UNIVERSITA' DEGLI STUDI DI VERONA

DEPARTMENT OF

COMPUTER SCIENCE

GRADUATE SCHOOL OF

NATURAL SCIENCE AND ENGINEERING

DOCTORAL PROGRAM IN

COMPUTER SCIENCE

WITH THE FINANCIAL CONTRIBUTION OF

Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU

Cycle / year 2022

TOWARDS A DIGITAL TWIN FOR AGRICULTURE:

MODELING OF COMPLEX PROCESSES FOR MONITORING, PREDICTION AND CONTROL IN GREENHOUSE FARMING

S.S.D. INF/01

Coordinator: Prof. FERDINANDO CICALESE

Tutor: Prof TIZIANO VILLA

Prof DAVIDE QUAGLIA

Doctoral Student: Dott ELIA BRENTAROLLI

Abstract

Ten years ago, industrial processes underwent a significant change due to the prominent application of digital technologies in various forms, which greatly helped boost productivity and safety. In recent times, digital technologies have reached the agricultural sector, where they appear to be the solution for higher yields, better quality, and resource optimization. Despite such premise, however, nowadays, digital technologies are poorly used in this field, as they are often simply relegated to a way of collecting and processing data. This research shows that benefits provided by computer science span over ones already used by integrating advanced modeling and control methodologies from systems engineering to monitor, predict, and control complex processes in a greenhouse context. First, we address and solve the problem of microclimatic variability within different zones by developing a modeling technique based on soft sensors to predict location-specific temperature starting from a centralized reading, thus removing the need for physical sensors. Secondly, throughout the usage of Hybrid Automata, we re-implement the well-known tomato model TOMGRO using the mentioned formalism alongside a pathogen model for Oidium Lycopersici, which, thanks to automata formalism, we manage to compose with the tomato model. Next, we optimize control over such automata thanks to the supervisory control theory, which allows us to automatically synthesize supervisors for a system given the requirement, or by taking an already existing parametric controller and optimize it via simulation. Finally, we put all the pieces together using soft sensors to specialize climatic readings to act as input to different pairs of tomato-pathogen automata dislocated in a field, where the pathogen can grow and spread to neighbors, propagating the infection. Meanwhile, the supervisor tries to keep the temperature value within a favorable range for crops. The presented models have been validated using actual data retrieved from fields, and simulation models' outputs have been proved to be trustworthy by comparing results with actual ones retrieved from past experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Agriculture and computer science have never been as close as they were in the last few years. With the increasing demand for agricultural goods in terms of quantity and quality and the constant need to reduce chemical treatments such as fertilizers and pesticides, the exploitation of computer science technologies considerably helped achieve control and optimization of agricultural processes. Despite this, Computer Science and Agronomy nowadays remain two separate worlds, as the former is mainly devoted to providing data needed by the latter. In contrast, the contribution of Computer Science can go much further due to its historical background concerning modeling and control methodologies, that could prove extremely beneficial to agriculture.

In the last decade, however, a similar situation was faced when the Industry and Computer Science worlds met to create what today is called Industry 4.0 [3]. Although the term, originally introduced back in 2011, had the purpose of indicating the application of Cyber-Physical System (CPS) concepts in the Industrial Domain, the idea quickly grew popular and gave start to a new research field where scientists tried to apply other concepts known in Computer Science, beside CPS, to the industrial domain in order to further improve the newborn conjunction between these two, previously separated, worlds. For example, concepts such as Electronic Design Automation (EDA) [4], simulation [5], or cybersecurity [6] were also applied to the industrial sphere and proved to be successful research fields. This situation is represented in figure 1.1: up to today, researchers limited themselves to using only knowledge related to sensing and data processing; contrarily, computer science's true strength and knowledge lies at a much higher level of abstraction and problem formulation, which is however not considered by agronomist due to the different study background.

The main objective of this thesis is to bridge computer science and agronomy to establish a different modeling methodology that leverages computational modeling, automata formalism, and data-driven techniques to simulate, predict, and improve control strategies in crop growth processes, taking inspiration from Industry 4.0 and adapt similar principles to agricultural systems. Automata theory [7] in particular seems to be a promising approach for plant and disease modeling, as they can easily replace the traditional way used to describe an agronomic model, constituted by relational diagrams [8], used to display relations between various state variables, and plain equations which rule the evolution of state variables and, in some cases, also to express a change of state. However, such description lacks formality, which significantly limits how the model can be used due to factors such as ambiguous writing, which may cause errors in the manual translation into source code, the impossibility of performing verification of specific properties on the system, or difficulty on combining different models into a more complex one without rewriting them into a single, bigger, monolithic model. We believe that automata formalism can easily solve all the mentioned problems and others while being the best formalism to use due to its simplicity and expressive power. Furthermore, by using automata formalism, we can access more than 30 years of theories, such as minimization, composition, and many more. Moreover, automata can also be synthesized automatically into executable code for simulation and control, whereas traditional models do not. Using such formalisms and theories, we aim to provide expert in agronomic science with a new set of tools they can use to improve the overall quality of their work, such as achieving higher agricultural yields, reduced environmental impact, and more precise resource management.

To prove our claims, we apply automata-based modeling to the context of greenhouse farming, as the closed environment offers a much more interesting challenge for monitoring and control with respect to the open field where, instead, there is no method to change climatic conditions with actions. The considered case study comprehends the main elements constituting the greenhouse environment where crops grow: the climatic conditions inside the greenhouse, the crops growing

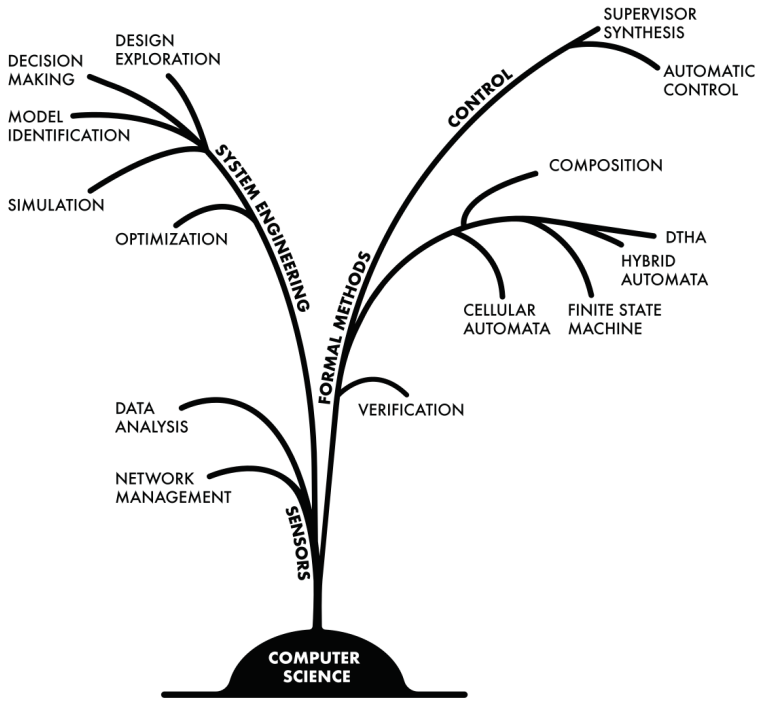


Figure 1.1: An artistic representation of (part of) the knowledge Computer Science offers in matter of system modeling.

in it, the possible pathogens spreading, and finally, the feedback control actions have on the crops. For each considered aspect, we examine its state of the art, address limitations of current solutions, and propose new ones based on computer science and system engineering methodology.

