[1]])
#   for i in range(0, 1):
#      if Area_offset[i] == max_offset:
#         Increased = i
#         break
#   Area[Increased] += Areas[0][0]
#   Area_offset[Increased] -= Areas[0][0]
#   print(Area, "Area manually adjusted in loop (increase tray rows)")

CapacityWeight = 1/R1 * LOS1 + 1/R2 * LOS2 + 1/R3 * LOS3 + 1/R4a * LOS4a + 1/R4b
    * LOS4b + 1/R5 * LOS5
SowingRate = Effective_area / CapacityWeight
SowingRate = SowingRate_Gurobi #uncomment if manual is used
for i in range(1, len(SowingRate)):
   if CapacityWeight[i] == CapacityWeight[i-1]:
      dif = 1.1
   else:
      dif = (CapacityWeight[i] - CapacityWeight[i-1]) / CapacityWeight[i-1] *
          Factor_to_diff_sowing_rate * 20 + 1
   if dif > 4:
      dif = 1.05
   dif = np.nan_to_num(dif)
   if dif < 1:
      dif = 1

   SowingRate[i] = SowingRate[i] * dif
   #if SowingRate[i] < 19000:
   #   SowingRate[i] = 19000

#b, a = sp.signal.butter(2,0.03,'low')
#SowingRate = sp.signal.filtfilt(b,a, SowingRate)
# Inspected optimized -> close to optimal below: (not exact as gurobi)
#Area = [540, 2160, 5760, 12960, 12960, 8640]
#print("Advised optimal area (djusted areas by hand): ", Area)

plt.figure(11, figsize=[16, 8])
plt.plot(s, SowingRate * trusses_per_plant)#, s, SowingRate_Gurobi *
    trusses_per_plant)
# plt.plot(f_U(Days))
```

```
# plt.plot(f_L(Days))
plt.xlim([s[0], s[runtime - max_length]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date of onset")
plt.ylabel("Trusses")
plt.title("Trusses sowed")
plt.xlim([s[135], s[500]])
#plt.legend(["Hand calculated", "Gurobi"], loc = 'upper right')
#SowingRate = SowingRate/trusses_per_plant if manual is used
SowingRate = SowingRate #if gurobi is used
for i in range(501,730):
   SowingRate[i] = 0
#SowingRate = SowingRate_Gurobi
if len(Add_row_to_compartment) > 0:
   for i in range(len(Add_row_to_compartment)):
      if Add_row_to_compartment[i] == 'C1':
         Area[0] += Areas_trays[0]
      if Add_row_to_compartment[i] == 'C2':
         Area[1] += Areas_trays[0]
      if Add_row_to_compartment[i] == 'C3':
         Area[2] += Areas_gutters[0]
      if Add_row_to_compartment[i] == 'C4a':
         Area[3] += Areas_gutters[0]
      if Add_row_to_compartment[i] == 'C4b':
         Area[4] += Areas_gutters[0]
      if Add_row_to_compartment[i] == 'C5':
         Area[5] += Areas_gutters[0]

if len(Remove_row_from_compartment) > 0:
   for i in range(len(Remove_row_from_compartment)):
      if Remove_row_from_compartment[i] == 'C1':
         Area[0] -= Areas_trays[0]
      if Remove_row_from_compartment[i] == 'C2':
         Area[1] -= Areas_trays[0]
      if Remove_row_from_compartment[i] == 'C3':
         Area[2] -= Areas_gutters[0]
      if Remove_row_from_compartment[i] == 'C4a':
         Area[3] -= Areas_gutters[0]
      if Remove_row_from_compartment[i] == 'C4b':
         Area[4] -= Areas_gutters[0]
      if Remove_row_from_compartment[i] == 'C5':
         Area[5] -= Areas_gutters[0]
total_area_used = sum(Area)
if total_area_used > Effective_area:
   print('WARNING')
   print('Total area used exceeds available area')
#print('Area allocation', Area)
#Area = [540, 2160, 5760, 12960, 12960, 8640]
# ======================================================================
# ========== Plot the compartments =====================================
# ======================================================================
def Plot_floorplan(Length, Width, Crop_handling_area, Row_length,
    Transport_system_width, orientation, no_crop_handling):
   fig, ax = plt.subplots(figsize=[16, 8], tight_layout=True)
   x_axis = np.arange(0, Length + 1, 1)
   y_axis = np.arange(0, Width + 1, 1)

   Crop_handling_x = int(Crop_handling_area / Width)
   Crop_handling_y = Width

   plt.plot(x_axis, np.zeros(len(x_axis)), color='black')
```

```
plt.plot(np.zeros(len(y_axis)), y_axis, color='black')
plt.plot(x_axis, np.ones(len(x_axis)) * Width, color='black')
plt.plot(np.ones(len(y_axis)) * Length, y_axis, color='black')

if orientation == 'short rows':

   if no_crop_handling == 'single':
      length, width = int(Length - Crop_handling_x), Transport_system_width
      x_loc, y_loc = 0, 0
      plt.gca().add_patch(
         patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
             edgecolor='grey', facecolor='grey',
                       label="Transport system"))

      length, width = int(Length - Crop_handling_x), Transport_system_width
      x_loc, y_loc = 0, Width - Transport_system_width
      plt.gca().add_patch(
         patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
             edgecolor='grey', facecolor='grey'))

   if no_crop_handling == 'double':
      length, width = int(Length - Crop_handling_x), Transport_system_width
      x_loc, y_loc = Crop_handling_x / 2, 0
      plt.gca().add_patch(
         patches.Rectangle([x_loc, y_loc],length, width, linewidth=1,
             edgecolor='grey', facecolor='grey',
                       label="Transport system"))

      length, width = int(Length - Crop_handling_x), Transport_system_width
      x_loc, y_loc = Crop_handling_x / 2, Width - Transport_system_width
      plt.gca().add_patch(
         patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
             edgecolor='grey', facecolor='grey'))

   length, width = Area[0] / Row_length, Row_length
   x_loc, y_loc = x_loc, Transport_system_width
   plt.gca().add_patch(
      patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
          edgecolor='rosybrown', facecolor='rosybrown',
                    label='compartment 1'))

   x_loc, y_loc = x_loc + length, Transport_system_width
   length, width = Area[1] / Row_length, Row_length
   plt.gca().add_patch(
      patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
          edgecolor='lightcoral', facecolor='lightcoral',
                    label='compartment 2'))

   x_loc, y_loc = x_loc + length, Transport_system_width
   length, width = Area[2] / Row_length, Row_length
   plt.gca().add_patch(
      patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
          edgecolor='firebrick', facecolor='firebrick',
                    label='compartment 3'))

   x_loc, y_loc = x_loc + length, Transport_system_width
   length, width = Area[3] / Row_length, Row_length
   plt.gca().add_patch(
      patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
          edgecolor='lightsalmon', facecolor='lightsalmon',
                    label='compartment 4a'))
```

```
        x_loc, y_loc = x_loc + length, Transport_system_width
        length, width = Area[4] / Row_length, Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='tomato', facecolor='tomato',
                         label='compartment 4b'))

        x_loc, y_loc = x_loc + length, Transport_system_width
        length, width = Area[5] / Row_length, Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='red', facecolor='red',
                         label='compartment 5'))

        if no_crop_handling == 'single':
           x_loc, y_loc = x_loc + length, 0
           length, width = Crop_handling_x, Crop_handling_y
           plt.gca().add_patch(
              patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
                  edgecolor='blue', facecolor='blue',
                            label="Crop handling area"))

        if no_crop_handling == 'double':
           x_loc, y_loc = x_loc + length, 0
           length, width = Crop_handling_x / 2, Crop_handling_y
           plt.gca().add_patch(
              patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
                  edgecolor='blue', facecolor='blue',
                            label="Crop handling area"))

           x_loc, y_loc = 0, 0
           length, width = Crop_handling_x / 2, Crop_handling_y
           plt.gca().add_patch(
              patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
                  edgecolor='blue', facecolor='blue'))

    if orientation == 'long rows':
        Crop_handling_x = Length
        Crop_handling_y = Crop_handling_area / Length

        x_loc, y_loc = Transport_system_width, 0
        length, width = Row_length, Area[0] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='rosybrown', facecolor='rosybrown',
                         label='compartment 1'))

        x_loc, y_loc = Transport_system_width, y_loc + width
        length, width = Row_length, Area[1] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='lightcoral', facecolor='lightcoral',
                         label='compartment 2'))

        x_loc, y_loc = Transport_system_width, y_loc + width
        length, width = Row_length, Area[2] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='firebrick', facecolor='firebrick',
                         label='compartment 3'))
```

```
        x_loc, y_loc = Transport_system_width, y_loc + width
        length, width = Row_length, Area[3] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='lightsalmon', facecolor='lightsalmon',
                         label='compartment 4a'))

        x_loc, y_loc = Transport_system_width, y_loc + width
        length, width = Row_length, Area[4] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='tomato', facecolor='tomato',
                         label='compartment 4b'))


        x_loc, y_loc = Transport_system_width, y_loc + width
        length, width = Row_length, Area[5] / Row_length
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='red', facecolor='red',
                         label='compartment 5'))

        x_loc, y_loc = 0, y_loc + width
        length, width = Crop_handling_x, Crop_handling_y
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='blue', facecolor='blue',
                         label="Crop handling area"))
        y_loc_fix = y_loc
        x_loc, y_loc = Row_length + Transport_system_width, 0
        length, width = Transport_system_width, Width - (Width - y_loc_fix)
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='grey', facecolor='grey'))

        length, width = Transport_system_width, Width - (Width - y_loc_fix)
        x_loc, y_loc = 0, 0
        plt.gca().add_patch(
           patches.Rectangle([x_loc, y_loc], length, width, linewidth=1,
               edgecolor='grey', facecolor='grey',
                         label="Transport system"))

    plt.legend(loc='right', bbox_to_anchor=(1.35, 0.5))
    plt.xlabel("[m]")
    plt.ylabel("[m]")
    plt.grid()

    return fig, ax
fig, ax = Plot_floorplan(Length, Width, Crop_handling_area, Row_length,
     Transport_system_width, orientation, no_crop_handling)
# ================================================================================
# ========== Take other needed inputs from planning model ========================
# ================================================================================
# First convert the truss rate back to a sowing rate

# ===

ProductionRate = np.zeros((runtime))
for i in range(runtime):
    if i < 600:
```

```
      ProductionRate[int(i + LOST[i])] = SowingRate[i]

for i in range(runtime):
   if i > 1:
      if ProductionRate[i] < 100:
         ProductionRate[i] = ProductionRate[i - 1]

Harvested_weight = np.zeros((runtime))
Harvested_weight_cum = np.zeros((runtime))
for i in range(runtime):
   Harvested_weight[i] = ProductionRate[i] * FFW_final[i]
   if i >= 135:
      Harvested_weight_cum[i] += Harvested_weight[i] + Harvested_weight_cum[i - 1]

Difference_harvest_sowed = np.zeros((runtime))
for i in range(runtime):
   Difference_harvest_sowed[i] = np.sum(SowingRate[0:i] - ProductionRate[0:i])

SowingRateTray = np.zeros((runtime))
Transplant1Rate = np.zeros((runtime))
Transplant2Rate = np.zeros((runtime))
MonitoringRate = np.zeros((runtime))
TreatmentRate = np.zeros((runtime))
HarvestedRate = np.zeros((runtime))
SortedRate = np.zeros((runtime))
TransportTrayRateGRCR = np.zeros((runtime))
TransportTrayRateCRGR = np.zeros((runtime))
TransportGutterRateGRCR = np.zeros((runtime))
TransportGutterRateCRGR = np.zeros((runtime))

for i in range(runtime): # Assumption: no loss of crops
   if i < 600:
      SowingRateTray[i] += SowingRate[i] / Tray1Capacity
      Transplant1Rate[int(i + LOS[0][i])] += SowingRate[i] / Tray1Capacity
      Transplant2Rate[int(i + LOS[0][i] + LOS[1][i])] += SowingRate[i] /
          Tray2Capacity
      MonitoringRate[int(i + LOS[0][i] + LOS[1][i] + LOS[2][i])] += SowingRate[i]
          / GutterCapacity
      MonitoringRate[int(i + LOS[0][i] + LOS[1][i] + LOS[2][i] + LOS[3][i] +
          LOS[4][i])] += SowingRate[i] / GutterCapacity
for i in range(runtime):
   TreatmentRate[i] = MonitoringRate[i] * 0.2
   HarvestedRate[i] = ProductionRate[i] / GutterCapacity
   SortedRate[i] = HarvestedRate[i] * GutterCapacity * number_of_trusses[i]
   TransportTrayRateCRGR[i] = SowingRateTray[i] + Transplant1Rate[i]
   TransportTrayRateGRCR[i] = Transplant1Rate[i] / 2 + Transplant2Rate[i] / 2
   TransportGutterRateCRGR[i] = Transplant2Rate[i] + MonitoringRate[i]
   TransportGutterRateGRCR[i] = MonitoringRate[i] + HarvestedRate[i]

print(”Advised requirements”)
print()
print(”Area per compartment”, Area)
print()
print(”Trays transported to greenhouse”,
    np.average(TransportTrayRateCRGR[135:500]),
     np.max(TransportTrayRateCRGR[135:500]),
         np.min(TransportTrayRateCRGR[135:500]))
print(”Trays transport to crop handing”,
    np.average(TransportTrayRateGRCR[135:500]),
     np.max(TransportTrayRateGRCR[135:500]),
         np.min(TransportTrayRateGRCR[135:500]))
```

```
print("Gutters transported to greenhouse",
    np.average(TransportGutterRateCRGR[135:500]),
     np.max(TransportGutterRateCRGR[135:500]),
         np.min(TransportGutterRateCRGR[135:500]))
print("Gutters transported to crop handing",
    np.average(TransportGutterRateGRCR[135:500]),
     np.max(TransportGutterRateGRCR[135:500]),
         np.min(TransportGutterRateGRCR[135:500]))
print("Sowed rate", np.average(SowingRate[135:500]), np.max(SowingRate[135:500]),
     np.min(SowingRate[135:500]))
print("Transplant step 1", np.average(Transplant1Rate[135:500]),
    np.max(Transplant1Rate[135:500]),
     np.min(Transplant1Rate[135:500]))
print("Transplant step 2", np.average(Transplant2Rate[135:500]),
    np.max(Transplant2Rate[135:500]),
     np.min(Transplant2Rate[135:500]))
print("Monitored", np.average(MonitoringRate[135:500]),
    np.max(MonitoringRate[135:500]), np.min(MonitoringRate[135:500]))
print("Special treatment", np.average(TreatmentRate[135:500]),
    np.max(TreatmentRate[135:500]),
     np.min(TreatmentRate[135:500]))
print("Harvested gutters", np.average(HarvestedRate[135:500]),
    np.max(HarvestedRate[135:500]),
     np.min(HarvestedRate[135:500]))
print("Number of tomatoes sorted", np.average(SortedRate[135:500]),
    np.max(SortedRate[135:500]),
     np.min(SortedRate[135:500]))
print("Average number of crops present",
    np.average(Difference_harvest_sowed[135:500]))

# ======================================================================
# ==== Determine transport times and capacities=====================
# ======================================================================

# Capacity per cultivation medium
Tray1Capacity = 8
Tray2Capacity = 4
GutterCapacity = 16

# Throughputs
working_hours_day = 8
Sowing_capacity = 176 * working_hours_day #trays/day
Transplant1_capacity = 5600 / Tray2Capacity * working_hours_day #trays/day
Transplant2_capacity = 5600 / GutterCapacity * working_hours_day#trays/day
Monitoring_capacity = 360 * working_hours_day #gutters/day
Harvesting_capacity = 8 * 6* 360 / trusses_per_plant / GutterCapacity *
    working_hours_day#gutters/day
Sorting_capacity = 12000 * working_hours_day #trusses/day

# Handlingtimes
Sowing_time = 1/Sowing_capacity / 1
Transplant1_time = 1/Transplant1_capacity / 1
Transplant2_time = 1/Transplant2_capacity / 1
Monitoring_time = 1/Monitoring_capacity / 1
Harvesting_time = 1/Harvesting_capacity / 1
Sorting_time = 1/Sorting_capacity / 1