First, we examine the current status of greenhouse environmental monitoring, discovering how, in traditional modeling, microclimatic heterogeneity is often left behind in favor of a simplified model where environmental conditions are uniform. This factor is a big issue that must be addressed, as such differences may reach significant values (even more than $5°C$), leading to irregular crop growth and the possible appearance of pathogens. Therefore, our work aims to solve this situation by providing a methodology that can be computationally efficient, scalable, and capable of deployment in diverse agricultural settings to address and solve the problem of microclimatic heterogeneity within a greenhouse. We do so by implementing a soft sensors-based technique for cost-effective data acquisition at various points of interest in a greenhouse, which proved to be very precise while maintaining low costs, by combining data gathered at fixed locations and data obtained by persistent weather stations via linear regression, to develop models capable of estimating weather condition at a specific location in function of data from the fixed station at any time without the need of a physical device.

Next, we shift our focus to crop models and their descriptions. To avoid developing a crop model from scratch, we take advantage of the vast agronomic literature available and chose a well-known model for our work. Therefore, our choice falls on the tomato model TOMGRO, a model present in literature since the early '90s that has been intensely reviewed and examined throughout the years. Starting from such a model, described only using differential equation and common language, we leverage the descriptive power discrete-time hybrid automata (DTHA) provide us to derive three separate models, each one representing a critical growth factor for the tomato crop, these being biomass, leaf area index (LAI), and node development. Following the identification phase, we formalize each sub-model independently to remove ambiguity while also allowing all available formal techniques, not the last supervisory control theory, to study models' properties further. Subsequently, we put our automata under a parameter identification procedure, using data provided by the Department of Agriculture, Food and Environment of the University of Pisa, which is performed not on the entire system, as traditional methods would do, but rather on singles automata thanks to the compositional property of used formalism. Finally, we simulate the three automata together, taking strength in the synchronization property, to show how the original model could be recomposed very easily, showing that it is possible to study each sub-model independently and apply the obtained results to the total model without any additional effort.

Following the adaption of the crop model, we shift our attention to pathogens that could infect crops. Following the indication of experts, we focused our research on *Oidium Lycopersici*, a mold commonly known to infect tomato leaves and cover them with a white veil that stops photosynthetic activities. A model for such disease is identified in the literature, and just like for the crop's model, we show how, by starting from a pathogen model described in a research paper, we can derive its DTHA counterpart, which shares the same behavior as the original but with the addition of being able to compose with other automata thanks to the formalism used. As such, we show that two completely different models, one for the plant and one for the pathogen, can be put together in simulation as if they were a single model while also obtaining trustworthy results. Furthermore, we also extend the developed model by using automata's innate ability to compose with each other to add a spatial component to our system and allow a pathogen to spread onto nearby crops and propagate the infection.

Successively, we focus our research on greenhouses' control techniques, for which two studies are conducted. First, we showcase how, from the crop model, a new control strategy can be obtained via automatic synthesis using the control supervisory theory (CST) implemented within the ESCET tool to generate the controller's model to enforce desired properties on the system. In order to do so, we first develop a simplistic model of how the temperature inside a greenhouse evolves and how control actions affect it, therefore resulting in the creation of a simple greenhouse model that estimates the internal temperature in function of external conditions (external temperature and solar radiation) and the angle ruling windows opening, which regulates the airflow and help dissipating the heat and is controlled itself by a separate component which receives command by a controller. Following the modeling part, we perform an abstraction procedure to transform DTHA into FSA, allowing us to use ESCET's supervisory control synthesis, which only accepts these kinds of automata as input of the procedure, and to obtain an automaton representing our supervisor. Finally, we applied the resulting supervisor directly onto the original DTHA model by composing it alongside the greenhouse system and the crop model to showcase its functionality.

The second study we perform on control techniques, instead, focuses on improving already existing control strategies, often relying on user-defined parameters to work optimally. To do so, we show how we can use our model to explore the state space of possible parameter combinations for a controller's variables and set points in order to search which one is the best for a group of input traces. In this way, we provide end users with a new set of knowledge to help them in the decision-making process. Furthermore, we also showcase how this approach can be extended by iteratively searching for the best values while the crop is still growing by using forecast data, thus implementing a Model Predictive Control (MPC) approach for decision-making. As in previous cases, we start here too by modeling the controller using automata and composing it with our system; next, we use the ESCET tool again, but this time to perform code synthesis and obtain an executable source code perfectly representing our complete system. From there, we perform simulation on multiple data traces to perform the search process.

Finally, all solutions are assembled in a single complex system representing the entire greenhouse. Estimations provided by our climatic model are connected with soft sensors to estimate microclimatic differences, allowing the use of multiple climatic inputs to different crop-pathogen couples placed in a spatial grid to model the evolutions of multiple plants in different positions within the greenhouse. Moreover, pathogen models are also connected so that the infection could spread from a starting host to nearby crops, just like it would happen in reality. Meanwhile, crop data are used as input for the control model responsible for changing window opening policies. In the end, we manage to simulate in both space and time the evolution of the entire greenhouse system, made possible only thanks to the usage of formal models that allowed us to easily apply the various results obtained throughout the years (simulation, composition, code generation, and more), at the sole price of the effort done to translate models from their original writing to the computer science formalism.

This document is divided as follows: Chapter 2 provides all knowledge needed to understand the presented work fully; Chapter 3 discusses the current state of the art, presents nowadays solutions for problems faced during this work and addresses differences and limitation of such solutions concerning proposed ones. Chapters from 4 to 8 describe in detail the work done for each objective: Chapter 4 shows how soft sensor modeling was applied to greenhouse microclimatic monitoring and results obtained; Chapter 5 explains how the tomato model TOMGRO was rewritten as a *discrete-time hybrid automaton*, while Chapter 6 describes the same procedure applied for a disease model; Chapter 7 describes step performed to create a new control strategy starting from developed models and specific requirements; Chapter 8, on the other hand, implements an already existing control strategy and optimize it by suggesting end user the best parameters to input in the system. Finally, Chapter 9 concludes this research and summarizes the obtained results.

Chapter 2

Background

In this chapter, we will provide all the basic knowledge in crop modeling and automata theory needed by readers to fully understand the terminology and tools used in this thesis.

2.1 Environment monitoring in the Greenhouse Context

As biological beings, crops are heavily influenced by the environment surrounding them. This factor will also be seen in future sections, where climatic parameters will become inputs to both models for crops and pathogens. However, such parameters represent only a subset of actual environmental parameters in the greenhouse context, which still impact plant behavior and development. The main environmental parameters to monitor are Solar Radiation and CO_2 concentration, both used to perform the photosynthetic process needed by crops to live, as well as temperature and relative humidity (RH), which act both as a limiting factor for crops while also ruling the life cycle of possible disease.

Besides already mentioned environmental parameters, we can also add the need to monitor the so-called *Abiotic Constrains* (also known as*Abiotic stressors* [9]), i.e., all those environmental parameters which cause and abiotic stress and therefore trigger morphological, physiological and biochemical reactions that affect crops' growth negatively [10]. On a similar note, we also find *Biotic stressors*, which are the biological factors (such as insects, pest illnesses) hindering a crop's development [1]. Abiotic stressors can be divided mainly into two categories: soil properties, such as salinity, nutrient concentration, waterlogging, and metal toxicity, and climatic stresses, such as drought, high/low temperatures, and extreme radiation. In the case of soil properties, an imbalance in the soil salinity, due, for example, to insufficient drainage, may cause osmotic stress and ionic stress, which have as consequences growth inhibition due stomatal closure and suppression of photosynthesis [11]. Alternatively, a lack of nutrients in the soil may also limit crops' growth, whereas their excess may become toxic for the crop, leading to the same result as before. Figure 2.1 schematically represents the major stressors for a crop.

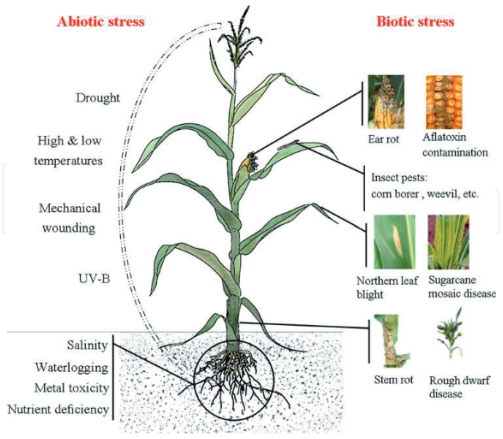


Figure 2.1: Overview of major causes of biotic and abiotic stress(from [1])

If we move instead to analyze the causes of climatic stress, drought is probably the most dangerous. Limiting water supply to crops for an extended period due to insufficient irrigation or

rainfalls may cause an abnormal metabolism responsible for reducing and potentially halting the plant's growth [10, 1], and even causing the crop's death. Other causes of climatic stress may be excessive heat, which may affect photosynthetic efficiency and crop phenology, and excessive cold, which causes poor leaf expansion, necrosis, and eventually death [10].

Considering how much crops are affected by the surrounding environment, keeping track of all parameters becomes essential to properly assess crops' growth conditions and eventually take action to avoid possible dangerous situations [12]. In order to do this, sensors are the best solution, as they allow the monitoring of such parameters directly in situ and continuously. Temperature can be measured by using traditional thermocouple or Resistance Thermometer Detectors (RTDs): relative humidity is usually measured using capacitive-type sensors, constituted by a pair of electrodes having in-between a specific material with a known dielectric constant to create a capacitor. Light can be sensed in different methods: traditional solar irradiation (measured as Wm^{-2}) can be measured using pyranometer, while photosynthetic active radiation (PAR, μmol photons $m^{-2}s^{-1}$) and photosynthetic photon flux density (PPFD, μmol photons s^{-1}) measure only the photosynthetic active portion of incoming light, which spans between 400 and 700 nm, using quantum sensors; CO_2 is instead typically monitored using nondispersive infrared (NDIR) sensors which measure the absorption of the characteristic wavelength of the air. Finally, soil moisture can be monitored again using capacitative sensors similar to relative humidity or a tensiometer. At the same time, salinity can be measured using soil's electrical conductibility (EC) sensors, with a direct correlation between soil conduciveness and salt presence.

2.2 History of Crop Modeling

The history of crop models and the theory behind them goes back up to the early second half of the last century, as first theories and proposed models can be tracked down between the '60s and the '70s. It is in such years, in fact, that Forrester publishes his work "Industrial Dynamics" [13] (1961), a book where the author discusses the methodology to represent different types of systems through simple equations updating state variables in the classic form of $x(t+1) = x(t) + R$ with x being a state variable and R a rate at which x changes. Alongside the classical representation via equations, Forrester also introduces the so-called *relational diagrams*, visual graphs that represent the flow and correlation between the various state variables of a system, as shown in figure 2.2. Despite

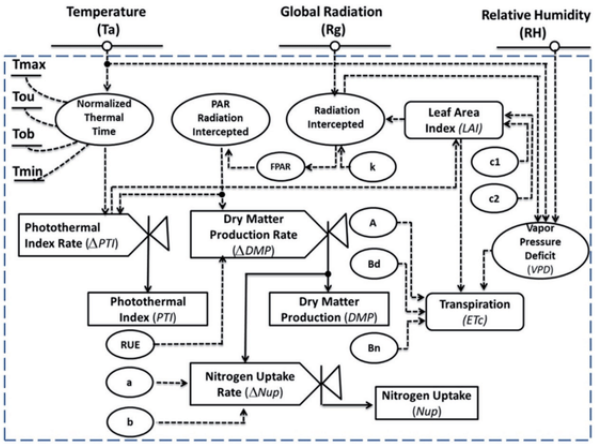


Figure 2.2: Example of relational diagram (from [2])

Forrester's book being mainly intended for describing industrial processes, its methodology spread over and reached the agronomical world, where it was adopted to describe systems constituted by crops and pests. It is possible to see, in fact, that following Forrester's work, a series of books and manuscripts began to appear in the agronomical literature explaining how crop systems could be modeled and simulated using those same methods described by Forrester. An example of such works is given by Zadoks's manuscript [14] (1971), De Vries [15], and Rabbinge's [16] books (1982 and 1989 respectively). Thanks to such works, Forrester's modeling approach started spreading, becoming more and more prominent in agronomy, and quickly turning into the central paradigm used to describe most agronomic models developed at the time and even up to today [17, 18].

2.3 The TOMGRO model

The original TOMGRO model was developed in 1991 by Jones *et al.*[19]. It aimed to describe tomato's crop growth in response to climatic conditions (temperature, light, CO_2) while assuming optimal conditions for the soil. The model divides tomato's growth into ten different phases, or age classes, and for each one assigns seven state variables to monitor the evolution [20]: number of leaves, number of stem nodes, number of fruits, dry weight of leaves, dry weight of stem nodes, dry weight of fruits, and area of leaves. Therefore, the total number of state variables describing the system would be 70, but since some are shared among age classes, the total number is reduced to 69 [21]. In the following years, a refined version of the TOMGRO, namely TOMGRO 3.0, was developed with an increased number of state variables, for a total of 574. In 1999, Jones *et al.*, in order to reduce the complexity of models developed so far, proposed a reduced version of the TOMGRO, which used only five total state variables [21]: number of nodes N (represented as a real value despite in reality being counted as an integer), leaf area index LAI, total aboveground dry weight W, total fruit dry weight W_f and total mature fruit dry weight W_m. The model runs on a daily basis, each variable evolving according to its differential equations. Ruling dynamics for the model are shown in equations (2.1) to (2.5), reported from the original work of Jones *et al.*.