UnloadTime = 1 / 24 / 60 / 60 / Number_of_Transport_Lanes
LoadTime = 1 / 24 / 60 / 60 / Number_of_Transport_Lanes

def TransportTime(StartLocation, TargetLocation):
```

```
    Acceleration = 0.5 # m/s^2
    Vmax = 3 # m/s

    # Possible Locations:
    # GreenhouseS1
    # GreenhouseS2
    # GreenhouseS3
    # GreenhouseS4
    # GreenhouseS5

    # Assume the simple case where only the movements from left to right are of
        importance
    # If startlocation is one of the greenhouse sections, always crop handling is
        the next destination
    # If startLocation is crop handling, one of the greenhouse sections is the
        destination
    if no_crop_handling == 'single':
        if (StartLocation == "GreenhouseS5") or (TargetLocation == "GreenhouseS5"):
            distance = 50
        if (StartLocation == "GreenhouseS4a") or (TargetLocation ==
            "GreenhouseS4a"):
            distance = 68
        if (StartLocation == "GreenhouseS4b") or (TargetLocation ==
            "GreenhouseS4b"):
            distance = 88
        if (StartLocation == "GreenhouseS3") or (TargetLocation == "GreenhouseS3"):
            distance = 125
        if (StartLocation == "GreenhouseS2") or (TargetLocation == "GreenhouseS2"):
            distance = 197
        if (StartLocation == "GreenhouseS1") or (TargetLocation == "GreenhouseS1"):
            distance = 237
        if (StartLocation == "Internal") or (TargetLocation == "Internal"):
            distance = 25
    elif no_crop_handling == 'double':
        if (StartLocation == "GreenhouseS5") or (TargetLocation == "GreenhouseS5"):
            distance = 62
        if (StartLocation == "GreenhouseS4a") or (TargetLocation ==
            "GreenhouseS4a"):
            distance = 136
        if (StartLocation == "GreenhouseS4b") or (TargetLocation ==
            "GreenhouseS4b"):
            distance = 134
        if (StartLocation == "GreenhouseS3") or (TargetLocation == "GreenhouseS3"):
            distance = 68
        if (StartLocation == "GreenhouseS2") or (TargetLocation == "GreenhouseS2"):
            distance = 42
        if (StartLocation == "GreenhouseS1") or (TargetLocation == "GreenhouseS1"):
            distance = 32
        if (StartLocation == "Internal") or (TargetLocation == "Internal"):
            distance = 25

    Acceleration_time = Vmax / Acceleration
    Acceleration_distance = Vmax / 2 * Acceleration_time
    Vmax_distance = distance - 2 * Acceleration_distance
    Vmax_time = Vmax_distance / Vmax

    DriveTime = (Vmax_time + Acceleration_time) / 24 / 3600 /
        Number_of_Transport_Lanes# convert to days
    return DriveTime, distance

DriveTime = np.zeros((number_of_compartment+1))
```

```
distance = np.zeros((number_of_compartment+1))
DriveTime[0], distance[0] = TransportTime("CropHandling", "GreenhouseS1")
DriveTime[1], distance[1] = TransportTime("CropHandling", "GreenhouseS2")
DriveTime[2], distance[2] = TransportTime("CropHandling", "GreenhouseS3")
DriveTime[3], distance[3] = TransportTime("CropHandling", "GreenhouseS4a")
DriveTime[4], distance[4] = TransportTime("CropHandling", "GreenhouseS4b")
DriveTime[5], distance[5] = TransportTime("CropHandling", "GreenhouseS5")
DriveTime[6], distance[6] = TransportTime("Internal", "Internal")

# ========================================================================
# ======= import the representative year if needed ======================
# ========================================================================
if Base_data == 'Rotterdam':
   # Select relevant years
   if year == '10-year mean':
      T[0:365] = df_weather_TG['mean'][0:365]
      T[365:730] = df_weather_TG['mean'][0:365]

      Q = np.zeros((runtime))
      Q[0:365] = df_weather_GQ['mean'][0:365]
      Q[365:730] = df_weather_GQ['mean'][0:365]

      RefDLI = np.zeros((runtime))
      RefT = np.zeros((runtime))
      RefT[0:365] = df_weather_TG['mean'][0:365]
      RefT[365:730] = df_weather_TG['mean'][0:365]
      RefDLI = DLI
      Screening_lighting = np.zeros((runtime))

      for i in range(runtime):
         if Q[i] < DLI_lower:
            Screening_lighting[i] = DLI_lower - Q[i]
            Q[i] = DLI_lower
         if Q[i] > DLI_upper:
            Screening_lighting[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

      for i in range(runtime):
         if T[i] < Tmin:
            T[i] = Tmin
         if RefT[i] < Tmin:
            RefT[i] = Tmin

      DLI = Q

      df = pd.DataFrame(T)
      RollingTG = df.rolling(14).mean()
      df = pd.DataFrame(DLI)
      RollingGQ = df.rolling(14).mean()

      s = pd.date_range('2018-01-01', periods=runtime, freq='D').to_series()
      sns.set_theme(context='paper', palette='colorblind', font='Georgia',
          font_scale=3, style="whitegrid")
      plt.figure(81, figsize=[16, 8])
      plt.plot(s, RefT, lw = 1)
      plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
      #plt.plot(s, T, lw = 0.3)
      plt.xlim([s[0], s[runtime - 1]])
      plt.tick_params(labelrotation=45)
      plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
          (inside)"], loc = 'upper right')
```

```
plt.xlabel(”Date”)
plt.ylabel(”Celsius”)
plt.title(”Temperature profile”)

plt.figure(82, figsize=[16, 8])
plt.plot(s, RefDLI, lw = 1)
plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
plt.plot(s, Screening_lighting, lw = 0.3)
# plt.plot(s, DLI_inside_primar)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel(”Date”)
plt.ylabel(”MJ/m^2/day”)
plt.legend([”Multi-year average final DLI inside”, ”Moving mean 14 days
    (inside)”, ”Applied filtering or lighting”],
        loc=’lower right’)
plt.title(”DLI PAR”)

T5yearplot = np.zeros((365*6))
DLI5yearplot = np.zeros((365*6))
RefT5yearplot = np.zeros((365*6))
RefDLI5yearplot = np.zeros((365*6))
T5yearplot[0:365] = df_weather_TG[’2016’][0:365]
T5yearplot[365:730] = df_weather_TG[’2017’][0:365]
T5yearplot[730:1095] = df_weather_TG[’2018’][0:365]
T5yearplot[1095:1460] = df_weather_TG[’2019’][0:365]
T5yearplot[1460:1825] = df_weather_TG[’2020’][0:365]
T5yearplot[1825:2190] = df_weather_TG[’2021’][0:365]
DLI5yearplot[0:365] = df_weather_GQ[’2016’][0:365]
DLI5yearplot[365:730] = df_weather_GQ[’2017’][0:365]
DLI5yearplot[730:1095] = df_weather_GQ[’2018’][0:365]
DLI5yearplot[1095:1460] = df_weather_GQ[’2019’][0:365]
DLI5yearplot[1460:1825] = df_weather_GQ[’2020’][0:365]
DLI5yearplot[1825:2190] = df_weather_GQ[’2021’][0:365]
RefT5yearplot[0:365] = df_weather_TG[’mean’][0:365]
RefT5yearplot[365:730] = df_weather_TG[’mean’][0:365]
RefT5yearplot[730:1095] = df_weather_TG[’mean’][0:365]
RefT5yearplot[1095:1460] = df_weather_TG[’mean’][0:365]
RefT5yearplot[1460:1825] = df_weather_TG[’mean’][0:365]
RefT5yearplot[1825:2190] = df_weather_TG[’mean’][0:365]
RefDLI5yearplot[0:365] = df_weather_GQ[’mean’][0:365]
RefDLI5yearplot[365:730] = df_weather_GQ[’mean’][0:365]
RefDLI5yearplot[730:1095] = df_weather_GQ[’mean’][0:365]
RefDLI5yearplot[1095:1460] = df_weather_GQ[’mean’][0:365]
RefDLI5yearplot[1460:1825] = df_weather_GQ[’mean’][0:365]
RefDLI5yearplot[1825:2190] = df_weather_GQ[’mean’][0:365]

s = pd.date_range(’2016-01-01’, periods=2190, freq=’D’).to_series()
sns.set_theme(context=’paper’, palette=’colorblind’, font=’Georgia’,
    font_scale=3, style=”whitegrid”)
plt.figure(41, figsize=[16, 4])
plt.plot(s, T5yearplot, lw = 1)
plt.plot(s, RefT5yearplot, lw = 1)
#plt.plot(s, T, lw = 0.3)
plt.xlim([s[0], s[2189]])
plt.tick_params(labelrotation=45)
plt.legend([”Daily data”, ”10-year mean”], loc = ’upper right’)
plt.xlabel(”Date”)
plt.ylabel(”Celsius”)
plt.title(”Inside temperature”)
```

```
        plt.figure(42, figsize=[16, 4])
        plt.plot(s, DLI5yearplot, lw = 1)
        plt.plot(s, RefDLI5yearplot, lw = 1)
        # plt.plot(s, DLI_inside_primar)
        plt.xlim([s[0], s[2189]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Daily data", "10-year mean"], loc='upper right')
        plt.xlabel("Date")
        plt.ylabel("MJ/m^2/day")
        plt.title("Inside DLI PAR")
        plt.show()

    if year == "2016-2017":
        T[0:365] = df_weather_TG['2016'][0:365]
        T[365:730] = df_weather_TG['2017'][0:365]

        Q = np.zeros((runtime))
        Q[0:365] = df_weather_GQ['2016'][0:365]
        Q[365:730] = df_weather_GQ['2017'][0:365]

        RefDLI = np.zeros((runtime))
        RefT = np.zeros((runtime))
        RefT[0:365] = df_weather_TG['mean'][0:365]
        RefT[365:730] = df_weather_TG['mean'][0:365]
        RefDLI = DLI
        Screening_lighting = np.zeros((runtime))

        for i in range(runtime):
            if Q[i] < DLI_lower:
                Screening_lighting[i] = DLI_lower - Q[i]
                Q[i] = DLI_lower
            if Q[i] > DLI_upper:
                Screening_lighting[i] = DLI_upper - Q[i]
                Q[i] = DLI_upper

        for i in range(runtime):
            if T[i] < Tmin:
                T[i] = Tmin
            if RefT[i] < Tmin:
                RefT[i] = Tmin

        DLI = Q

        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2016-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
        plt.plot(s, RefT, lw = 1)
        plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
        plt.plot(s, T, lw = 0.3)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
            (inside)", "Daily values"], loc = 'upper right')
        plt.xlabel("Date")
        plt.ylabel("Celsius")
```

```
    plt.title("Temperature profile")

    plt.figure(82, figsize=[16, 8])
    plt.plot(s, RefDLI, lw = 1)
    plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
    plt.plot(s, DLI, s, Screening_lighting, lw = 0.3)
    # plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
        (inside)", "Daily final inside values", "Applied filtering or lighting"],
            loc='lower right')
    plt.title("DLI PAR")

if year == "2017-2018":
    T[0:365] = df_weather_TG['2017'][0:365]
    T[365:730] = df_weather_TG['2018'][0:365]

    Q = np.zeros((runtime))
    Q[0:365] = df_weather_GQ['2017'][0:365]
    Q[365:730] = df_weather_GQ['2018'][0:365]

    RefDLI = np.zeros((runtime))
    RefT = np.zeros((runtime))
    RefT[0:365] = df_weather_TG['mean'][0:365]
    RefT[365:730] = df_weather_TG['mean'][0:365]
    RefDLI = DLI
    Screening_lighting = np.zeros((runtime))

    for i in range(runtime):
        if Q[i] < DLI_lower:
            Screening_lighting[i] = DLI_lower - Q[i]
            Q[i] = DLI_lower
        if Q[i] > DLI_upper:
            Screening_lighting[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

    for i in range(runtime):
        if T[i] < Tmin:
            T[i] = Tmin
        if RefT[i] < Tmin:
            RefT[i] = Tmin

    DLI = Q

    df = pd.DataFrame(T)
    RollingTG = df.rolling(14).mean()
    df = pd.DataFrame(DLI)
    RollingGQ = df.rolling(14).mean()

    s = pd.date_range('2017-01-01', periods=runtime, freq='D').to_series()
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
        font_scale=3, style="whitegrid")
    plt.figure(81, figsize=[16, 8])
    plt.plot(s, RefT, lw = 1)
    plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
    plt.plot(s, T, lw = 0.3)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
```

```
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
            (inside)", "Daily values"], loc = 'upper right')
        plt.xlabel("Date")
        plt.ylabel("Celsius")
        plt.title("Temperature profile")

        plt.figure(82, figsize=[16, 8])
        plt.plot(s, RefDLI, lw = 1)
        plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
        plt.plot(s, DLI, s, Screening_lighting, lw = 0.3)
        # plt.plot(s, DLI_inside_primar)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.xlabel("Date")
        plt.ylabel("MJ/m^2/day")
        plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
            (inside)", "Daily final inside values", "Applied filtering or lighting"],
                loc='lower right')
        plt.title("DLI PAR")

    if year == "2018-2019":

        T[0:365] = df_weather_TG['2018'][0:365]
        T[365:730] = df_weather_TG['2019'][0:365]

        Q = np.zeros((runtime))
        Q[0:365] = df_weather_GQ['2018'][0:365]
        Q[365:730] = df_weather_GQ['2019'][0:365]

        RefDLI = np.zeros((runtime))
        RefT = np.zeros((runtime))
        RefT[0:365] = df_weather_TG['mean'][0:365]
        RefT[365:730] = df_weather_TG['mean'][0:365]
        RefDLI = DLI
        Screening_lighting20182019 = np.zeros((runtime))

        for i in range(runtime):
            if Q[i] < DLI_lower:
                Screening_lighting20182019[i] = DLI_lower - Q[i]
                Q[i] = DLI_lower
            if Q[i] > DLI_upper:
                Screening_lighting20182019[i] = DLI_upper - Q[i]
                Q[i] = DLI_upper

        for i in range(runtime):
            if T[i] < Tmin:
                T[i] = Tmin
            if RefT[i] < Tmin:
                RefT[i] = Tmin

        DLI = Q

        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2018-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
```

```
    plt.plot(s, RefT, lw = 1)
    plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
    plt.plot(s, T, lw = 0.3)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
        (inside)", "Daily values"], loc = 'upper right')
    plt.xlabel("Date")
    plt.ylabel("Celsius")
    plt.title("Temperature profile")

    plt.figure(82, figsize=[16, 8])
    plt.plot(s, RefDLI, lw = 1)
    plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
    plt.plot(s, DLI, s, Screening_lighting20182019, lw = 0.3)
    # plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
        (inside)", "Daily final inside values", "Applied filtering or lighting"],
            loc='lower right')
    plt.title("DLI PAR")

if year == "2019-2020":

    T[0:365] = df_weather_TG['2019'][0:365]
    T[365:730] = df_weather_TG['2020'][0:365]

    Q = np.zeros((runtime))
    Q[0:365] = df_weather_GQ['2019'][0:365]
    Q[365:730] = df_weather_GQ['2020'][0:365]

    RefDLI = np.zeros((runtime))
    RefT = np.zeros((runtime))
    RefT[0:365] = df_weather_TG['mean'][0:365]
    RefT[365:730] = df_weather_TG['mean'][0:365]
    RefDLI = DLI
    Screening_lighting20192020 = np.zeros((runtime))

    for i in range(runtime):
        if Q[i] < DLI_lower:
            Screening_lighting20192020[i] = DLI_lower - Q[i]
            Q[i] = DLI_lower
        if Q[i] > DLI_upper:
            Screening_lighting20192020[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

    for i in range(runtime):
        if T[i] < Tmin:
            T[i] = Tmin
        if RefT[i] < Tmin:
            RefT[i] = Tmin

    DLI = Q

    df = pd.DataFrame(T)
    RollingTG = df.rolling(14).mean()
    df = pd.DataFrame(DLI)
    RollingGQ = df.rolling(14).mean()
```