$$\frac{dN}{dt} = N_m f_N(T_m) \tag{2.1}$$

$$\frac{dLAI}{dt} = \begin{cases} \rho\sigma\lambda(T_d)\dfrac{e^{\beta(N-N_b)}}{1+e^{\beta(N-N_b)}}\dfrac{dN}{dt} & \text{if } LAI \leq LAIMAX \\ 0 & \text{if } LAI > LAIMAX \end{cases} \tag{2.2}$$

$$\frac{dW}{dt} = \text{MIN}(\text{GR}_{\text{net}} - p_1\rho\frac{dN}{dt}, \frac{dW_f}{dt} + (V_{\max} - p1)\rho\frac{dN}{dt}) \tag{2.3}$$

$$\frac{dW_f}{dt} = \begin{cases} 0 & \text{if } N < N_{\text{FF}} \\ \text{GR}_{\text{net}}\alpha_f F_f(T_m)(1-e^{-\theta(N-N_{\text{FF}})}) & \text{if } T_{dm} \leq T_{\text{crit}} \\ \text{GR}_{\text{net}}\alpha_f F_f(T_m)(1-e^{-\theta(N-N_{\text{FF}})})(1-0.154(T_{dm}-T_{\text{crit}})) & \text{if } T_{dm} > T_{\text{crit}} \end{cases} \tag{2.4}$$

$$\frac{dW_m}{dt} = \begin{cases} 0 & \text{if } N \leq N_{\text{ff}} + K_{\text{ff}} \\ D_f(T_m)(W_f - W_m) & \text{if } N > N_{\text{ff}} + K_{\text{ff}} \end{cases} \tag{2.5}$$

In equations (2.1)-(2.5), the terms T_m and T_{dm} represent respectively the daily average temperature and the daily average *daytime* temperature, f_N is a piecewise function computing growth reduction under sub-optimal temperatures (fully examined in section 5.4), and GR_{net} is an algebraic term that can be substituted with the expression(2.6), representing the plant's nutrient gross assimilation for the day:

$$\text{GR}_{\text{net}} = E(P_g - R_m)(1 - f_R(N)) \tag{2.6}$$

Finally, we remand to table 5.1 in Chapter 5 to check the meaning for all the remaining parameters and support functions.

1999 TOMGRO model remained almost unchanged in the years. However, it is worth mentioning a significant change proposed in the last decade in Bacci *et al.* [22]'s work, where LAI's growth function ((2.2)) was changed from the original expolinear formula with a more typical sigmoid function which, according to writers, better represents leaf area evolution. Bacci's alternative formulation for LAI is presented in equation 2.7. In such formula, it is important to note that the LAI's value is computed directly instead of its increment, as this formulation computes LAI using only the number of nodes N..

$$\text{LAI}(N) = \frac{ab + \text{LAIMAX} \cdot N^d}{b + N^d} \tag{2.7}$$

Figure 2.3 visually represents the differences between the two formulations. Bacci's is more natural and closer to the crop's actual behavior, whereas TOMGRO's presents sharper changes and is less close to the natural behavior.

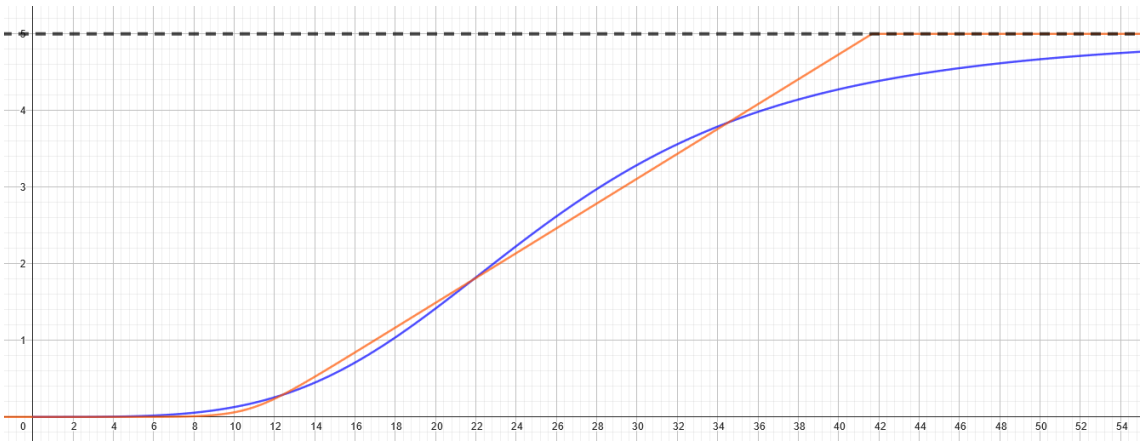


Figure 2.3: Evolution of the LAI using proposed formulas: orange is (2.2) and blue is 2.7

One of TOMGROs crucial components consists in the computation of crops' biomass dry weight, either total and for fruit, both mature and not. The first element needed to compute such values is a way to obtain the total gross nutrient assimilation, computed in equation (2.6). In the formula, the terms P_g and R_m represent respectively the daily gross photosynthesis and maintenance respiration, i.e. the total nutrients accumulated over day and the amount of these which are required by the crop to just survive. These two terms evolve faster than the others and are computed hourly. Therefore, in the TOMGRO model, these two elements are computed as the hourly integral over the day duration, according to equations (2.8) and (2.9), where dh denotes an integration on an hourly basis. Equation (2.9) is derived from [21], while equation (2.8) from the orignal model([19]) instead. The term CO_2 indicates the average hourly CO_2 concentration, T_{dm} the average hourly temperature, and PPFD the average hourly photosynthetic photon flux density, all inputs sensed from the environment.

$$Pg = \int \left(\frac{D\tau CO_2 \mathrm{PgR}(T_{mh})}{K} ln \left(\frac{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD}}{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD} \cdot e^{-K \cdot \mathrm{LAI}}} \right) \right) dh \tag{2.8}$$

$$R_m = \int \left(Q_{10}^{\frac{T-20}{10}} \mathrm{rm} \cdot (W - W_m) \right) dh \tag{2.9}$$

Since in this document we will mainly focus on the 1999 reduced TOMGRO model, from now on we will refer to it simply as "TOMGRO", except for when explicitly stated otherwise.

2.4 Crop Diseases

One of the main concerns in agriculture comes for sure from the several plant pathogens, such as bacteria and fungi, that can cause severe damage to consumers, crops, and the economy, an example being the 1845 catastrophic spread of *Peronospora* on potato production in Ireland in [23]. Plant protection, however, is not an easy task, as using products to avoid diseases often clashes with the requriements of sustainability and environmental protection. For example, recent European regulation (REG 1107/09/CE)[1], focusing on health and environment protection, forces farmers to abandon traditional chemical-based agricultural production and to adopt plant production strategies coming from integrated or organic production. Furthermore, REG 128/09/CE[2] establish a framework for sustainable use of pesticides. These regulations force the adoption of more knowledge-based approaches to predict and monitor plant infections.

Infections on plant hosts are favored by specific environmental conditions, which may change depending on the pathogen itself and within the time such pathogen spends in the field or greenhouse crops. Generally speaking, it is possible to establish three conditions that must necessarily occur for an infection to happen:

1. Presence of a susceptible host.

[1] `https://eur-lex.europa.eu/EN/legal-content/summary/pesticide-safety-on-the-eu-market.html`
[2] `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:02009L0128-20190726`

2. Presence of a virulent pathogen in the environment.

3. Availability of nutrients and favorable environmental conditions for both the host and the pathogen.

Since the first two conditions are unavoidable, one can try to work on the third, especially in greenhouses, where one can have more or less sophisticated control over environmental variables (temperature, relative humidity, light).

Since the presented work focuses mainly on the tomato crop, a selection of diseases typical for such plants has been considered for modeling. Among them, some are caused by bacteria, others by insect species that can damage the tomato crop (*Trialeurodes vaporariorum*, *Bemisia tabaci*, *Tuta absoluta*, etc.), other diseases may be caused by fungal pathogens, such as Powdery mildew, Grey mold, Late blight, Cladosporium leaf mold, Alternaria leaf spot and White mold to name a few. Among the presented ones, powdery mildew is probably one of the most common and dangerous diseases in tomatoes, caused by the biotrophic tomato pathogens *Oidium Neolycopersici* [24] and *Leveillula Taurica* [25, 26], where *biotrophic* means that the pathogen act as a parasite, living in symbiosis with the host, using the host's cells as fuel for its reproductive capabilities. Because of this, that pathogen's goal is to survive symbiosis with the host, not kill it.

Parasites causing Powdery Mildew are distributed everywhere in the world. However, they are most frequently found in temperate climate zones and rarely in humid tropical areas, where daily precipitation can wash the spores away from the host plant [24]. Because of such behaviors, these pathogens can find in greenhouses' climate the ideal environmental conditions to establish and proliferate, leading to crop-level infection if not controlled. The greenhouse environment is, in fact, tailored to maximize crops' (hosts') growth in order to increase the yield, and without extreme fluctuations in temperature and relative humidity values that may instead happen in open fields. Moreover, as plant biomass increases in the greenhouse, their total transpiration increases as well, thus maximizing relative humidity, a condition highly correlated with the probability of powdery mildew infections [27, 28, 25]. In addition, other microclimatic factors may further enhance the chances of a pathogen finding the perfect environment to spread. For example, large water availability may allow the plant to maximize its transpiration function, causing an increase in relative humidity near the leaves, thus promoting infection and growth of pathogens that appreciate high relative humidity. Once the infection begins, powdery mildews can be easily recognized due to their peculiarity of creating a white superficial mycelium, a set of round colonies (pustules) that gradually coalesce to white continuous coatings. These coatings are more often formed on leaves and less frequently on fruits and stalks [24]. However, when this happens, it signals that the infection has already started spreading.

Among causes of Powdery Mildew, *Oidium Neolycopersici* is a particular species belonging to the group of powdery mildew fungi. *Oidium Neolycopersici* is a highly polyphagous fungus that infects tomato plants and causes powdery white lesions on the tomato leaf surface while typically leaving the fruits uninfected. Severe infections may lead to the manifestation of these symptoms and cause leaf chlorosis, necrosis of the tissue, premature senescence, and a marked reduction in fruit size and quality [29]. As a consequence, the overall productivity of a plant can be reduced by 20-40%. The causes for such reduction are found in the development of the fungal mass on the leaf surface, which acts as an obstacle to the proper functioning of chlorophyll function, blocking sunlight radiations. In addition, lesions produced by the fungus reduce the functionality of the leaf surface for both chlorophyll function and transpiration. As such, *Oidium Neolycopersici* has become a problem, primarily since it has spread rapidly around the world, possibly due to the ever-increasing movement of plants through the international horticulture trade. However, the main reason behind such spread is due to the aerial dispersion of conidia (spores), their subsequent survival on numerous alternative hosts, and their ability to hibernate to infect when environmental conditions permit.

2.5 A model for *Oidium Lycopersici*

Due to all the reasons listed in the previous section, we decided to put our focus on disease modeling on the *Oidium Neolycopersici* pathogen and its interaction with a tomato plant host, represented by the TOMGRO model. In particular, we based our research on the work of Chelal *et al.* [30], which is, to our knowledge, the only model to simulate the interaction between tomato growth and *Oidium Neolycopersici*. The model examines the behavior of the tomato-powdery mildew pathosystem with a set of differential equations, specifying how healthy, diseased, and defoliated leaf tissue varies over time and using a time step of one day. The model, however, assumes that

environmental factors' variations in the greenhouse can be neglected and, therefore, constant rates for host and disease dynamics can be assumed, as well as that the total leaf tissue produced by the plant during its growth remains the same both in the case of healthy and diseased plant. Finally, similar to other previously described agronomic models, this one is also heavily parametrized. The model's key evolution of infected LAI can be summarized in equations (2.10)-(2.12).

$$\frac{dH}{dt} = r_H(H+Y)\frac{1-(H+Y+D)}{H_{max}} - \text{Rate}_Y \tag{2.10}$$

$$\frac{dY}{dt} = \text{Rate}_Y - \frac{dD}{dt} \tag{2.11}$$

$$\frac{dD}{dt} = r_D Y \tag{2.12}$$

where H stands as the absolute leaf area of the plant, Y stands for the diseased leaf area, and D stands for the defoliated area, i.e., the total area of leaves lost.

Since modeling the entire infection model would be a complex task, we decided to make some assumptions about the considered system with the purpose of reducing the model's complexity while at the same time keeping its meaning intact. Such assumptions are:

- Disease does not impact the overall leaf growth. We assumed that, due to the pathogen growing mainly on the lower leaves, which remain in the shade, taller ones responsible for much of the chlorophyll function, as directly exposed to sunlight, remain sane and thus allow the plant to grow [26]. Furthermore, the highest growth of the pathogen is observed on mature leaves that, even in the absence of the pathogen, would not have further growth.
- Disease-induced defoliation does not occur. We decided not to consider relevant natural defoliation; therefore, it is not modeled in the tomato model. Consequently, the D variable introduced earlier will always be set to zero and never appear in our model. This is because if even the leaf is severely diseased, detachment is unlikely to occur. Moreover, even if pathogen-induced defoliation could occur, it is always possible to measure healthy leaf area and diseased leaf area at the time of detachment and set those values as constants over time for that specific leaf. This approach was first used in [31].
- The diseased leaf part of a leaf is not recoverable, as it becomes covered with a coating that is difficult to "wash away". Moreover, even if washed away, the pathogen has already emerged, meaning it has colonized the leaf, producing lesions. Assuming the pathogen's death also does not help, as necrosis can be observed in infected areas, meaning the chlorophyll function of that tissue is still to be considered compromised.
- Secondary infections occur immediately after the end of the latent period. In an actual situation, conidia mature must be dispersed before they can germinate, meaning they have to leave the site they are via aerial dispersal to reach an uninfected area, thus requiring some time. In our model, instead, we assume that mature spores immediately find a new leaf area to infect. This assumption is also in accordance with the reference model [30].