```
    s = pd.date_range('2019-01-01', periods=runtime, freq='D').to_series()
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
        font_scale=3, style="whitegrid")
    plt.figure(81, figsize=[16, 8])
    plt.plot(s, RefT, lw = 1)
    plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
    plt.plot(s, T, lw = 0.3)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
        (inside)", "Daily values"], loc = 'upper right')
    plt.xlabel("Date")
    plt.ylabel("Celsius")
    plt.title("Temperature profile")

    plt.figure(82, figsize=[16, 8])
    plt.plot(s, RefDLI, lw = 1)
    plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
    plt.plot(s, DLI, s, Screening_lighting20192020, lw = 0.3)
    # plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
        (inside)", "Daily final inside values", "Applied filtering or lighting"],
            loc='lower right')
    plt.title("DLI PAR")
if year == "2020-2021":

    T[0:365] = df_weather_TG['2020'][0:365]
    T[365:730] = df_weather_TG['2021'][0:365]

    Q = np.zeros((runtime))
    Q[0:365] = df_weather_GQ['2020'][0:365]
    Q[365:730] = df_weather_GQ['2021'][0:365]

    RefDLI = np.zeros((runtime))
    RefT = np.zeros((runtime))
    RefT[0:365] = df_weather_TG['mean'][0:365]
    RefT[365:730] = df_weather_TG['mean'][0:365]
    RefDLI = DLI
    Screening_lighting20202021 = np.zeros((runtime))

    for i in range(runtime):
        if Q[i] < DLI_lower:
            Screening_lighting20202021[i] = DLI_lower - Q[i]
            Q[i] = DLI_lower
        if Q[i] > DLI_upper:
            Screening_lighting20202021[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

    for i in range(runtime):
        if T[i] < Tmin:
            T[i] = Tmin
        if RefT[i] < Tmin:
            RefT[i] = Tmin

    DLI = Q
```

```
        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2020-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
        plt.plot(s, RefT, lw = 1)
        plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
        plt.plot(s, T, lw = 0.3)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
            (inside)", "Daily values"], loc = 'upper right')
        plt.xlabel("Date")
        plt.ylabel("Celsius")
        plt.title("Temperature profile")

        plt.figure(82, figsize=[16, 8])
        plt.plot(s, RefDLI, lw = 1)
        plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
        plt.plot(s, DLI, s, Screening_lighting20202021, lw = 0.3)
        # plt.plot(s, DLI_inside_primar)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.xlabel("Date")
        plt.ylabel("MJ/m^2/day")
        plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
            (inside)", "Daily final inside values", "Applied filtering or lighting"],
                loc='lower right')
        plt.title("DLI PAR")

if (Base_data == 'Darwin') or (Base_data == 'Melbourne'):
    # Select relevant years
    if year == '10-year mean':
        T = np.zeros((runtime))
        T = TMean
        T = np.nan_to_num(T)
        for i in range(len(T)):
            if T[i]<0.1:
                T[i] = T[i-1]


        Q = np.zeros((runtime))
        Q = Smean
        Q = np.nan_to_num(Q)
        for i in range(len(Q)):
            if Q[i]<0.1:
                Q[i] = Q[i-1]

        RefDLI = np.zeros((runtime))
        RefT = np.zeros((runtime))
        RefT = TMean
        RefDLI = Smean
        Screening_lighting = np.zeros((runtime))

        for i in range(runtime):
            if Q[i] < DLI_lower:
                Screening_lighting[i] = DLI_lower - Q[i]
```

```
            Q[i] = DLI_lower
        if Q[i] > DLI_upper:
            Screening_lighting[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

    for i in range(runtime):
        if T[i] < Tmin:
            T[i] = Tmin
        if RefT[i] < Tmin:
            RefT[i] = Tmin

    DLI = Q

    df = pd.DataFrame(T)
    RollingTG = df.rolling(14).mean()
    df = pd.DataFrame(DLI)
    RollingGQ = df.rolling(14).mean()

    s = pd.date_range('2018-01-01', periods=runtime, freq='D').to_series()
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
        font_scale=3, style="whitegrid")
    plt.figure(81, figsize=[16, 8])
    plt.plot(s, RefT, lw = 1)
    #plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
    #plt.plot(s, T, lw = 0.3)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.ylim([15,40])
    #plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14
        days (inside)", "Daily values"], loc = 'upper right')
    plt.xlabel("Date")
    plt.ylabel("Celsius")
    plt.title("Inside temperature")

    plt.figure(82, figsize=[16, 8])
    plt.plot(s, RefDLI, lw = 1)
    #plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
    #plt.plot( s, Screening_lighting, lw = 0.3)
    # plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[runtime - 1]])
    plt.tick_params(labelrotation=45)
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.ylim([0,14])
    #plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
        (inside)", "Applied filtering or lighting"],
    #        loc='lower right')
    plt.title("Inside DLI PAR ")

if year == "2016-2017":

    T = np.zeros((runtime))
    T[0:365] = T2016[0:365]
    T[365:730] = T2017[0:365]
    T = np.nan_to_num(T)
    for i in range(len(T)):
        if T[i]<0.1:
            T[i] = T[i-1]

    Q = np.zeros((runtime))
    Q[0:365] = Solar2016[0:365]
```

```
Q[365:730] = Solar2017[0:365]
Q = np.nan_to_num(Q)
for i in range(len(Q)):
   if Q[i]<0.1:
      Q[i] = Q[i-1]

RefDLI = np.zeros((runtime))
RefT = np.zeros((runtime))
RefT[0:365] = TMean[0:365]
RefT[365:730] = TMean[0:365]
RefDLI[0:365] = Smean[0:365]
RefDLI[365:730] = Smean[0:365]
Screening_lighting20182019 = np.zeros((runtime))

for i in range(runtime):
   if Q[i] < DLI_lower:
      Screening_lighting20182019[i] = DLI_lower - Q[i]
      Q[i] = DLI_lower
   if Q[i] > DLI_upper:
      Screening_lighting20182019[i] = DLI_upper - Q[i]
      Q[i] = DLI_upper

for i in range(runtime):
   if T[i] < Tmin:
      T[i] = Tmin
   if RefT[i] < Tmin:
      RefT[i] = Tmin

DLI = Q

df = pd.DataFrame(T)
RollingTG = df.rolling(14).mean()
df = pd.DataFrame(DLI)
RollingGQ = df.rolling(14).mean()

s = pd.date_range('2016-01-01', periods=runtime, freq='D').to_series()
sns.set_theme(context='paper', palette='colorblind', font='Georgia',
    font_scale=3, style="whitegrid")
plt.figure(81, figsize=[16, 8])
plt.plot(s, RefT, lw = 1)
plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
plt.plot(s, T, lw = 0.3)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
    (inside)", "Daily values"], loc = 'upper right')
plt.xlabel("Date")
plt.ylabel("Celsius")
plt.title("Inside Temperature")

plt.figure(82, figsize=[16, 8])
plt.plot(s, RefDLI, lw = 1)
plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
plt.plot(s, DLI, s, Screening_lighting20182019, lw = 0.3)
# plt.plot(s, DLI_inside_primar)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date")
plt.ylabel("MJ/m^2/day")
plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
    (inside)", "Daily final inside values", "Applied filtering or lighting"],
```

```
            loc='lower right')
        plt.title("Inside DLI PAR")

    if year == "2017-2018":

        T = np.zeros((runtime))
        T[0:365] = T2017[0:365]
        T[365:730] = T2018[0:365]
        T = np.nan_to_num(T)
        for i in range(len(T)):
            if T[i]<0.1:
                T[i] = T[i-1]

        Q = np.zeros((runtime))
        Q[0:365] = Solar2017[0:365]
        Q[365:730] = Solar2018[0:365]
        Q = np.nan_to_num(Q)
        for i in range(len(Q)):
            if Q[i]<0.1:
                Q[i] = Q[i-1]

        RefDLI = np.zeros((runtime))
        RefT = np.zeros((runtime))
        RefT[0:365] = TMean[0:365]
        RefT[365:730] = TMean[0:365]
        RefDLI[0:365] = Smean[0:365]
        RefDLI[365:730] = Smean[0:365]
        Screening_lighting20182019 = np.zeros((runtime))

        for i in range(runtime):
            if Q[i] < DLI_lower:
                Screening_lighting20182019[i] = DLI_lower - Q[i]
                Q[i] = DLI_lower
            if Q[i] > DLI_upper:
                Screening_lighting20182019[i] = DLI_upper - Q[i]
                Q[i] = DLI_upper

        for i in range(runtime):
            if T[i] < Tmin:
                T[i] = Tmin
            if RefT[i] < Tmin:
                RefT[i] = Tmin

        DLI = Q

        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2017-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
        plt.plot(s, RefT, lw = 1)
        plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
        plt.plot(s, T, lw = 0.3)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
            (inside)", "Daily values"], loc = 'upper right')
```

```
plt.xlabel(”Date”)
plt.ylabel(”Celsius”)
plt.title(”Inside Temperature”)

plt.figure(82, figsize=[16, 8])
plt.plot(s, RefDLI, lw = 1)
plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
plt.plot(s, DLI, s, Screening_lighting20182019, lw = 0.3)
# plt.plot(s, DLI_inside_primar)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel(”Date”)
plt.ylabel(”MJ/m^2/day”)
plt.legend([”Multi-year average final DLI inside”, ”Moving mean 14 days
    (inside)”, ”Daily final inside values”, ”Applied filtering or lighting”],
        loc=’lower right’)
plt.title(”Inside DLI PAR”)

T5yearplot = np.zeros((365*6))
DLI5yearplot = np.zeros((365*6))
RefT5yearplot = np.zeros((365*6))
RefDLI5yearplot = np.zeros((365*6))
T5yearplot[0:365] = T2016[0:365]
T5yearplot[365:730] = T2017[0:365]
T5yearplot[730:1095] = T2018[0:365]
T5yearplot[1095:1460] = T2019[0:365]
T5yearplot[1460:1825] = T2020[0:365]
T5yearplot[1825:2190] = T2021[0:365]
DLI5yearplot[0:365] = Solar2016[0:365]
DLI5yearplot[365:730] = Solar2017[0:365]
DLI5yearplot[730:1095] = Solar2018[0:365]
DLI5yearplot[1095:1460] = Solar2019[0:365]
DLI5yearplot[1460:1825] = Solar2020[0:365]
DLI5yearplot[1825:2190] = Solar2021[0:365]
RefT5yearplot[0:365] = TMean[0:365]
RefT5yearplot[365:730] = TMean[0:365]
RefT5yearplot[730:1095] = TMean[0:365]
RefT5yearplot[1095:1460] = TMean[0:365]
RefT5yearplot[1460:1825] = TMean[0:365]
RefT5yearplot[1825:2190] = TMean[0:365]
RefDLI5yearplot[0:365] = Smean[0:365]
RefDLI5yearplot[365:730] = Smean[0:365]
RefDLI5yearplot[730:1095] = Smean[0:365]
RefDLI5yearplot[1095:1460] = Smean[0:365]
RefDLI5yearplot[1460:1825] = Smean[0:365]
RefDLI5yearplot[1825:2190] = Smean[0:365]

s = pd.date_range(’2016-01-01’, periods=2190, freq=’D’).to_series()
sns.set_theme(context=’paper’, palette=’colorblind’, font=’Georgia’,
    font_scale=3, style=”whitegrid”)
plt.figure(41, figsize=[16, 4])
plt.plot(s, T5yearplot, lw = 1)
plt.plot(s, RefT5yearplot, lw = 1)
#plt.plot(s, T, lw = 0.3)
plt.xlim([s[0], s[2189]])
plt.tick_params(labelrotation=45)
plt.legend([”Daily data”, ”10-year mean”], loc = ’upper right’)
plt.xlabel(”Date”)
plt.ylabel(”Celsius”)
plt.title(”Inside temperature”)
```

```
    plt.figure(42, figsize=[16, 4])
    plt.plot(s, DLI5yearplot, lw = 1)
    plt.plot(s, RefDLI5yearplot, lw = 1)
    # plt.plot(s, DLI_inside_primar)
    plt.xlim([s[0], s[2189]])
    plt.tick_params(labelrotation=45)
    plt.legend(["Daily data", "10-year mean"], loc='upper right')
    plt.xlabel("Date")
    plt.ylabel("MJ/m^2/day")
    plt.title("Inside DLI PAR")
    s = pd.date_range('2017-01-01', periods=runtime, freq='D').to_series()

if year == "2018-2019":

    T = np.zeros((runtime))
    T[0:365] = T2018[0:365]
    T[365:730] = T2019[0:365]
    T = np.nan_to_num(T)
    for i in range(len(T)):
        if T[i]<0.1:
            T[i] = T[i-1]

    Q = np.zeros((runtime))
    Q[0:365] = Solar2018[0:365]
    Q[365:730] = Solar2019[0:365]
    Q = np.nan_to_num(Q)
    for i in range(len(Q)):
        if Q[i]<0.1:
            Q[i] = Q[i-1]

    RefDLI = np.zeros((runtime))
    RefT = np.zeros((runtime))
    RefT[0:365] = TMean[0:365]
    RefT[365:730] = TMean[0:365]
    RefDLI[0:365] = Smean[0:365]
    RefDLI[365:730] = Smean[0:365]
    Screening_lighting20182019 = np.zeros((runtime))

    for i in range(runtime):
        if Q[i] < DLI_lower:
            Screening_lighting20182019[i] = DLI_lower - Q[i]
            Q[i] = DLI_lower
        if Q[i] > DLI_upper:
            Screening_lighting20182019[i] = DLI_upper - Q[i]
            Q[i] = DLI_upper

    for i in range(runtime):
        if T[i] < Tmin:
            T[i] = Tmin
        if RefT[i] < Tmin:
            RefT[i] = Tmin

    DLI = Q

    df = pd.DataFrame(T)
    RollingTG = df.rolling(14).mean()
    df = pd.DataFrame(DLI)
    RollingGQ = df.rolling(14).mean()

    s = pd.date_range('2018-01-01', periods=runtime, freq='D').to_series()
    sns.set_theme(context='paper', palette='colorblind', font='Georgia',
```

```
    font_scale=3, style="whitegrid")
plt.figure(81, figsize=[16, 8])
plt.plot(s, RefT, lw = 1)
plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
plt.plot(s, T, lw = 0.3)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
    (inside)", "Daily values"], loc = 'upper right')
plt.xlabel("Date")
plt.ylabel("Celsius")
plt.title("Inside Temperature")

plt.figure(82, figsize=[16, 8])
plt.plot(s, RefDLI, lw = 1)
plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
plt.plot(s, DLI, s, Screening_lighting20182019, lw = 0.3)
# plt.plot(s, DLI_inside_primar)
plt.xlim([s[0], s[runtime - 1]])
plt.tick_params(labelrotation=45)
plt.xlabel("Date")
plt.ylabel("MJ/m^2/day")
plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
    (inside)", "Daily final inside values", "Applied filtering or lighting"],
        loc='lower right')
plt.title("Inside DLI PAR")

if year == "2019-2020":

   T = np.zeros((runtime))
   T[0:365] = T2019[0:365]
   T[365:730] = T2020[0:365]
   T = np.nan_to_num(T)
   for i in range(len(T)):
      if T[i] < 0.1:
         T[i] = T[i - 1]

   Q = np.zeros((runtime))
   Q[0:365] = Solar2019[0:365]
   Q[365:730] = Solar2020[0:365]
   Q = np.nan_to_num(Q)
   for i in range(len(Q)):
      if Q[i] < 0.1:
         Q[i] = Q[i - 1]

   RefDLI = np.zeros((runtime))
   RefT = np.zeros((runtime))
   RefT[0:365] = TMean[0:365]
   RefT[365:730] = TMean[0:365]
   RefDLI[0:365] = Smean[0:365]
   RefDLI[365:730] = Smean[0:365]
   Screening_lighting20182019 = np.zeros((runtime))

   for i in range(runtime):
      if Q[i] < DLI_lower:
         Screening_lighting20182019[i] = DLI_lower - Q[i]
         Q[i] = DLI_lower
      if Q[i] > DLI_upper:
         Screening_lighting20182019[i] = DLI_upper - Q[i]
         Q[i] = DLI_upper
```