Following the assumptions made, we can rewrite the model's equations by removing all formulas related to dead leaves. Moreover, according to the model, the first term in equation (2.10) refers to the increase of sane leaf tissue over time; hence, it can be described as $\frac{dH_s}{dt}$, while only the second term Rate_Y can now be rewritten using as the increment of the ill tissue over time $\frac{dY}{dt}$, as they are now the same thing, as shown in equations (2.13) and (2.14)

$$\frac{dH}{dt} = \frac{dH_s}{dt} - \frac{dY}{dt} \tag{2.13}$$

$$\frac{dY}{dt} = \text{Rate}_Y \tag{2.14}$$

where H_s represents the increment in sane leaf tissue. Such specification is crucial, as healthy tissue is already provided by the tomato model, albeit using LAI, meaning that it is possible to substitute the H_s terms with the output, properly adapted, of the TOMGRO. Therefore, the disease model's remaining element is the variable Y, representing the diseased leaf area. Such variable is driven by the Rate_Y terms, which is defined as

$$\text{Rate}_Y = \begin{cases} 0 & \text{if } 0 \le t < \text{IP} \\ r_{\text{LINmax}}\text{DE}(T,\text{RH}) & \text{if IP} \le t < \text{LP} + \text{IP} \\ r_{Y\max}\text{DE}(T,\text{RH})Y(1-\dfrac{Y}{H+Y}) & \text{otherwise.} \end{cases} \tag{2.15}$$

where IP and LP stands respectively for the *Incubation period* and *Latent Period*. The incubation period lasts from when a spore lands on a host to when symptoms become visible; meanwhile, the latent period overlaps with the incubation period but ends when the consequent infection has released other spores to infect again. Therefore, the expressed by LP + IP indicates the moment in time when the first spores have finished their life cycle while new ones have become visible, signaling the completion of the second cycle. As a consequence, we can split the pathogen's growth inside the cultivation into three main phases: first, a silent phase where the disease starts growing while hidden; a second phase, also known as *primary infection*, where the disease symptoms appear on the initial host and start spreading until it ends its life cycle; a final phase, or *secondary infections*, where the all subsequent infections start spreading. The presented equation perfectly mimics such behavior: a zero-growth phase at the beginning, a linear growth during primary infection to account for a single pathogen infecting a host, and a logistic growth to account for all subsequent infections following the disease spread in a population. Figure 2.4 shows the timeline of a disease spread from a host regarding the incubation period, latent period, primary infection, and secondary infections.

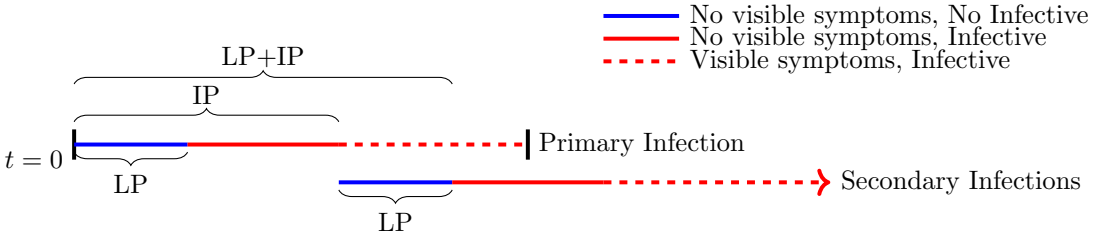


Figure 2.4: Disease spread timeline.

The last term in equation (2.15) is the DE function. Such term indicates the *Disease Efficiency*, a reduction coefficient between 0 and 1 indicating how much environmental conditions favor the pathogen's growth. In [30], the DE function is defined as

$$\mathrm{DE}(T, \mathrm{RH}) = \begin{cases} d(T - T_{\min})^n (T_{\max} - T)^m (1 - e^{-a\mathrm{RH}})^b & \text{if } T_{\min} \leq T \leq T_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

where T and RH represent the temperature and relative humidity, respectively, while $T_{\min}$ and $T_{\max}$ the minimum and maximum temperature between which the pathogen is allowed to grow.

2.6 Control in Agronomic context

Control in the greenhouse context plays much different compared to its role in industrial plants. In that context, control can be enforced onto each machine separately and with precise actions; therefore, the system's controllable components and targets of our control match perfectly. For example, it is always possible to turn on or off a specific machine in a production line or schedule the list of actions a set of equipment must follow to reach a particular goal, be it productivity or safety. In greenhouses' contexts, however, this does not apply since the controllable components of the system (actuators) are different from the ones we aim to control plants. Since crops are, in fact, living beings that grow and respond passively to the specific environment they are in, there are only very few actions we can force onto them to reach a given goal, meaning we must apply indirect control over them by enforcing properties on the surrounding environment which, by reflex, brings crops toward the wished goals. As a consequence, almost all greenhouse control techniques aim to build a suitable environment around crops, be it in the ground they grow in or in the climatic conditions inside the greenhouse, by building an environment adapted to sustain the grown and by fighting changes caused by the unpredictable nature of the environment itself. Therefore, control in the greenhouse is a continuous task that must balance between keeping conditions optimal against a changing environment while also trying to keep costs for actuation the lowest possible.

The choice of control system heavily depends on the technological level of the greenhouse used, meaning that low-level greenhouses will have much fewer options to control the environment. In contrast, high-tech solutions offer multiple methods to manipulate the environment, with the downside of having much higher costs in building and maintenance. In general, environmental control in the greenhouse can be achieved using the following methods [32]:

- Heating

- Ventilation
- Cooling
- Humidity Control
- CO_2 enrichment
- Light modulation

The first three methods all control the greenhouse's internal temperature, which is the most important driving factor for crop growth: active heating can be achieved via heaters that blow hot air or pump hot water in pipes; ventilation instead can be used to reduce internal temperature and can be achieved naturally by regulating window's opening or mechanically by using fans; cooling, finally, act as a further tool to reduce temperature, and can be achieved for example by using evaporative pads or fog systems. Both ventilation and cooling can be used together, with the former trading limited control capabilities for more affordable costs, and is therefore used more in low-tech greenhouses. Following, humidity control helps in dealing with pathogens and regulating transpiration, and can be regulated either indirectly by changing the temperature or raised by using humidification panels; CO_2 can be used to enhance crop growth and is typically injected in the environment via tubes or blowers; Finally, light modulation is achieved by using LEDs, and aims to provide missing illumination when sunlight is not sufficient for crop's development.

2.7 Hybrid Automata and Discrete-Time Hybrid Automata

When there is the need to formally model systems ruled by complex dynamics, such as biological ones, *Hybrid Automata* is often the first choice, thanks to their expressive power and ease of use. A hybrid automaton (HA) is defined by Sifakis *et al.* [33] as a tuple $H = (\text{Loc}, \text{Var}, \text{Lab}, \text{Edg}, \text{Act}, \text{Inv})$ consists of six components:

- Loc: A finite set of locations.
- Var: A finite set of real-valued variables. A valuation ν for the variables is a function that assigns a real value $\nu(x) \in \mathbb{R}\ \forall x \in \text{Var}$. We write V for the set of valuations. A state is a pair (ℓ, ν) consisting of a location $\ell \in \text{Loc}$ and a valuation $\nu \in V$. We write Σ for the set of states.
- Lab: A finite set of labels representing actions or events that can trigger transitions, which must include the *stutter* label τ.
- Edg: A finite set of edges called *transitions*. For each transition e, $e = (\ell, a, \mu, \ell')$, where $\ell, \ell' \in \text{Loc}$ represent the source and target locations, $a \in \text{Lab}$ is the transition label, and $\mu \subseteq V^2$ is the transition relation. The transition e is enabled in a state ℓ, ν if for some valuation $\nu' \in V, (\nu, \nu') \in \mu$. The state (ℓ', ν'), then, is a transition successor of the state (ℓ, ν). It is required that for each location ℓ it exists a transition in the form of (ℓ, τ, Id, ℓ) with $Id = \{(\nu, \nu) | \nu \in V\}$.
- Act: a labeling function that assigns to each location ℓ a set of *time-invariant* activities-i.e functions in the form $f \in \text{Act}(\ell), f : \mathbb{R}^{\geq 0} \to V$ for which it holds that $\forall \ell \in \text{Loc}$, activities $f \in Act(\ell)$ and nonnegative reals $t \in R^{\geq 0}$, also $f + t \in \text{Act}(\ell)$, where $(f + t)(t') = f(t + t') \forall t \in R^{\geq 0}$.
- Inv: A labeling function that assigns to each location ℓ an invariant $\text{Inv}(\ell) \subseteq V$, defined over some boolean predicate over $p : V \to [0, 1], \text{Inv}(\ell) = \{\nu \in V | p(\nu)) = 1\}$

Hybrid automata are exceptionally powerful for describing systems, as their high level of abstraction allows them to model a wide range of systems, from simple to highly complex ones, without requiring changes to the modeling approach. However, despite their expressive power, hybrid automata have certain limitations. Their generality, which enables the modeling of diverse systems, also makes traditional automata operations—such as composition, reachability analysis, and supervisor synthesis—significantly more complex to perform and implement in automated tools than less generic formalisms. As a result, support for hybrid automata is often limited to simulation, which means they are less likely to be the default choice for system modeling.

Depending on the considered system, a full hybrid description may not be necessary, especially for slow systems. In this case, we can refer to *discrete-time hybrid system* formalism [34], a subset

of hybrid systems that restricts dynamics to discrete-time intervals. In this regard, we can define a *Discrete-time hybrid automaton* (DTHA) by following Stauner's definition provided in [35], which gives all conditions necessary to identify a hybrid automaton as a discrete-time one. According to Stauner, a HA $A = (\text{Loc}, \text{Var}, \text{Lab}, \text{Edg}, \text{Act}, \text{Inv})$ is also a DTHA with time granularity T if an only if:

- $\text{Var} = \{x_1, x_2, \dots x_n, c\}$ for $n \in \mathbb{N}$. c is a special variable called the *clock* of A.
- For every location $\ell \in \text{Loc}$, the invariant $\text{Inv}(\ell) \subseteq V$ is in the form $(c = 0 \wedge p) \vee c \neq T$, where p is a predicate over $\text{Var}\backslash\{c\}$. This means that the invariant for each location, expressed by p, must hold at maximum for one timestep, forcing the system to take a transition after such an amount of time.
- For every location $\ell \in \text{Loc}$, the initial conditions are in the form of $p \wedge c = 0$-that is, for every location, its invariant predicates must hold, and the clock must be set to zero.
- For every activity $f(t) \in Act$, it must satisfy $f(t) = \nu \in \{v \in V | v(c) = t \wedge v(h) = h\}$. Therefore, all activity function must only update the clock variable, while keeping constant all the others. In other words, $f(t)$ can be considered a function which returns only valuations that assign to variables the values that are solutions to the differential equations $\dot{c} = 1$; $\dot{x} = 0 \forall x \in \text{Var}$. Therefore, from here on, we will consider writing the $f(t)$ function as defined above and $f(t) : \dot{c} = 1$ as equivalent.
- The transitions' set Edg can be split into two *disjointed* set E_u and E_l as defined below, where E_u encompass transitions *updating* system's variables, while E_l groups *logic* transitions.

Transition set E_u consists of all system transitions that aim to update the system's variables. If $e \in E_u$, then e's guard must be in the form $c = T \wedge p$ with p being a convex predicate over $\text{Var}\backslash\{c\}$, its update must be in the form of $q \wedge c = 0$, with q being a convex predicate over $\text{Var}'\backslash\{c\}$, with Var' representing the variables' set after the update, and source location and destination location must be the same. In other words, transitions belonging to the E_u set are all self-loops enabled every time the clock completes a time step cycle of T, that update variables as needed and then resets the clock to zero, replacing the continuous updates performed in a pure hybrid automaton. Similarly, the transition set E_l groups all system transitions representing a logical switch in the system's mode. A transition e belongs to E_l if and only if its guard is in the form $c = 0 \wedge p$ and the update function does not modify c. Here, again, we have p being a convex predicate over $\text{Var}\backslash\{c\}$.

More formally, following [35] we first define a *transition-step relation* $\multimap$ as $(\ell, \nu) \multimap (\ell', \nu')$ iff $(\ell, a, \mu, \ell') \in \text{Edg}$ and $(\nu.\nu') \in \mu$. Next, we define the δ *time step relation* $\rightsquigarrow^{\delta}_{\tau}$ defined as $(\ell, \nu) \rightsquigarrow^{\delta}_{\tau} (\ell, \nu')$ iff there exists a differentiable funtion $\tau \in [0, \delta] \to V, \delta \geq 0$ for which:

- $\tau(0) = \nu$ and $\tau(\delta) = \nu'$
- $\tau(t) \in \text{Inv}(\ell)$ for $0 \leq t < \delta$
- $\tau \in \text{Act}(\ell)$

Using these two relations as basis, we can now define the *update relation* $\multimap_u$ as $(\ell, \nu) \multimap_u (\ell', \nu')$ iff $(\ell, \nu) \multimap (\ell', \nu')$ by an edge in E_u, and the *control-step relation* $\rightsquigarrow^{j}_{(u_i)}$ as $(\ell, \nu) \rightsquigarrow^{j}_{(u_i)} (\ell, \nu')$ iff:

- (u_i) is a sequence of function where each function is defined as $u_i \in [0, T] \to \mathbb{R}^{n+1}, i \in [1 \dots j-1]; (u_i(t))(c) = t \wedge (u_i(t))(x) = \nu''(x) \forall x \in \text{Var}\backslash\{c\}$
- there exists $\nu_i, i \in [1 \dots j-1] n \geq 1$ and ν_n such that $(\ell, \nu_i)(\rightsquigarrow^{T}_{u_i}; \multimap_u)(\ell, \nu_{i+1})$, where ";" denotes the sequential composition of relations
- $\nu_0 = \nu, \nu_n = \nu'$

Finally, we can define the *discrete hybrid time step relation* $\boxtimes^{j}_{u_i}$ as $(\ell, \nu) \boxtimes^{j}_{u_i} (\ell', \nu')$ iff there exists a $\nu'' \in \mathbb{R}^{n+1}$ such that$(\ell, \nu)(\multimap)^*(\ell', \nu'') \rightsquigarrow^{j}_{(u_i)} (\ell', \nu')$; and, subsequently, the *discrete-time execution* as a finite sequence of discrete-time hybrid step: $(\ell_0, \nu_0) \boxtimes^{j_0}_{u_i,0} (\ell_1, \nu_1) \boxtimes^{j_1}_{u_i,1} \cdots \boxtimes^{j_m}_{u_i,m} (\ell_{m+1}, \nu_{m+1})$.