```
        for i in range(runtime):
            if T[i] < Tmin:
                T[i] = Tmin
            if RefT[i] < Tmin:
                RefT[i] = Tmin

        DLI = Q

        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2019-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
        plt.plot(s, RefT, lw=1)
        plt.plot(s[14:-14], RollingTG[14:-14], lw=2)
        plt.plot(s, T, lw=0.3)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14
            days (inside)", "Daily values"],
                loc='upper right')
        plt.xlabel("Date")
        plt.ylabel("Celsius")
        plt.title("Inside Temperature")

        plt.figure(82, figsize=[16, 8])
        plt.plot(s, RefDLI, lw=1)
        plt.plot(s[14:-14], RollingGQ[14:-14], lw=2)
        plt.plot(s, DLI, s, Screening_lighting20182019, lw=0.3)
        # plt.plot(s, DLI_inside_primar)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.xlabel("Date")
        plt.ylabel("MJ/m^2/day")
        plt.legend(
            ["Multi-year average final DLI inside", "Moving mean 14 days
                (inside)", "Daily final inside values",
            "Applied filtering or lighting"],
            loc='lower right')
        plt.title("Inside DLI PAR")

if year == "2020-2021":

    T = np.zeros((runtime))
    T[0:365] = T2020[0:365]
    T[365:730] = T2021[0:365]
    T = np.nan_to_num(T)
    for i in range(len(T)):
        if T[i]<0.1:
            T[i] = T[i-1]

    Q = np.zeros((runtime))
    Q[0:365] = Solar2020[0:365]
    Q[365:730] = Solar2021[0:365]
    Q = np.nan_to_num(Q)
    for i in range(len(Q)):
        if Q[i]<0.1:
```

```
            Q[i] = Q[i-1]

        RefDLI = np.zeros((runtime))
        RefT = np.zeros((runtime))
        RefT[0:365] = TMean[0:365]
        RefT[365:730] = TMean[0:365]
        RefDLI[0:365] = Smean[0:365]
        RefDLI[365:730] = Smean[0:365]
        Screening_lighting20182019 = np.zeros((runtime))

        for i in range(runtime):
           if Q[i] < DLI_lower:
              Screening_lighting20182019[i] = DLI_lower - Q[i]
              Q[i] = DLI_lower
           if Q[i] > DLI_upper:
              Screening_lighting20182019[i] = DLI_upper - Q[i]
              Q[i] = DLI_upper

        for i in range(runtime):
           if T[i] < Tmin:
              T[i] = Tmin
           if RefT[i] < Tmin:
              RefT[i] = Tmin

        DLI = Q
        df = pd.DataFrame(T)
        RollingTG = df.rolling(14).mean()
        df = pd.DataFrame(DLI)
        RollingGQ = df.rolling(14).mean()

        s = pd.date_range('2020-01-01', periods=runtime, freq='D').to_series()
        sns.set_theme(context='paper', palette='colorblind', font='Georgia',
            font_scale=3, style="whitegrid")
        plt.figure(81, figsize=[16, 8])
        plt.plot(s, RefT, lw = 1)
        plt.plot(s[14:-14], RollingTG[14:-14], lw = 2)
        plt.plot(s, T, lw = 0.3)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.legend(["Multi-year average Temperature (inside)", "Moving mean 14 days
            (inside)", "Daily values"], loc = 'upper right')
        plt.xlabel("Date")
        plt.ylabel("Celsius")
        plt.title("Inside Temperature")

        plt.figure(82, figsize=[16, 8])
        plt.plot(s, RefDLI, lw = 1)
        plt.plot(s[14:-14], RollingGQ[14:-14], lw = 2)
        plt.plot(s, DLI, s, Screening_lighting20182019, lw = 0.3)
        # plt.plot(s, DLI_inside_primar)
        plt.xlim([s[0], s[runtime - 1]])
        plt.tick_params(labelrotation=45)
        plt.xlabel("Date")
        plt.ylabel("MJ/m^2/day")
        plt.legend(["Multi-year average final DLI inside", "Moving mean 14 days
            (inside)", "Daily final inside values", "Applied filtering or lighting"],
                loc='lower right')
        plt.title("Inside DLI PAR")

# ======================================================================
# ============ Simulation ==============================================
```

```
# ====================================================================
class TBatch(sim.Component):
    def setup(self, batchsize, LossRate, runtime):
        self.FDS = 0 # Initialize FDS
        self.TW = 0 # Initialize FFW
        self.NrPlants = batchsize # Minimum batch size
        self.TrussesPerPlant = 0
        self.NoGutters = 0
        self.NoPlantsGutter = 0
        self.NoPlantsTray1 = 0
        self.NoPlantsTray2 = 0
        self.NoTrays1 = 0
        self.NoTrays2 = 0
        self.Timer = 0
        self.StartDate = 0
        self.CurrentTime = 0
        self.TrussDensity = 0
        self.InterArrivalTime = sim.Constant(1)
        self.LossRate = LossRate
        self.LOS = 0
        self.PlacedInWrongCompartment = "" #fill in here the actual desired
            compartment
        self.TimeInWrongCompartment = 0
        self.TransportTimeCum = 0 # initialize
        self.TransportDistanceCum = 0
        self.NumberTransportsCum = 0

    def process(self):
        self.Day = self.InterArrivalTime.sample()
        self.j = 0
        while self.j <= runtime:
            if self.LossRate > 0:
                self.NrPlants -= 0# int(self.NrPlants * self.LossRate / 100 /
                    np.sum(self.LOS))
            self.j += self.Day
            yield self.hold(self.Day)

class THarvestedTruss(sim.Component):
    def setup(self):
        self.TW = 0
        self.LOS = 0
        self.Revenue = 0
        self.TransportTimeCum = 0
        self.TransportDistanceCum = 0
        self.NumberTransportsCum = 0

class TLostCrop(sim.Component):
    def setup(self):
        self.LossTime = 0
        self.LossReason = 0
        self.TransportTimeCum = 0
        self.TransportDistanceCum = 0
        self.NumberTransportsCum = 0

class TSowingGenerator(sim.Component):
    def setup(self, SowingRate, SowingTime, LOS, runtime, number_of_trusses,
        batchsize, plants_per_gutter,
            plants_per_tray_1, plants_per_tray_2, TrussDensity, LossRate):
        self.InterArrivalTime = sim.Constant(1)
        self.SeedsPerDay = SowingRate
        self.SowingTime = SowingTime
```

```
        self.number_of_trusses = number_of_trusses
        self.LOS = LOS
        self.plants_per_gutter = plants_per_gutter
        self.plants_per_tray1 = plants_per_tray_1
        self.plants_per_tray2 = plants_per_tray_2
        self.runtime = runtime
        self.TimeNeeded = 0
        self.CurrentDay = 0
        self.Timer = 0
        self.batchsize = batchsize
        self.TrussDensity = TrussDensity
        self.LossRate = LossRate
        self.CalculatedSowingCum = 0
        self.OperationalTime = np.zeros((self.runtime))
    def process(self):
        self.SingleDay = self.InterArrivalTime.sample()
        for self.j in range(self.runtime):
            self.CurrentDay += self.SingleDay
            print(np.round(env.now(), decimals = 1))
            self.Timer = 0
            self.CalculatedSowingCum += self.SeedsPerDay[self.j]
            self.SowToday = self.CalculatedSowingCum - np.sum(Sowed[0:self.j])
            self.NrPlants = self.batchsize
            if self.SowToday > 0:
                self.NrBatches = int(np.floor(self.SowToday / self.batchsize))
            else:
                self.NrBatches = 0
            #self.BatchArea = self.NrPlants / (self.TrussDensity[0][self.j] /
                self.number_of_trusses[self.j])
            #self.NrBatches = int(np.floor((GreenhouseC1.Capacity -
                GreenhouseC1.Occupancy) / self.BatchArea))
            self.BatchTime = self.SowingTime * self.NrPlants /
                self.plants_per_tray1[self.j]
            Sowed[self.j] = self.NrBatches * self.NrPlants
            if self.NrBatches > 0:
                self.MaxBatches = int(np.floor([self.SingleDay / self.BatchTime]))
                if self.MaxBatches < self.NrBatches:
                    print("Only ", self.MaxBatches, "Batches of desired ",
                        self.NrBatches, "Sowed")
                    self.NrBatches = self.MaxBatches

                for self.i in range(self.NrBatches):
                    self.Batch = TBatch(batchsize = self.batchsize, LossRate =
                        LossRate, runtime = runtime)
                    self.Batch.TrussesPerPlant = self.number_of_trusses[self.j]
                    self.Batch.NoGutters = np.ceil(self.batchsize /
                        self.plants_per_gutter[self.j])
                    self.Batch.NoPlantsGutter = self.plants_per_gutter[self.j]
                    self.Batch.NoTrays1 = np.ceil(self.batchsize /
                        self.plants_per_tray1[self.j])
                    self.Batch.NoTrays2 = np.ceil(self.batchsize /
                        self.plants_per_tray2[self.j])
                    self.Batch.Timer = 0
                    self.Batch.StartDate = self.j
                    self.Batch.CurrentTime = self.j
                    self.Batch.TrussDensity = self.TrussDensity[:,self.j]
                    self.Batch.LOS = [self.LOS[0][self.j], self.LOS[1][self.j],
                        self.LOS[2][self.j],
                                  self.LOS[3][self.j], self.LOS[4][self.j],
                                      self.LOS[5][self.j]]
```

```
                yield self.hold(self.SowingTime * self.NrPlants /
                    self.plants_per_tray1[self.j]) #Sowing Time per Tray
                self.OperationalTime[int(env.now())] += self.SowingTime *
                    self.NrPlants / self.plants_per_tray1[self.j]
                self.Timer += self.SowingTime * self.NrPlants /
                    self.plants_per_tray1[self.j]
                self.Batch.Timer = env.now()
                Sowing_done_queue.add(self.Batch)

            if (self.SingleDay - self.Timer) > 10 ** -10:
               yield self.hold((self.SingleDay - self.Timer))

class TTrayCropGreenhouseShuttle(sim.Component):
   def setup(self, DriveTime, distance, runtime):
      self.DriveTime = DriveTime
      self.distance = distance
      self.runtime = runtime
      self.InterArrivalTime = sim.Constant(1)
      self.Day = self.InterArrivalTime.sample()

   def process(self):
      while True:
      #for self.z in range(self.runtime):
         while (Sowing_done_queue.length() == 0) and
             (Transplant1_done_queue.length() == 0):
            yield self.standby()

         #yield self.hold(self.Day/2)
         #self.Time = 0
         while Sowing_done_queue.length() > 0:
         #for self.j in range(Sowing_done_queue.length()):
            # start_time = timeit.default_timer()
            self.Batch = Sowing_done_queue.pop()
            self.Batch.PlacedInWrongCompartment = ""

            if self.Batch.Timer < (env.now() - 1):
               print("Waited too long after sowing")
               Lost_component_queue.add(self.Batch)
               for self.i in range(self.Batch.NrPlants):
                  self.LostCrop = TLostCrop()
                  self.LostCrop.LossTime = env.now()
                  if (env.now() < 135) and (env.now() < 500):
                     self.LostCrop.LossReason = 22
                  LostCrops_queue.add(self.LostCrop)
               if (env.now() >= 135) and (env.now() <= 501):
                  Finsihed_Batch_queue.add(self.Batch)

            elif GreenhouseC1.Capacity > (GreenhouseC1.Occupancy +
                self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                self.Batch.TrussDensity[0]):
               yield self.hold(self.DriveTime[0] * self.Batch.NoTrays1)
               self.Batch.TransportTimeCum += self.DriveTime[0] *
                   self.Batch.NoTrays1
               self.Batch.TransportDistanceCum += self.distance[0] *
                   self.Batch.NoTrays1
               self.Batch.NumberTransportsCum += self.Batch.NoTrays1
               self.Batch.Timer = env.now()
               S1_queue.add(self.Batch)
               self.BatchArea = self.Batch.NrPlants * self.Batch.TrussesPerPlant
                   / self.Batch.TrussDensity[0]
               GreenhouseC1.Occupancy += self.BatchArea
```

```
            # print(timeit.default_timer() - start_time)

        elif GreenhouseC2.Capacity > (GreenhouseC2.Occupancy +
            self.Batch.NrPlants * self.Batch.TrussesPerPlant /
            self.Batch.TrussDensity[0]):
            GreenhouseC2.Occupancy += self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
            yield self.hold(self.DriveTime[1] * self.Batch.NoTrays1)
            self.Batch.TransportTimeCum += self.DriveTime[1] *
                self.Batch.NoTrays1
            self.Batch.TransportDistanceCum += self.distance[1] *
                self.Batch.NoTrays1
            self.Batch.NumberTransportsCum += self.Batch.NoTrays1
            self.Batch.Timer = env.now()
            self.Batch.PlacedInWrongCompartment = "C1"
            S2_queue.add(self.Batch)
        else:
            print("Lost due to no space in C1")
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                self.LostCrop.LossReason = 1
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)

    while Transplant1_done_queue.length() > 0:
    #for self.k in range(Transplant1_done_queue.length()):
        # start_time = timeit.default_timer()
        self.Batch = Transplant1_done_queue.pop()
        self.Batch.PlacedInWrongCompartment = ""

        if self.Batch.Timer < (env.now() - 1):
            Lost_component_queue.add(self.Batch)
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                self.LostCrop.LossReason = 12
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)

        elif GreenhouseC2.Capacity > (GreenhouseC2.Occupancy +
            self.Batch.NrPlants * self.Batch.TrussesPerPlant /
            self.Batch.TrussDensity[1]):
            GreenhouseC2.Occupancy += self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
            yield self.hold(self.DriveTime[1] * self.Batch.NoTrays2)
            self.Batch.TransportTimeCum += self.DriveTime[1] *
                self.Batch.NoTrays2
            self.Batch.TransportDistanceCum += self.distance[1] *
                self.Batch.NoTrays2
            self.Batch.NumberTransportsCum += self.Batch.NoTrays2
            self.Batch.Timer = env.now()
            S2_queue.add(self.Batch)

        elif GreenhouseC1.Capacity > (GreenhouseC1.Occupancy +
            self.Batch.NrPlants * self.Batch.TrussesPerPlant /
            self.Batch.TrussDensity[1]):
            yield self.hold(self.DriveTime[1] * self.Batch.NoTrays2)
            self.Batch.TransportTimeCum += self.DriveTime[1] *
```

```
                    self.Batch.NoTrays2
                self.Batch.TransportDistanceCum += self.distance[1] *
                    self.Batch.NoTrays2
                self.Batch.NumberTransportsCum += self.Batch.NoTrays2
                self.Batch.Timer = env.now()
                self.Batch.PlacedInWrongCompartment = "C2"
                S1_queue.add(self.Batch)
                self.BatchArea = self.Batch.NrPlants * self.Batch.TrussesPerPlant
                    / self.Batch.TrussDensity[1]
                GreenhouseC1.Occupancy += self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]

            else:
                print("Lost due to no space in C2")
                for self.i in range(self.Batch.NrPlants):
                    LostCrop = TLostCrop()
                    LostCrop.LossTime = env.now()
                    LostCrop.LossReason = 2
                    LostCrops_queue.add(LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)

        #self.Time = env.now() - self.z

            # print(timeit.default_timer() - start_time)

class TTrayGreenhouseCropShuttle(sim.Component):
    def setup(self, DriveTime, distance, runtime):
        self.DriveTime = DriveTime
        self.distance = distance
        self.runtime = runtime
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()

    def process(self):
        while True:
        #for self.z in range(self.runtime):
            #self.Time = 0
            while (S1_leave_queue.length() == 0) and (S2_leave_queue.length() == 0):
                yield self.standby()

            while S1_leave_queue.length() > 0:
            #for self.j in range(S1_leave_queue.length()):
                #start_time = timeit.default_timer()
                self.Batch = S1_leave_queue.pop()

                # 2) Check if it waits too long
                if self.Batch.Timer < (env.now() - 1):
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        if (env.now() < 135) and (env.now() < 500):
                            self.LostCrop.LossReason = 21
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)
                # 3) Go through tree
                else:
                    # if in right compartment
                    if self.Batch.PlacedInWrongCompartment == "":
```

```
            yield self.hold(self.DriveTime[0] * self.Batch.NoTrays1)
            self.Batch.TransportTimeCum += self.DriveTime[0] *
                self.Batch.NoTrays1
            self.Batch.TransportDistanceCum += self.distance[0] *
                self.Batch.NoTrays1
            self.Batch.NumberTransportsCum += self.Batch.NoTrays1
            self.Batch.Timer = env.now()
            Transplant1_queue.add(self.Batch)

        else: # Not in right compartment
            # Assign space
            if self.Batch.PlacedInWrongCompartment == "C2":
                self.BatchArea = self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
                # check if ready for next stage
                if (dFDS1 + dFDS2) < self.Batch.FDS:
                    yield self.hold(self.DriveTime[0] * self.Batch.NoTrays2)
                    self.Batch.TransportTimeCum += self.DriveTime[0] *
                        self.Batch.NoTrays2
                    self.Batch.TransportDistanceCum += self.distance[0] *
                        self.Batch.NoTrays2
                    self.Batch.NumberTransportsCum += self.Batch.NoTrays2
                    self.Batch.Timer = env.now()
                    Transplant2_queue.add(self.Batch)

                # Check for space in right compartment
                elif (GreenhouseC2.Occupancy + self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1])
                    <= GreenhouseC2.Capacity:
                    self.Batch.PlacedInWrongCompartment = ""
                    GreenhouseC2.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
                    yield self.hold(self.DriveTime[0] * self.Batch.NoTrays2)
                        # update this drive time
                    self.Batch.TransportTimeCum += self.DriveTime[0] *
                        self.Batch.NoTrays2
                    self.Batch.TransportDistanceCum += self.distance[0] *
                        self.Batch.NoTrays2
                    self.Batch.NumberTransportsCum += self.Batch.NoTrays2
                    S2_queue.add(self.Batch)

                # Check for space in another acceptable compartment
                elif (GreenhouseC1.Occupancy + self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1])
                    < (GreenhouseC2.Capacity):
                    self.Batch.Timer = env.now()
                    self.Batch.PlacedInWrongCompartment = "C2"
                    GreenhouseC1.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
                    S1_queue.add(self.Batch)

                # Else batch is lost
                else:
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 2
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)
```

```
            else:
                print("Unknown Batch lost 2")
                Lost_component_queue.add(self.Batch)
                for self.i in range(self.Batch.NrPlants):
                    self.LostCrop = TLostCrop()
                    self.LostCrop.LossTime = env.now()
                    if (env.now() < 135) and (env.now() < 500):
                        self.LostCrop.LossReason = 21
                    LostCrops_queue.add(self.LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)
    #print(timeit.default_timer() - start_time)

while S2_leave_queue.length():
#for self.k in range(S2_leave_queue.length()):
    # start_time = timeit.default_timer()
    self.Batch = S2_leave_queue.pop()

    # 2) Check if it waits too long
    if self.Batch.Timer < (env.now() - 1):
        Lost_component_queue.add(self.Batch)
        for self.i in range(self.Batch.NrPlants):
            self.LostCrop = TLostCrop()
            self.LostCrop.LossTime = env.now()
            if (env.now() < 135) and (env.now() < 500):
                self.LostCrop.LossReason = 21
            LostCrops_queue.add(self.LostCrop)
        if (env.now() >= 135) and (env.now() <= 501):
            Finsihed_Batch_queue.add(self.Batch)
    # 3) Go through tree
    else:
        # if in right compartment
        if self.Batch.PlacedInWrongCompartment == "":
            yield self.hold(self.DriveTime[2] * self.Batch.NoTrays2)
            self.Batch.TransportTimeCum += self.DriveTime[1] *
                self.Batch.NoTrays2
            self.Batch.TransportDistanceCum += self.distance[1] *
                self.Batch.NoTrays2
            self.Batch.NumberTransportsCum += self.Batch.NoTrays2
            self.Batch.Timer = env.now()
            Transplant2_queue.add(self.Batch)

        else: # Not in right compartment
            # Assign space
            if self.Batch.PlacedInWrongCompartment == "C1":
                self.BatchArea = self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
                # check if ready for next stage
                if dFDS1 < self.Batch.FDS:
                    yield self.hold(self.DriveTime[0] * self.Batch.NoTrays1)
                    self.Batch.TransportTimeCum += self.DriveTime[0] *
                        self.Batch.NoTrays1
                    self.Batch.TransportDistanceCum += self.distance[0] *
                        self.Batch.NoTrays1
                    self.Batch.NumberTransportsCum += self.Batch.NoTrays1
                    self.Batch.Timer = env.now()
                    Transplant1_queue.add(self.Batch)

                # Check for space in right compartment
                elif (GreenhouseC1.Occupancy + self.BatchArea) <=
```

```
                        (GreenhouseC1.Capacity):
                        self.Batch.PlacedInWrongCompartment = ""
                        yield self.hold(self.DriveTime[0] * self.Batch.NoTrays1)
                            # update this drive time
                        self.Batch.TransportTimeCum += self.DriveTime[0] *
                            self.Batch.NoTrays1
                        self.Batch.TransportDistanceCum += self.distance[0] *
                            self.Batch.NoTrays1
                        self.Batch.NumberTransportsCum += self.Batch.NoTrays1
                        GreenhouseC1.Occupancy += self.Batch.NrPlants *
                            self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
                        S1_queue.add(self.Batch)

                    # Check for space in another acceptable compartment
                    elif (GreenhouseC2.Occupancy + self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0])
                        < (
                           GreenhouseC2.Capacity):
                        self.Batch.PlacedInWrongCompartment = "C1"
                        self.Batch.Timer = env.now()
                        GreenhouseC2.Occupancy += self.Batch.NrPlants *
                            self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
                        S2_queue.add(self.Batch)

                    # Else batch is lost
                    else:
                        Lost_component_queue.add(self.Batch)
                        for self.i in range(self.Batch.NrPlants):
                            self.LostCrop = TLostCrop()
                            self.LostCrop.LossTime = env.now()
                            self.LostCrop.LossReason = 1
                            LostCrops_queue.add(self.LostCrop)
                        if (env.now() >= 135) and (env.now() <= 501):
                            Finsihed_Batch_queue.add(self.Batch)

                else:
                    print("Unknown Batch lost 4")
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        if (env.now() < 135) and (env.now() < 500):
                            self.LostCrop.LossReason = 21
                        LostCrops_queue.add(self.LostCrop) #
                            print(timeit.default_timer() - start_time)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)

class TGutterCropGreenhouseShuttle(sim.Component):
    def setup(self, DriveTime, distance, runtime):
        self.DriveTime = DriveTime
        self.distance = distance
        self.runtime = runtime
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()

    def process(self):
        while True:
        #for self.z in range(self.runtime):
            #yield self.hold(self.Day/2)
            while (Transplant2_done_queue.length() == 0) and
```

```
        (Monitoring_done_queue.length() == 0):
        yield self.standby()

    while Transplant2_done_queue.length() > 0:
    #for self.j in range(Transplant2_done_queue.length()):
        #start_time = timeit.default_timer()
        self.Batch = Transplant2_done_queue.pop()
        self.Batch.PlacedInWrongCompartment = ""

        if self.Batch.Timer < (env.now() - 1):
            print("Lost due to too late 20")
            Lost_component_queue.add(self.Batch)
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                if (env.now() < 135) and (env.now() < 500):
                    self.LostCrop.LossReason = 23
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)

        elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
            self.Batch.NrPlants * self.Batch.TrussesPerPlant /
            self.Batch.TrussDensity[2]):
            yield self.hold(self.DriveTime[2] * self.Batch.NoGutters)
            self.Batch.TransportTimeCum += self.DriveTime[2] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[2] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            self.Batch.Timer = env.now()
            S3_queue.add(self.Batch)
            self.BatchArea = self.Batch.NrPlants * self.Batch.TrussesPerPlant
                / self.Batch.TrussDensity[2]
            GreenhouseC3.Occupancy += self.BatchArea

        elif GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
            self.Batch.NrPlants * self.Batch.TrussesPerPlant /
            self.Batch.TrussDensity[2]):
            yield self.hold(self.DriveTime[2] * self.Batch.NoGutters)
            self.Batch.TransportTimeCum += self.DriveTime[2] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[2] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            self.Batch.Timer = env.now()
            self.Batch.PlacedInWrongCompartment = "C3"
            S4a_queue.add(self.Batch)
            self.BatchArea = self.Batch.NrPlants * self.Batch.TrussesPerPlant
                / self.Batch.TrussDensity[2]
            GreenhouseC4a.Occupancy += self.BatchArea

        else:
            print("Lost due to no space in C3")
            for self.i in range(self.Batch.NrPlants):
                LostCrop = TLostCrop()
                LostCrop.LossTime = env.now()
                LostCrop.LossReason = 3
                LostCrops_queue.add(LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)
```

```
            # print(timeit.default_timer() - start_time)

        while Monitoring_done_queue.length() > 0:
        #for self.k in range(Monitoring_done_queue.length()):
            # start_time = timeit.default_timer()
            self.Batch = Monitoring_done_queue.pop()
            self.Batch.PlacedInWrongCompartment = ""
#           print(self.Batch.FDS)

            if self.Batch.Timer < (env.now() - 1):
                print("Lost due to too late 22")
                Lost_component_queue.add(self.Batch)
                for self.i in range(self.Batch.NrPlants):
                    self.LostCrop = TLostCrop()
                    self.LostCrop.LossTime = env.now()
                    self.LostCrop.LossReason = 14
                    LostCrops_queue.add(self.LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)

            elif self.Batch.FDS < (dFDS1 + dFDS2 + dFDS3 + dFDS4a):

                if GreenhouseC4a.Capacity > (GreenhouseC4a.Occupancy +
                    self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                    self.Batch.TrussDensity[3]):
                    yield self.hold(self.DriveTime[3] * self.Batch.NoGutters)
                    self.Batch.TransportTimeCum += self.DriveTime[3] *
                        self.Batch.NoGutters
                    self.Batch.TransportDistanceCum += self.distance[3] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    self.Batch.Timer = env.now()
                    S4a_queue.add(self.Batch)
                    GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]

                elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                    self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                    self.Batch.TrussDensity[3]):
                    self.BatchArea = self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]
                    GreenhouseC4b.Occupancy += self.BatchArea
                    if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                        print("Error is here 1")
                    yield self.hold(self.DriveTime[4] * self.Batch.NoGutters)
                    self.Batch.TransportTimeCum += self.DriveTime[4] *
                        self.Batch.NoGutters
                    self.Batch.TransportDistanceCum += self.distance[4] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    self.Batch.Timer = env.now()
                    self.Batch.PlacedInWrongCompartment = "C4a"
                    S4b_queue.add(self.Batch)

                elif GreenhouseC3.Capacity > (GreenhouseC3.Occupancy +
                    self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                    self.Batch.TrussDensity[3]):
                    self.BatchArea = self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]
                    GreenhouseC3.Occupancy += self.BatchArea
                    yield self.hold(self.DriveTime[2] * self.Batch.NoGutters)
```

```
                self.Batch.TransportTimeCum += self.Batch.NoGutters # initialize
                self.Batch.TransportDistanceCum += self.distance[2] *
                    self.Batch.NoGutters
                self.Batch.NumberTransportsCum += self.Batch.NoGutters
                self.Batch.Timer = env.now()
                self.Batch.PlacedInWrongCompartment = "C4a"
                S3_queue.add(self.Batch)


            else:
                print("Lost due to no place in C4a")
                for self.i in range(self.Batch.NrPlants):
                    LostCrop = TLostCrop()
                    LostCrop.LossTime = env.now()
                    LostCrop.LossReason = 4
                    LostCrops_queue.add(LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)


        else:
            if GreenhouseC5.Capacity > (GreenhouseC5.Occupancy +
                self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                self.Batch.TrussDensity[5]):
                yield self.hold(self.DriveTime[5] * self.Batch.NoGutters)
                self.Batch.TransportTimeCum += self.DriveTime[5] *
                    self.Batch.NoGutters
                self.Batch.TransportDistanceCum += self.distance[5] *
                    self.Batch.NoGutters
                self.Batch.NumberTransportsCum += self.Batch.NoGutters
                self.Batch.Timer = env.now()
                S5_queue.add(self.Batch)
                GreenhouseC5.Occupancy += self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]

            elif GreenhouseC4b.Capacity > (GreenhouseC4b.Occupancy +
                self.Batch.NrPlants * self.Batch.TrussesPerPlant /
                self.Batch.TrussDensity[5]):
                yield self.hold(self.DriveTime[4] * self.Batch.NoGutters)
                self.Batch.TransportTimeCum += self.DriveTime[4] *
                    self.Batch.NoGutters
                self.Batch.TransportDistanceCum += self.distance[4] *
                    self.Batch.NoGutters
                self.Batch.NumberTransportsCum += self.Batch.NoGutters
                self.Batch.Timer = env.now()
                self.Batch.PlacedInWrongCompartment = "C5"
                S4b_queue.add(self.Batch)
                GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]
                if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                    print("Error is here 2")
            else:
                print("Lost due to no place in C5")
                for self.i in range(self.Batch.NrPlants):
                    LostCrop = TLostCrop()
                    LostCrop.LossTime = env.now()
                    LostCrop.LossReason = 5
                    LostCrops_queue.add(LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)
```

```
            # print(timeit.default_timer() - start_time)

class TGutterGreenhouseCropShuttle(sim.Component):
   def setup(self, DriveTime, distance, runtime):
      self.DriveTime = DriveTime
      self.distance = distance
      self.runtime = runtime
      self.InterArrivalTime = sim.Constant(1)
      self.Day = self.InterArrivalTime.sample()

   def process(self):
      while True:
      #for self.z in range(self.runtime):
         while (S3_leave_queue.length() == 0) and (S4b_leave_queue.length() == 0)
             and (S5_leave_queue.length() == 0):
            yield self.standby()

         #for self.j in range(S3_leave_queue.length()):
            # start_time = timeit.default_timer()
         while S4a_leave_comp_queue.length() > 0:
            self.Batch = S4a_leave_comp_queue.pop()

            # 2) Check if it waits too long
            if self.Batch.Timer < (env.now() - 1):
               print(self.Batch.Timer, env.now())
               Lost_component_queue.add(self.Batch)
               print("Waited too long")
               for self.i in range(self.Batch.NrPlants):
                  self.LostCrop = TLostCrop()
                  self.LostCrop.LossTime = env.now()
                  if (env.now() < 135) and (env.now() < 500):
                     self.LostCrop.LossReason = 24
                  LostCrops_queue.add(self.LostCrop)
               if (env.now() >= 135) and (env.now() <= 501):
                  Finsihed_Batch_queue.add(self.Batch)

            # 3) Go through tree (simplified as per definition in wrong
                compartment)
            else:
               if self.Batch.PlacedInWrongCompartment == "C3":
                  self.Batch.PlacedInWrongCompartment = ""
                  self.BatchArea = self.Batch.NrPlants *
                      self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]
                  # check if ready for next stage
                  if (dFDS1 + dFDS2 + dFDS3) < self.Batch.FDS:
                     yield self.hold(self.DriveTime[3] * self.Batch.NoGutters)
                     self.Batch.TransportTimeCum += self.DriveTime[3] *
                         self.Batch.NoGutters
                     self.Batch.TransportDistanceCum += self.distance[3] *
                         self.Batch.NoGutters
                     self.Batch.NumberTransportsCum += self.Batch.NoGutters
                     self.Batch.PlacedInWrongCompartment = ""
                     self.Batch.Timer = env.now()
                     Monitoring_queue.add(self.Batch)

                  elif (GreenhouseC3.Occupancy + self.Batch.NrPlants *
                      self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]) <
                      GreenhouseC3.Capacity:
                     yield self.hold(self.DriveTime[6] * self.Batch.NoGutters)
                     self.Batch.TransportTimeCum += self.DriveTime[6] *
                         self.Batch.NoGutters
```

```
                    self.Batch.TransportDistanceCum += self.distance[6] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    self.Batch.Timer = env.now()
                    S3_queue.add(self.Batch)
                    self.BatchArea = self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]
                    GreenhouseC3.Occupancy += self.BatchArea

                elif (GreenhouseC4a.Occupancy + self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]) <
                    GreenhouseC4a.Capacity:
                    yield self.hold(self.DriveTime[6] * self.Batch.NoGutters)
                    self.Batch.TransportTimeCum += self.DriveTime[6] *
                        self.Batch.NoGutters
                    self.Batch.TransportDistanceCum += self.distance[6] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    self.Batch.Timer = env.now()
                    self.Batch.PlacedInWrongCompartment = "C3"
                    S4a_queue.add(self.Batch)
                    GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]
                else:
                    Lost_component_queue.add(self.Batch)
                    print("Lost due to no place in C3, 2")
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 3
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)

            elif self.Batch.PlacedInWrongCompartment == "C4b":
                self.Batch.PlacedInWrongCompartment = ""
                # check if ready for next stage
                if (dFDS1 + dFDS2 + dFDS3 + dFDS4a + dFDS4b) < self.Batch.FDS:
                    yield self.hold(self.DriveTime[3] * self.Batch.NoGutters)
                    self.Batch.TransportTimeCum += self.DriveTime[3] *
                        self.Batch.NoGutters
                    self.Batch.TransportDistanceCum += self.distance[3] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    self.Batch.PlacedInWrongCompartment = ""
                    self.Batch.Timer = env.now()
                    Monitoring_queue.add(self.Batch)

                elif (GreenhouseC4b.Occupancy + self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]) <
                    GreenhouseC4b.Capacity:
                    GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]
                    if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                        print("Error is here 3")
                    yield self.hold(self.DriveTime[6] * self.Batch.NoGutters)
                    self.Batch.TransportTimeCum += self.DriveTime[6] *
                        self.Batch.NoGutters
                    self.Batch.TransportDistanceCum += self.distance[6] *
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
```

```
                self.Batch.Timer = env.now()
                S4b_queue.add(self.Batch)


            elif (GreenhouseC4a.Occupancy + self.Batch.NrPlants *
                 self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]) <
                 GreenhouseC4a.Capacity:
                yield self.hold(self.DriveTime[6] * self.Batch.NoGutters)
                self.Batch.TransportTimeCum += self.DriveTime[6] *
                    self.Batch.NoGutters
                self.Batch.TransportDistanceCum += self.distance[6] *
                    self.Batch.NoGutters
                self.Batch.NumberTransportsCum += self.Batch.NoGutters
                self.Batch.Timer = env.now()
                self.Batch.PlacedInWrongCompartment = "C4b"
                S4a_queue.add(self.Batch)
                GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]
            else:
                Lost_component_queue.add(self.Batch)
                for self.i in range(self.Batch.NrPlants):
                    self.LostCrop = TLostCrop()
                    self.LostCrop.LossTime = env.now()
                    self.LostCrop.LossReason = 3
                    LostCrops_queue.add(self.LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)
        else:
            print("Unknown Batch lost 5")
            Lost_component_queue.add(self.Batch)
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                if (env.now() < 135) and (env.now() < 500):
                    self.LostCrop.LossReason = 24
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)

    while S3_leave_queue.length() > 0:
        self.Batch = S3_leave_queue.pop()

        # 2) Check if it waits too long
        if self.Batch.Timer < (env.now() - 1):
            Lost_component_queue.add(self.Batch)
            print("Waited too long 2")
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                if (env.now() < 135) and (env.now() < 500):
                    self.LostCrop.LossReason = 24
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)
        # 3) Go through tree
        else:
            # if in right compartment
            if self.Batch.PlacedInWrongCompartment == "C3":
                self.Batch.PlacedInWrongCompartment = ""
            if self.Batch.PlacedInWrongCompartment == "":
                yield self.hold(self.DriveTime[2] * self.Batch.NoGutters)
```

```
                self.Batch.TransportTimeCum += self.DriveTime[2] *
                    self.Batch.NoGutters
                self.Batch.TransportDistanceCum += self.distance[2] *
                    self.Batch.NoGutters
                self.Batch.NumberTransportsCum += self.Batch.NoGutters
                self.Batch.Timer = env.now()
                Monitoring_queue.add(self.Batch)

            else: # Not in right compartment
                # Assign space
                if self.Batch.PlacedInWrongCompartment == "C4a":
                    self.Batch.PlacedInWrongCompartment = ""
                    # check if ready for next stage
                    if (dFDS1 + dFDS2 + dFDS3 + dFDS4a) < self.Batch.FDS:
                        if (GreenhouseC4a.Occupancy + self.Batch.NrPlants *
                            self.Batch.TrussesPerPlant /
                            self.Batch.TrussDensity[3]) < GreenhouseC4a.Capacity:
                            yield self.hold(self.DriveTime[6] *
                                self.Batch.NoGutters)
                            self.Batch.TransportTimeCum += self.DriveTime[6] *
                                self.Batch.NoGutters
                            self.Batch.TransportDistanceCum += self.distance[6] *
                                self.Batch.NoGutters
                            self.Batch.NumberTransportsCum += self.Batch.NoGutters
                            self.Batch.PlacedInWrongCompartment = ""
                            self.Batch.Timer = env.now()
                            S4a_queue.add(self.Batch)
                            GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                                self.Batch.TrussesPerPlant /
                                self.Batch.TrussDensity[3]

                        elif (GreenhouseC4b.Occupancy + self.Batch.NrPlants *
                            self.Batch.TrussesPerPlant /
                            self.Batch.TrussDensity[3]) < GreenhouseC4b.Capacity:
                            GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                                self.Batch.TrussesPerPlant /
                                self.Batch.TrussDensity[3]
                            if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                                print("Error is here 4")
                            yield self.hold(self.DriveTime[6] *
                                self.Batch.NoGutters)
                            self.Batch.TransportTimeCum += self.DriveTime[6] *
                                self.Batch.NoGutters
                            self.Batch.TransportDistanceCum += self.distance[6] *
                                self.Batch.NoGutters
                            self.Batch.NumberTransportsCum += self.Batch.NoGutters
                            self.Batch.PlacedInWrongCompartment = "C4a"
                            self.Batch.Timer = env.now()
                            S4b_queue.add(self.Batch)

                        elif (GreenhouseC3.Occupancy + self.Batch.NrPlants *
                            self.Batch.TrussesPerPlant /
                            self.Batch.TrussDensity[3]) < GreenhouseC3.Capacity:
                            yield self.hold(self.DriveTime[6] *
                                self.Batch.NoGutters)
                            self.Batch.TransportTimeCum += self.DriveTime[6] *
                                self.Batch.NoGutters
                            self.Batch.TransportDistanceCum += self.distance[6] *
                                self.Batch.NoGutters
                            self.Batch.NumberTransportsCum += self.Batch.NoGutters
                            self.Batch.PlacedInWrongCompartment = "C4a"
```

```
                    self.Batch.Timer = env.now()
                    S3_queue.add(self.Batch)
                    GreenhouseC3.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant /
                        self.Batch.TrussDensity[3]

                else:
                    Lost_component_queue.add(self.Batch)
                    print("Lost due to no space in C4a")
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 8
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)


while S4b_leave_queue.length() > 0:
#for self.k in range(S4b_leave_queue.length()):
    #start_time = timeit.default_timer()
    self.Batch = S4b_leave_queue.pop()

    # 2) Check if it waits too long

    if self.Batch.Timer < (env.now() - 1):
        print("Waited too long 3")
        Lost_component_queue.add(self.Batch)
        for self.i in range(self.Batch.NrPlants):
            self.LostCrop = TLostCrop()
            self.LostCrop.LossTime = env.now()
            if (env.now() < 135) and (env.now() < 500):
                self.LostCrop.LossReason = 24
            LostCrops_queue.add(self.LostCrop)
        if (env.now() >= 135) and (env.now() <= 501):
            Finsihed_Batch_queue.add(self.Batch)
    # 3) Go through tree

    # if in right compartment
    elif self.Batch.PlacedInWrongCompartment == "":
        yield self.hold(self.DriveTime[4] * self.Batch.NoGutters)
        self.Batch.TransportTimeCum += self.DriveTime[4] *
            self.Batch.NoGutters
        self.Batch.TransportDistanceCum += self.distance[4] *
            self.Batch.NoGutters
        self.Batch.NumberTransportsCum += self.Batch.NoGutters
        self.Batch.Timer = env.now()
        Monitoring_queue.add(self.Batch)

    # Assign space
    elif self.Batch.PlacedInWrongCompartment == "C4a":
        self.Batch.PlacedInWrongCompartment = ""
        # check if ready for next stage
        if (dFDS1 + dFDS2 + dFDS3 + dFDS4b) < self.Batch.FDS:
            yield self.hold(self.DriveTime[3] * self.Batch.NoGutters)
            self.Batch.TransportTimeCum += self.DriveTime[3] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[3] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            self.Batch.Timer = env.now()
```

```
            Monitoring_queue.add(self.Batch)

        # Check for space in right compartment
        elif (GreenhouseC4a.Occupancy + self.Batch.NrPlants *
            self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[3]) <=
                                        (GreenhouseC4a.Capacity):
            yield self.hold(self.DriveTime[6] * self.Batch.NoGutters) #
                update this drive time
            self.Batch.TransportTimeCum += self.DriveTime[6] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[6] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            S4a_queue.add(self.Batch)
            GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[3]


            # Check for space in another acceptable compartment
        elif (GreenhouseC4b.Occupancy + self.Batch.NrPlants *
            self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[3]) <
                                        (GreenhouseC4b.Capacity):
            self.Batch.PlacedInWrongCompartment = "C4a"
            self.Batch.Timer = env.now()
            S4b_queue.add(self.Batch)
            GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[3]
            if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                print("Error is here 5")


        # Else batch is lost
        else:
            Lost_component_queue.add(self.Batch)
            print("Lost due to no space in C4a")
            for self.i in range(self.Batch.NrPlants):
                self.LostCrop = TLostCrop()
                self.LostCrop.LossTime = env.now()
                self.LostCrop.LossReason = 9
                LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
                Finsihed_Batch_queue.add(self.Batch)

    elif self.Batch.PlacedInWrongCompartment == "C5":
        self.Batch.PlacedInWrongCompartment = ""
            # check if ready for next stage
        if (1) < self.Batch.FDS:
            yield self.hold(self.DriveTime[4] * self.Batch.NoGutters)
            self.Batch.TransportTimeCum += self.DriveTime[4] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[4] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            self.Batch.Timer = env.now()
            Harvesting_queue.add(self.Batch)

        # Check for space in right compartment
```

```
            elif (GreenhouseC5.Occupancy + self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]) <=
                (GreenhouseC5.Capacity):
               self.Batch.PlacedInWrongCompartment = ""
               yield self.hold(self.DriveTime[6] * self.Batch.NoGutters) #
                   update this drive time
               self.Batch.TransportTimeCum += self.DriveTime[6] *
                   self.Batch.NoGutters
               self.Batch.TransportDistanceCum += self.distance[6] *
                   self.Batch.NoGutters
               self.Batch.NumberTransportsCum += self.Batch.NoGutters
               S5_queue.add(self.Batch)
               GreenhouseC5.Occupancy += self.Batch.NrPlants *
                   self.Batch.TrussesPerPlant / \
                                       self.Batch.TrussDensity[5]


            # Check for space in another acceptable compartment
            elif (GreenhouseC4b.Occupancy + self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]) <
                GreenhouseC4b.Capacity:
               self.Batch.PlacedInWrongCompartment = "C5"
               self.Batch.Timer = env.now()
               S4b_queue.add(self.Batch)
               GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                   self.Batch.TrussesPerPlant / \
                                       self.Batch.TrussDensity[5]
               if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                  print("Error is here 6")

               # Else batch is lost
            else:
               Lost_component_queue.add(self.Batch)
               print("Lost due to no place in C5")
               for self.i in range(self.Batch.NrPlants):
                  self.LostCrop = TLostCrop()
                  self.LostCrop.LossTime = env.now()
                  self.LostCrop.LossReason = 5
                  LostCrops_queue.add(self.LostCrop)
               if (env.now() >= 135) and (env.now() <= 501):
                  Finsihed_Batch_queue.add(self.Batch)


         else:
            print(self.Batch.PlacedInWrongCompartment)
            print("Unknown Batch lost 9")
            Lost_component_queue.add(self.Batch)
            for self.i in range(self.Batch.NrPlants):
               self.LostCrop = TLostCrop()
               self.LostCrop.LossTime = env.now()
               if (env.now() < 135) and (env.now() < 500):
                  self.LostCrop.LossReason = 24
               LostCrops_queue.add(self.LostCrop)
            if (env.now() >= 135) and (env.now() <= 501):
               Finsihed_Batch_queue.add(self.Batch)

      #print(timeit.default_timer() - start_time)

      while S5_leave_queue.length() > 0:
      #for self.m in range(S5_leave_queue.length()):
         #start_time = timeit.default_timer()
```

```
self.Batch = S5_leave_queue.pop()

# 2) Check if it waits too long
if self.Batch.Timer < (env.now() - 1):
   print("waited too long 4")
   Lost_component_queue.add(self.Batch)
   for self.i in range(self.Batch.NrPlants):
      self.LostCrop = TLostCrop()
      self.LostCrop.LossTime = env.now()
      if (env.now() < 135) and (env.now() < 500):
         self.LostCrop.LossReason = 24
      LostCrops_queue.add(self.LostCrop)
   if (env.now() >= 135) and (env.now() <= 501):
      Finsihed_Batch_queue.add(self.Batch)
# 3) Go through tree
else:
   # if in right compartment
   if self.Batch.PlacedInWrongCompartment == "C5":
      self.Batch.PlacedInWrongCompartment = ""
   if self.Batch.PlacedInWrongCompartment == "":
      yield self.hold(self.DriveTime[5] * self.Batch.NoGutters)
      self.Batch.TransportTimeCum += self.DriveTime[5] *
          self.Batch.NoGutters
      self.Batch.TransportDistanceCum += self.distance[5] *
          self.Batch.NoGutters
      self.Batch.NumberTransportsCum += self.Batch.NoGutters
      self.Batch.Timer = env.now()
      Harvesting_queue.add(self.Batch)

   else: # Not in right compartment
      # Assign space
      if self.Batch.PlacedInWrongCompartment == "C4b":
         self.Batch.PlacedInWrongCompartment = ""
         self.BatchArea = self.Batch.NrPlants *
             self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]
         # check if ready for next stage
         if (1) < self.Batch.FDS:
            yield self.hold(self.DriveTime[5] * self.Batch.NoGutters)
            self.Batch.TransportTimeCum += self.DriveTime[5] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[5] *
                self.Batch.NoGutters
            self.Batch.NumberTransportsCum += self.Batch.NoGutters
            self.Batch.Timer = env.now()
            Harvesting_queue.add(self.Batch)

         # Check for space in right compartment
         elif (GreenhouseC4b.Occupancy + self.Batch.NrPlants *
             self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4])
             < (GreenhouseC4b.Capacity):
            self.Batch.PlacedInWrongCompartment = ""
            GreenhouseC4b.Occupancy += self.Batch.NrPlants *
                self.Batch.TrussesPerPlant / \
                                self.Batch.TrussDensity[4]
            if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
               print("Error is here 7")
            yield self.hold(self.DriveTime[6] * self.Batch.NoGutters)
                # update this drive time
            self.Batch.TransportTimeCum += self.DriveTime[6] *
                self.Batch.NoGutters
            self.Batch.TransportDistanceCum += self.distance[6] *
```

```
                        self.Batch.NoGutters
                    self.Batch.NumberTransportsCum += self.Batch.NoGutters
                    S4b_queue.add(self.Batch)

                # Check for space in another acceptable compartment
                elif (GreenhouseC5.Occupancy + self.Batch.NrPlants *
                     self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4])
                    < (GreenhouseC5.Capacity):
                    self.Batch.Timer = env.now()
                    GreenhouseC5.Occupancy += self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / \
                                        self.Batch.TrussDensity[4]
                    self.Batch.PlacedInWrongCompartment = "C4b"
                    S5_queue.add(self.Batch)

                # Else batch is lost
                else:
                    Lost_component_queue.add(self.Batch)
                    print("Lost due to no place in C4b")
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 9
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)

            else:
                print(self.Batch.PlacedInWrongCompartment)
                print("Unknown Batch lost 11")
                Lost_component_queue.add(self.Batch)
                for self.i in range(self.Batch.NrPlants):
                    self.LostCrop = TLostCrop()
                    self.LostCrop.LossTime = env.now()
                    if (env.now() < 135) and (env.now() < 500):
                        self.LostCrop.LossReason = 24
                    LostCrops_queue.add(self.LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)

        #print(timeit.default_timer() - start_time)

class TTransplant1Machine(sim.Component):
    def setup(self, HandlingTime, LossRate):
        self.HandlingTime = HandlingTime
        self.LossRate = LossRate
        self.OperationalTime = np.zeros((730))
    def process(self):
        while True:
            while Transplant1_queue.length() == 0:
                yield self.standby()
            while Transplant1_queue.length() > 0:
                self.Batch = Transplant1_queue.pop()
                if self.Batch.Timer < (env.now() - 1):
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 31
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
```

```
                Finsihed_Batch_queue.add(self.Batch)
            else:
                yield self.hold(self.HandlingTime * self.Batch.NoTrays1)
                self.OperationalTime[int(env.now())] += self.HandlingTime *
                    self.Batch.NoTrays1
                self.Batch.Timer = env.now()
                Transplant1_done_queue.add(self.Batch)

class TTransplant2Machine(sim.Component):
    def setup(self, HandlingTime, LossRate):
        self.HandlingTime = HandlingTime
        self.LossRate = LossRate
        self.OperationalTime = np.zeros((730))
    def process(self):
        while True:
            while Transplant2_queue.length() == 0:
                yield self.standby()
            while Transplant2_queue.length() > 0:
                self.Batch = Transplant2_queue.pop()
                if self.Batch.Timer < (env.now() - 1):
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 32
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)
                else:
                    yield self.hold(self.HandlingTime * self.Batch.NoTrays2)
                    self.OperationalTime[int(env.now())] += self.HandlingTime *
                        self.Batch.NoTrays2
                    self.Batch.Timer = env.now()
                    Transplant2_done_queue.add(self.Batch)

class TMonitoringMachine(sim.Component):
    def setup(self, HandlingTime, LossRate):
        self.HandlingTime = HandlingTime
        self.LossRate = LossRate
        self.OperationalTime = np.zeros((730))
    def process(self):
        while True:
            while Monitoring_queue.length() == 0:
                yield self.standby()

            while Monitoring_queue.length() > 0:
                self.Batch = Monitoring_queue.pop()
                if self.Batch.Timer < (env.now() - 1):
                    Lost_component_queue.add(self.Batch)
                    for self.i in range(self.Batch.NrPlants):
                        self.LostCrop = TLostCrop()
                        self.LostCrop.LossTime = env.now()
                        self.LostCrop.LossReason = 33
                        LostCrops_queue.add(self.LostCrop)
                    if (env.now() >= 135) and (env.now() <= 501):
                        Finsihed_Batch_queue.add(self.Batch)
                else:
                    yield self.hold(self.HandlingTime * self.Batch.NoGutters)
                    self.OperationalTime[int(env.now())] += self.HandlingTime *
                        self.Batch.NoGutters
                    self.Batch.Timer = env.now()
```

```
                Monitoring_done_queue.add(self.Batch)

class THarvestingMachine(sim.Component):
   def setup(self, HandlingTime, LossRate):
      self.HandlingTime = HandlingTime
      self.LossRate = LossRate
      self.k = 0
      self.OperationalTime = np.zeros((730))
   def process(self):
      while True:
         while Harvesting_queue.length() == 0:
            yield self.standby()

         while Harvesting_queue.length() > 0:
            self.Batch = Harvesting_queue.pop()
            if self.Batch.Timer < (env.now() - 1):
               Lost_component_queue.add(self.Batch)
               for self.i in range(self.Batch.NrPlants):
                  self.LostCrop = TLostCrop()
                  self.LostCrop.LossTime = env.now()
                  self.LostCrop.LossReason = 34
                  LostCrops_queue.add(self.LostCrop)
            else:
               self.time = int(env.now())
               yield self.hold(self.HandlingTime * self.Batch.NoGutters)
               self.OperationalTime[int(env.now())] += self.HandlingTime *
                   self.Batch.NoGutters
               if (env.now() >= 135) and (env.now() <= 501):
                  self.Batch.NrPlants -= int(self.Batch.NrPlants *
                      self.Batch.LossRate / 100)
                  for self.i in range(int(self.Batch.NrPlants *
                      self.Batch.LossRate / 100)):
                     self.LostCrop = TLostCrop()
                     self.LostCrop.LossTime = env.now()
                     self.LostCrop.LossReason = 99
                     LostCrops_queue.add(self.LostCrop)

                  for self.i in range(self.Batch.NrPlants):
                     for self.k in range(int(self.Batch.TrussesPerPlant)):
                        self.HarvestedTruss = THarvestedTruss()
                        self.HarvestedTruss.TW = self.Batch.TW
                        self.HarvestedTruss.LOS = self.Batch.CurrentTime
                        self.HarvestedTruss.Revenue = self.HarvestedTruss.TW /
                            1000 * price[self.time] / 100
                        Harvested_trusses_queue.add(self.HarvestedTruss)
                        self.k += 1

            if (env.now() >= 135) and (env.now() <= 501):
               Finsihed_Batch_queue.add(self.Batch)

class TGreenhouseCompartment1(sim.Component):
   def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
      # self.FDR = FDR
      self.runtime = runtime
      self.Capacity = Capacity
      self.InterArrivalTime = sim.Constant(1)
      self.Day = self.InterArrivalTime.sample()
      self.day = 0
      self.Occupancy = 0
      self.FDR = FDR
      self.DLI = DLI
```

```
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        # while self.j < self.runtime:
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S1_queue.length()):
                self.Batch = S1_queue.pop()
                self.Batch.Timer = env.now() + self.Day
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                if self.Batch.PlacedInWrongCompartment == "C2":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[1]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[0, self.j] += self.Batch.NrPlants
                    GreenhouseC1.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
                    S1_leave_queue.add(self.Batch)
                elif self.Batch.FDS >= self.dFDS:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[0]
                    GreenhouseC1.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
                    S1_leave_queue.add(self.Batch)
                else:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[0]
                    S1_queue.add(self.Batch)
            # S1_queue[:][i].FDS += self.FDR[S1_queue[:][i].StartDate][i]
            # self.j += 1
            # print(timeit.default_timer() - start_time)
            yield self.hold(self.Day)
            OccupancyRates[0, self.j] = GreenhouseC1.Occupancy

class TGreenhouseCompartment2(sim.Component):
    def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
        # self.FDR = FDR
        self.runtime = runtime
        self.Capacity = Capacity
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()
        self.day = 0
        self.Occupancy = 0
        self.FDR = FDR
        self.DLI = DLI
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S2_queue.length()):
                self.Batch = S2_queue.pop()
                self.Batch.Timer = env.now() + self.Day
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                if self.Batch.PlacedInWrongCompartment == "C1":
```

```
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[0]
                self.Batch.TimeInWrongCompartment += 1
                WrongCompartment[1, self.j] += self.Batch.NrPlants
                GreenhouseC2.Occupancy -= self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[0]
                S2_leave_queue.add(self.Batch)
            elif self.Batch.FDS >= self.dFDS:
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[1]
                GreenhouseC2.Occupancy -= self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[1]
                S2_leave_queue.add(self.Batch)
            else:
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[1]
                S2_queue.add(self.Batch)

        # print(timeit.default_timer() - start_time)
        yield self.hold(self.Day)
        OccupancyRates[1, self.j] = GreenhouseC2.Occupancy

class TGreenhouseCompartment3(sim.Component):
    def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
        # self.FDR = FDR
        self.runtime = runtime
        self.Capacity = Capacity
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()
        self.day = 0
        self.Occupancy = 0
        self.FDR = FDR
        self.DLI = DLI
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S3_queue.length()):
                self.Batch = S3_queue.pop()
                self.Batch.Timer = env.now() + self.Day
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                if self.Batch.PlacedInWrongCompartment == "C4a":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[3]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[2, self.j] += self.Batch.NrPlants
                    GreenhouseC3.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]
                    S3_leave_queue.add(self.Batch)
                elif self.Batch.FDS >= self.dFDS:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[2]
                    GreenhouseC3.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]
                    S3_leave_queue.add(self.Batch)
                else:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
```

```
                    self.Batch.TrussDensity[2]
                S3_queue.add(self.Batch)

            yield self.hold(self.Day)
            OccupancyRates[2, self.j] = GreenhouseC3.Occupancy

class TGreenhouseCompartment4a(sim.Component):
    def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
        # self.FDR = FDR
        self.runtime = runtime
        self.Capacity = Capacity
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()
        self.day = 0
        self.Occupancy = 0
        self.FDR = FDR
        self.DLI = DLI
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S4a_queue.length()):
                self.Batch = S4a_queue.pop()
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                self.Batch.Timer = env.now() + self.Day

                if self.Batch.PlacedInWrongCompartment == "C3":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[2]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[3, self.j] += self.Batch.NrPlants
                    GreenhouseC4a.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[2]
                    S4a_leave_comp_queue.add(self.Batch) #

                elif self.Batch.PlacedInWrongCompartment == "C4b":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[4]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[3, self.j] += self.Batch.NrPlants
                    GreenhouseC4a.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]
                    S4a_leave_comp_queue.add(self.Batch) #

                elif self.Batch.FDS >= self.dFDS:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[3]
                    GreenhouseC4a.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]

                    if GreenhouseC4b.Capacity > (
                          GreenhouseC4b.Occupancy + self.Batch.NrPlants *
                              self.Batch.TrussesPerPlant /
                          self.Batch.TrussDensity[4]):
                        self.Batch.PlacedInWrongCompartment = ""
                        S4b_queue.add(self.Batch)
                        GreenhouseC4b.Occupancy += self.Batch.NrPlants *
```

```
                    self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[4]
                if GreenhouseC4b.Occupancy >= GreenhouseC4b.Capacity:
                    print("No place in 4b")

            elif GreenhouseC4a.Capacity > (
                    GreenhouseC4a.Occupancy + self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant /
                    self.Batch.TrussDensity[4]):
                self.Batch.PlacedInWrongCompartment = "C4b"
                S4a_queue.add(self.Batch)
                GreenhouseC4a.Occupancy += self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / \
                                    self.Batch.TrussDensity[4]
            else:
                print("No place in 4b")

                for self.i in range(self.Batch.NrPlants):
                    LostCrop = TLostCrop()
                    LostCrop.LossTime = env.now()
                    LostCrop.LossReason = 9
                    LostCrops_queue.add(LostCrop)
                if (env.now() >= 135) and (env.now() <= 501):
                    Finsihed_Batch_queue.add(self.Batch)
        else:
            self.Batch.TW += self.DLI[self.j] * self.LUE /
                self.Batch.TrussDensity[3]
            S4a_queue.add(self.Batch)

        yield self.hold(self.Day)
        OccupancyRates[3, self.j] = GreenhouseC4a.Occupancy

class TGreenhouseCompartment4b(sim.Component):
    def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
        # self.FDR = FDR
        self.runtime = runtime
        self.Capacity = Capacity
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()
        self.day = 0
        self.Occupancy = 0
        self.FDR = FDR
        self.DLI = DLI
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S4b_queue.length()):
                self.Batch = S4b_queue.pop()
                self.Batch.Timer = env.now() + self.Day
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                if self.Batch.PlacedInWrongCompartment == "C4a":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[3]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[4, self.j] += self.Batch.NrPlants
                    GreenhouseC4b.Occupancy -= self.Batch.NrPlants *
```

```
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[3]
                S4b_leave_queue.add(self.Batch)
            elif self.Batch.PlacedInWrongCompartment == "C5":
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[5]
                self.Batch.TimeInWrongCompartment += 1
                WrongCompartment[4, self.j] += self.Batch.NrPlants
                GreenhouseC4b.Occupancy -= self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]
                S4b_leave_queue.add(self.Batch)
            elif self.Batch.FDS >= self.dFDS:
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[4]
                S4b_leave_queue.add(self.Batch)
                GreenhouseC4b.Occupancy -= self.Batch.NrPlants *
                    self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]

            else:
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[4]
                S4b_queue.add(self.Batch)

        yield self.hold(self.Day)
        OccupancyRates[4, self.j] = GreenhouseC4b.Occupancy

class TGreenhouseCompartment5(sim.Component):
    def setup(self, FDR, DLI, LUE, runtime, Capacity, dFDS):
        # self.FDR = FDR
        self.runtime = runtime
        self.Capacity = Capacity
        self.InterArrivalTime = sim.Constant(1)
        self.Day = self.InterArrivalTime.sample()
        self.day = 0
        self.Occupancy = 0
        self.FDR = FDR
        self.DLI = DLI
        self.dFDS = dFDS
        self.LUE = LUE

    def process(self):
        for self.j in range(self.runtime):
            self.day += self.Day
            # start_time = timeit.default_timer()
            for self.i in range(S5_queue.length()):
                self.Batch = S5_queue.pop()
                self.Batch.Timer = env.now() + self.Day
                self.Batch.CurrentTime += self.Day
                self.Batch.FDS += self.FDR[self.j]
                if self.Batch.PlacedInWrongCompartment == "C4b":
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[4]
                    self.Batch.TimeInWrongCompartment += 1
                    WrongCompartment[5, self.j] += self.Batch.NrPlants
                    GreenhouseC5.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[4]
                    S5_leave_queue.add(self.Batch)
                elif self.Batch.FDS >= self.dFDS:
                    self.Batch.TW += self.DLI[self.j] * self.LUE /
                        self.Batch.TrussDensity[5]
                    GreenhouseC5.Occupancy -= self.Batch.NrPlants *
                        self.Batch.TrussesPerPlant / self.Batch.TrussDensity[5]
```

```
                S5_leave_queue.add(self.Batch)
            else:
                self.Batch.TW += self.DLI[self.j] * self.LUE /
                    self.Batch.TrussDensity[5]
                S5_queue.add(self.Batch)
        yield self.hold(self.Day)
        OccupancyRates[5, self.j] = GreenhouseC5.Occupancy

# ============================================================================
# ============ Perform simulation ============================================
# ============================================================================
# still under assumption of 4a and 4b being one compartment
env = sim.Environment(trace=False, time_unit='days', random_seed=1234567)
env.background_color('20%gray')

# Define Queues for the areas
S1_queue = sim.Queue("Greenhouse section 1")
S2_queue = sim.Queue("Greenhouse section 2")
S3_queue = sim.Queue("Greenhouse section 3")
S4a_queue = sim.Queue("Greenhouse section 4a")
S4b_queue = sim.Queue("Greenhouse section 4b")
S5_queue = sim.Queue("Greenhouse section 5")

S1_leave_queue = sim.Queue("Greenhouse section 1 leave")
S2_leave_queue = sim.Queue("Greenhouse section 2 leave")
S3_leave_queue = sim.Queue("Greenhouse section 3 leave")
S4a_leave_queue = sim.Queue("Greenhouse section 4a leave to 4b")
S4a_leave_comp_queue = sim.Queue("Greenhouse section 4a leave new comp")
S4b_leave_queue = sim.Queue("Greenhouse section 4b leave")
S5_leave_queue = sim.Queue("Greenhouse section 5 leave")

# Define Machine queues
Sowing_queue = sim.Queue("Sowing machine")
Transplant1_queue = sim.Queue("Transplant1 machine")
Transplant2_queue = sim.Queue("Transplant2 machine")
Monitoring_queue = sim.Queue("Monitoring machine")
Harvesting_queue = sim.Queue("Harvesting machine")
Sorting_queue = sim.Queue("Sorting machine")
Harvested_trusses_queue = sim.Queue("Harvested plants")

Sowing_done_queue = sim.Queue("Sowing machine")
Transplant1_done_queue = sim.Queue("Transplant1 machine")
Transplant2_done_queue = sim.Queue("Transplant2 machine")
Monitoring_done_queue = sim.Queue("Monitoring machine")
Harvesting_done_queue = sim.Queue("Harvesting machine")
Sorting_done_queue = sim.Queue("Sorting machine")

# Define Buffers
Tray1_queue = sim.Queue("Tray 1 buffer")
Tray2_queue = sim.Queue("Tray 2 buffer")
Gutter_queue = sim.Queue("Gutter buffer")

LostCrops_queue = sim.Queue("Lost crops")
Lost_component_queue = sim.Queue("Lost component")
Finsihed_Batch_queue = sim.Queue("Finished Batch")

SowingGen = TSowingGenerator(SowingRate = SowingRate * Factor_to_sowing_rate,
    SowingTime = Sowing_time, LOS = LOS, runtime = runtime,
                    number_of_trusses = number_of_trusses, batchsize =
                        batchsize, plants_per_gutter =
                    plants_per_gutter, plants_per_tray_1 = plants_per_tray_1,
```

```
                        plants_per_tray_2 =
                    plants_per_tray_2, TrussDensity = truss_density, LossRate =
                        LossRate)
TrayCropGreenhouseShuttle = TTrayCropGreenhouseShuttle(DriveTime = DriveTime,
    distance=distance, runtime = runtime)
TrayGreenhouseCropShuttle = TTrayGreenhouseCropShuttle(DriveTime = DriveTime,
    distance=distance,runtime = runtime)
GutterGreenhouseCropShuttle = TGutterGreenhouseCropShuttle(DriveTime = DriveTime,
    distance=distance,runtime = runtime)
GutterCropGreenhouseShuttle = TGutterCropGreenhouseShuttle(DriveTime = DriveTime,
    distance=distance,runtime = runtime)

GreenhouseC1 = TGreenhouseCompartment1(FDR = FDR[0], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[0], dFDS = dFDS1)
GreenhouseC2 = TGreenhouseCompartment2(FDR = FDR[1], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[1], dFDS = dFDS1 + dFDS2)
GreenhouseC3 = TGreenhouseCompartment3(FDR = FDR[2], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[2], dFDS = dFDS1 + dFDS2 + dFDS3)
GreenhouseC4a = TGreenhouseCompartment4a(FDR = FDR[3], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[3], dFDS = dFDS1 + dFDS2 + dFDS3 + dFDS4a)
GreenhouseC4b = TGreenhouseCompartment4b(FDR = FDR[4], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[4], dFDS = dFDS1 + dFDS2 + dFDS3 + dFDS4a +
    dFDS4b)
GreenhouseC5 = TGreenhouseCompartment5(FDR = FDR[5], DLI = DLI, LUE = LUE,
    runtime = runtime, Capacity = Area[5], dFDS = 1)

Transplant1Machine = TTransplant1Machine(HandlingTime = Transplant1_time,
    LossRate = LossRate)
Transplant2Machine = TTransplant2Machine(HandlingTime = Transplant2_time,
    LossRate = LossRate)
MonitoringMachine = TMonitoringMachine(HandlingTime = Monitoring_time, LossRate =
    LossRate)
HarvestingMachine = THarvestingMachine(HandlingTime = Harvesting_time, LossRate =
    LossRate)

# define queues to show on the simulation monitor
P0 = sim.AnimateMonitor(S5_queue.length, x=10, y=560, width=460, height=80,
    horizontal_scale=1.3, vertical_scale=0.2)
P1 = sim.AnimateMonitor(S2_queue.length, x=10, y=450, width=460, height=80,
    horizontal_scale=1.3, vertical_scale=0.2)
P2 = sim.AnimateMonitor(S3_queue.length, x=10, y=340, width=460, height=80,
    horizontal_scale=1.3, vertical_scale=0.2)
P3 = sim.AnimateMonitor(S4a_queue.length, x=10, y=230, width=460, height=80,
    horizontal_scale=1.3, vertical_scale=0.2)
P4 = sim.AnimateMonitor(S4b_queue.length, x=10, y=120, width=460, height=80,
    horizontal_scale=1.3, vertical_scale=0.2)
P5 = sim.AnimateMonitor(Harvested_trusses_queue.length, x=10, y=10, width=460,
    height=80, horizontal_scale=1.3,
                    vertical_scale=0.2)

Q0 = sim.AnimateMonitor(Transplant1_queue.length, x=510, y=560, width=460,
    height=80, horizontal_scale=1.3,
                    vertical_scale=1)
Q1 = sim.AnimateMonitor(Transplant2_queue.length, x=510, y=450, width=460,
    height=80, horizontal_scale=1.3,
                    vertical_scale=1)
Q2 = sim.AnimateMonitor(Monitoring_queue.length, x=510, y=340, width=460,
    height=80, horizontal_scale=1.3,
                    vertical_scale=1)
Q3 = sim.AnimateMonitor(Harvesting_queue.length, x=510, y=230, width=460,
    height=80, horizontal_scale=1.3, vertical_scale=1)
```

```
Q4 = sim.AnimateMonitor(Monitoring_done_queue.length, x=510, y=120, width=460,
    height=80, horizontal_scale=1.3,
                vertical_scale=0.5)
Q5 = sim.AnimateMonitor(Transplant2_done_queue.length, x=510, y=10, width=460,
    height=80, horizontal_scale=1.3,
                vertical_scale=0.5)

# simulate
env.modelname("Plants animation")
env.animate(False)
env.speed(10)
env.run(runtime) # run 2 years

OC1 = np.zeros((runtime))
OC2 = np.zeros((runtime))
OC3 = np.zeros((runtime))
OC4a = np.zeros((runtime))
OC4b = np.zeros((runtime))
OC5 = np.zeros((runtime))

OCS = np.zeros((runtime))
OCT1 = np.zeros((runtime))
OCT2 = np.zeros((runtime))
OCM = np.zeros((runtime))
OCE = np.zeros((runtime))
OCH = np.zeros((runtime))

OCSD = np.zeros((runtime))
OCT1D = np.zeros((runtime))
OCT2D = np.zeros((runtime))
OCMD = np.zeros((runtime))
OCHD = np.zeros((runtime))

LE1 = np.zeros((runtime))
LE2 = np.zeros((runtime))
LE3 = np.zeros((runtime))
LE4a = np.zeros((runtime))
LE4b = np.zeros((runtime))
LE5 = np.zeros((runtime))

for i in range(135,500):
   OCT1[i] = Transplant1_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCT2[i] = Transplant2_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCM[i] = Monitoring_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCH[i] = Harvesting_queue.length_of_stay.slice(i,i+1,runtime).maximum()

   OCSD[i] = Sowing_done_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCT1D[i] = Transplant1_done_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCT2D[i] = Transplant2_done_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   OCMD[i] = Monitoring_done_queue.length_of_stay.slice(i,i+1,runtime).maximum()

   LE1[i] = S1_leave_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   LE2[i] = S2_leave_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   LE3[i] = S3_leave_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   LE4b[i] = S4b_leave_queue.length_of_stay.slice(i,i+1,runtime).maximum()
   LE5[i] = S5_leave_queue.length_of_stay.slice(i,i+1,runtime).maximum()

Harvested_tomatoes = np.zeros((runtime))
LostCrops = np.zeros((runtime))
HarvestedWeight = np.zeros((runtime))
HarvestedWeight_cum = np.zeros((runtime))
```

```
Revenue_day = np.zeros((runtime))
LostCropsStart = LostCrops_queue.length.get(135)
for i in range(135,501):
   LostCrops[i] = LostCrops_queue.length.get(i) - LostCropsStart
   Harvested_tomatoes[i] = Harvested_trusses_queue.length.get(i) -
       Harvested_trusses_queue.length.get(i-1)

LossReason = np.zeros((LostCrops_queue.length()))
TransportTimeLost = 0
TransportDistanceLost = 0
TransportNoLost = 0

for i in range((Finsihed_Batch_queue.length())):
   Batch = Finsihed_Batch_queue.pop()
   TransportTimeLost += Batch.TransportTimeCum
   TransportDistanceLost += Batch.TransportDistanceCum
   TransportNoLost += Batch.NumberTransportsCum

SowingMachineOperational = SowingGen.OperationalTime
Transplant1Operational = Transplant1Machine.OperationalTime
Transplant2Operational = Transplant2Machine.OperationalTime
MonitoringMachineOperational = MonitoringMachine.OperationalTime
HarvestingMachineOperational = HarvestingMachine.OperationalTime

Lost_due_to_waiting = 0
for i in range(len(LossReason)):
   LostCrop = LostCrops_queue.pop()
   if (LostCrop.LossTime > 135) and (LostCrop.LossTime < 501):
      LossReason[i] = LostCrop.LossReason
      if (LossReason[i] > 20) and (LossReason[i] < 25):
         Lost_due_to_waiting += batchsize

TrussWeight = np.zeros((Harvested_trusses_queue.length()))
Revenue = np.zeros((Harvested_trusses_queue.length()))
for i in range(len(TrussWeight)):
   HarvestedTruss = Harvested_trusses_queue.pop()
   TrussWeight[i] = HarvestedTruss.TW
   Revenue[i] = HarvestedTruss.Revenue

k = 0
for i in range(135,500):
   for j in range(int(Harvested_tomatoes[i])):
      k += 1
      if k >= len(TrussWeight):
         break
      HarvestedWeight[i] += TrussWeight[k] / 1000

for i in range(135,501):
   HarvestedWeight_cum[i] = HarvestedWeight[i] + HarvestedWeight_cum[i-1]
   Revenue_day[i] = Revenue_day[i-1] + HarvestedWeight[i] * price[i]

Sowed_cum_Actual = np.zeros((runtime))
Sowed_cum_Calculated = np.zeros((runtime))
for i in range(135,501):
   Sowed_cum_Actual[i] = Sowed_cum_Actual[i-1] + Sowed[i]
   Sowed_cum_Calculated[i] = Sowed_cum_Calculated[i-1] + SowingRate[i]

OccupancyRateGreenhouse = 100 * (OccupancyRates[0,:] * 2 + OccupancyRates[1,:] +
    OccupancyRates[2,:]
                                 + OccupancyRates[3,:]+ OccupancyRates[4,:]+
                                     OccupancyRates[5,:]) / (np.sum(Area))
```

```
# fig 1: Occupancy rate greenhouse: per compartment
plt.figure(90, figsize = [16,8])
plt.plot(s, OccupancyRates[0,:] / Area[0] * 100, s, OccupancyRates[1,:] / Area[1]
    * 100, s, OccupancyRates[2,:] / Area[2] * 100,
       s, OccupancyRates[3,:] / Area[3] * 100, s, OccupancyRates[4,:] / Area[4] *
           100, s, OccupancyRates[5,:] / (Area[5]) * 100,
       s, OccupancyRateGreenhouse)
plt.title("Occupancy rate greenhouse")
plt.legend(["comp 1 ", "comp 2", "comp 3", "comp 4a", "comp 4b", "comp 5",
    "Total"], loc = "upper right")
plt.xlim(s[135], s[500])
plt.ylim([0,100])
plt.ylabel("%")
plt.xlabel("Date")

# fig 2: Occupancy rate labour / robots: Length of queue
plt.figure(91, figsize = [16,8])
plt.plot(s, OCT1 * 24, s, OCT2 * 24, s, OCM * 24, s, OCH * 24)
plt.title("Max waiting time before handling")
plt.legend(["Transplant 1", "Transplant 2", "Monitor", "Harvest"], loc = "upper
    right")
plt.xlim(s[135], s[500])
plt.ylabel("Hours")
plt.xlabel("Date")

# fig 3: Tasks performed in time: Destroyed crops
WrongCompartment_cum = np.zeros((runtime))
for i in range(135,501):
   WrongCompartment_cum[i] = WrongCompartment_cum[i-1] + WrongCompartment[0][i] +
       WrongCompartment[1][i] + WrongCompartment[2][i] \
                       + WrongCompartment[3][i] + WrongCompartment[4][i] +
                           WrongCompartment[5][i]
plt.figure(92, figsize = [16,8])
plt.plot(s, LostCrops, s, Sowed_cum_Actual, s, WrongCompartment_cum)
plt.title("Plants lost per day")
plt.xlim(s[135], s[500])
plt.ylabel("-")
plt.legend(["Lost plants cumulative", "Actual sowed cumulative", "Days plant in
    wrong compartment cumulative"])
plt.xlabel("Date")

# fig 4: Production per m^2
plt.figure(93, figsize = [16,8])
plt.plot(s, HarvestedWeight_cum / np.sum(Area))
plt.xlim(s[135], s[500])
plt.title("Cumulative production")
plt.ylabel("kg/m^2")
plt.xlabel("Date of harvest")

# fig 5: Truss weight at end of cycle
plt.figure(94, figsize = [16,8])
plt.boxplot(TrussWeight, showfliers = False, vert=False)
x = np.random.normal(1, 0.04, size = len(TrussWeight))
#plt.scatter(TrussWeight, x)
plt.title("Truss weight at harvest")
plt.ylabel("")
plt.xlabel("gram")

# fig 6: Truss weight at end of cycle
binshist = np.arange(0,50,1)
```

```
plt.figure(95, figsize = [16,8])
plt.hist(LossReason, bins = binshist)
plt.title("Reason for loss")
plt.ylabel("Frequency")
plt.xlabel("Reason number")

#plt.figure(96, figsize = [16,8])
#plt.plot(s, OCSD * 24, s, OCT1D * 24, s, OCT2D * 24, s, OCMD * 24)
#plt.title("Max waiting time before pick up after handling")
#plt.legend(["Sowing","Transplant 1", "Transplant 2", "Monitor", "Harvest"], loc
    = "upper right")
#plt.xlim(s[135], s[500])
#plt.ylabel("Hours")
#plt.xlabel("Date")

plt.figure(97, figsize = [16,8])
plt.plot(s, LE1 * 24, s, LE2 * 24, s, LE3 * 24, s, LE4b, s, LE5)
plt.title("Max waiting time before pick up from compartment")
plt.legend(["Comp 1", "Comp 2", "Comp 3", "Comp 4b", "Comp 5"], loc = "upper
    right")
plt.xlim(s[135], s[500])
plt.ylabel("Hours")
plt.xlabel("Date")

# Revenue per m^2
plt.figure(98, figsize = [16,8])
plt.plot(s, Revenue_day / 100 / (np.sum(Area)))
plt.xlim(s[135], s[500])
plt.title("Cumulative turnover per m^2")
plt.ylabel("€/m^2")
plt.xlabel("Date of harvest")

plt.figure(99, figsize = [16,8])
plt.plot(s, SowingMachineOperational * 24, s, Transplant1Operational * 24, s,
    Transplant2Operational * 24, s, MonitoringMachineOperational * 24, s,
    HarvestingMachineOperational * 24)
plt.title("Operational hours per day")
plt.legend(["Sowing","Transplant 1", "Transplant 2", "Monitor", "Harvest"], loc =
    "upper right")
plt.xlim(s[135], s[500])
plt.ylim(0,24)
plt.ylabel("Hours")
plt.xlabel("Date")


print("")
#print("Configuration")
#print("Light profile", Light)
#print("Temperature profile", Temperature_strategy)
#print("Loss of crops", LossRate, "%")
#print("Price curve", price_curve)
#print("")
print("Output KPI's")
print("Occupancy median rate greenhouse",
    np.median(OccupancyRateGreenhouse[135:500]), "%")
print("Lost crops", np.max(LostCrops), "Percentage lost",
    np.max(LostCrops)/np.max(Sowed_cum_Actual) * 100)
print("1 year production per m^2", np.max(HarvestedWeight_cum) / (np.sum(Area)),
    "[kg/m^2]")
print("Total Turnover", np.max(Revenue_day / 100 / np.sum(Area)), "euro/m^2")
#print("Sensitivity", (np.max(Revenue_day / 100 / np.sum(Area)) -
```

```
    76.87296593778537) / 76.87296593778537)
print('')
print('')
print("Transport Time", TransportTimeLost)
print("Transport Distance", TransportDistanceLost)
print("Transport No", TransportNoLost)

plt.show()
```


TU Delft


RIDDER