To consolidate such concepts, let us consider the following example: we want to model a DTHA which model a digital clock which prints the hour using the AM/PM format. As such, our model is defined as follows:

- Loc={AM,PM}

- Var = $\{h, c\}$
- Lab={toPM, toAM,update τ}
- Edg={(AM,update,μ_u,AM),(PM,update,μ_u,PM),(AM,toPM,μ_1,PM),(PM,toAM,μ_2,AM), (AM,τ,Id,AM),(PM,τ,Id,PM)}
- Act(L) = $\{f(t) = \nu \in V | \nu(c) = t \wedge \nu(h) = h\}$, L=[AM,PM]
- Inv(L) = $\{\nu \in V | v(c) < H \wedge h \leq 12\}$, L=[AM,PM]

where $\mu_1 : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = 0 \wedge v(h) > 12\}, \nu_u \in \{v \in V | v(h) = 1\}$ and $\mu_u : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = H \wedge v(h) < 12\}, \nu_u \in \{v \in V | v(c) = 0, v(h) = h + 1\}$, with H symbolizing a time period of one hour. Figure 2.5 graphically depicts the model, expressed both using Sifakis' definition for Act functions and the simplified version expressing Acts functions as a differential equation. Assuming the automaton's initial state to be (AM,($c = 0, h = 1$)), a possible

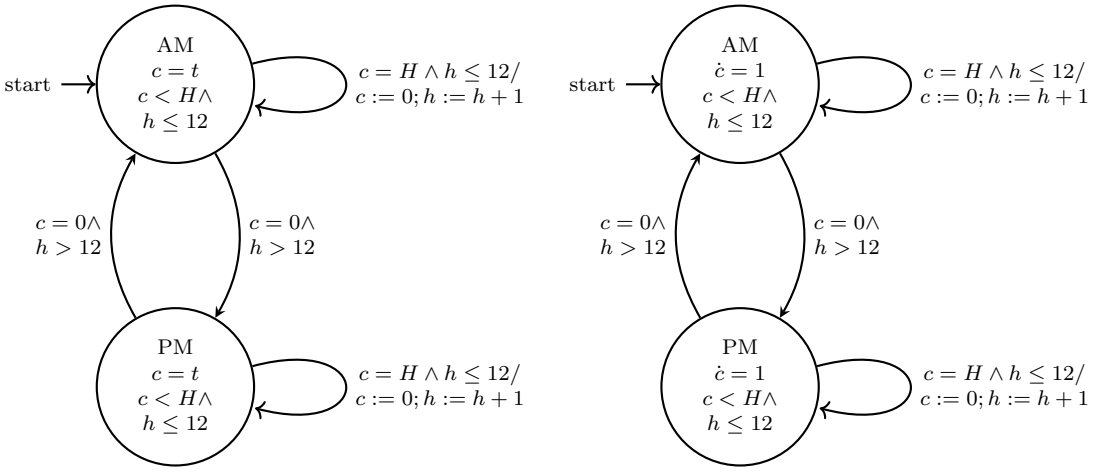


Figure 2.5: Example clock automaton

execution would be the following:

$(\text{AM}, (c = 0, h = 1)) \rightsquigarrow_{u_i} (\text{AM}, (c = H, h = 1)) \dashrightarrow (\text{AM}, (c = 0, h = 2)) \rightsquigarrow_{u_i} (\text{AM}, (c = H, h = 2))$
$\dashrightarrow (\text{AM}, (c = 0, h = 3)) \rightsquigarrow_{u_i} \cdots \dashrightarrow (AM, (c = 0, h = 12)) \rightsquigarrow_{u_i} (AM, (c = H, h = 12)) \dashrightarrow$
$(\text{AM}, (c = 0, h = 13)) \dashrightarrow (\text{PM}, (c = 0, h = 1)) \rightsquigarrow_{u_i} (\text{PM}, (c = H, h = 1)) \dashrightarrow \ldots$
$\dashrightarrow (PM, (c = 0, h = 12)) \rightsquigarrow_{u_i} (PM, (c = H, h = 12)) \dashrightarrow (\text{PM}, (c = 0, h = 13)) \dashrightarrow$
$(\text{AM}, (c = 0, h = 1)) \rightsquigarrow_{u_i} \ldots$

Basically, after H time has elapsed, clock variable h is updated by one following the self loop of location AM; then, once the clock reaches 13 the system switches to the PM location, resetting the clock back to one. Here, the cycle repeats with clock variable being incremented by one only after H time has passed until it yet again reaches 13. Once this happens, the system switches back to the AM location and the execution loops from the beginning.

2.8 The Control Supervisory Theory

The supervisory control theory (SCT), known also as the Ramadge–Wonham framework [36], is a methodology developed by Ramadge *et al.*in 1989 as a way to restrict a system's behavior described as a finite state machine in order to avoid reach unwanted state, under the assumption that some events in the system are *uncontrollable*, i.e., they can not be disabled by a controller, and can happen at any time. The task of SCT is then to automatically synthesize supervisors that restrict the model's behavior so that the given specifications are fulfilled as much as possible.

In a similar way to how we defined automata in section 2.7, we now define *Finite state automata* (FSA) as a tuple $G = \langle X, \Sigma, f, \Gamma, x_0, X_m \rangle$, where X is the finite set of states with $x_0 \in X, X_m \subset X$ being the initial state and the marked states, Σ the finite set of events, $f : X \times E \rightarrow X$ the transition function, where it holds $f(x, s\sigma) = f(f(x, s), \sigma)$ for $s \in \Sigma^*, \sigma \in \Sigma$, and $\Gamma : X \rightarrow 2^{\Sigma}$ is the active event function, which specify for each state x events e where $f(x, e)$ is defined. Next, we

define the language generated by G as $\mathbb{L} = \{s \in \Sigma^* : f(x_0, s) \text{ is defined}\}$, as well as the language marked by G as $\mathbb{L}_m(G) = \{\sigma \in \mathbb{L} : f(x, s \in X_m)\}$. For simplicity, we will refer to $\mathbb{L}(G)$ simply as $\mathbb{L}$ and to $\mathbb{L}_m(G)$ simply as $\mathbb{L}_m$. Moreover, we also define the set of events Σ as $\Sigma = \Sigma_c \cup \Sigma_u$, where Σ_c is the set of controllable events, while Σ_u is the set of uncontrollable events.

Given an automaton G, we can define the supervisor function S as a function $S : \mathbb{L} \to 2^{\Sigma}$, which defines the set of enabled events on G as $\forall s \in \mathbb{L} : S(s) \cap \Gamma(f(x_0, s))$, i.e. the automaton G can not perform any action which is not also allowed by the supervisor. We say a supervisor to be *admissible* if it holds that $\forall s \in \mathbb{L} : \Sigma_u \cap \Gamma(f(x_0, s)) \subseteq S(s)$, that is, the supervisor never turns off an uncontrollable event. Given S, we can define the language generated by the controlled automata $\mathbb{L}(S|G)$ and $\mathbb{L}_m(S|G)$ as

$$\begin{aligned}\mathbb{L}(S|G) :&\epsilon \in \mathbb{L}(S|G);\\ &[s \in \mathbb{L}(S|G) \wedge (s\sigma \in \mathbb{L}) \wedge (\sigma \in S(s))] \iff [s\sigma \in \mathbb{L}(S|G)]\\ \mathbb{L}_m(S|G) =&\mathbb{L}(S|G) \cap \mathbb{L}(G)\end{aligned}$$

We say that the supervisor S is *nonblocking* if $S|G$ is nonblocking itself, which means it must holds that $\mathbb{L}(S|G) = \overline{\mathbb{L}_m(S|G)}$, where $\overline{\mathbb{L}_m(S|G)}$ is the prefix-closure of $\mathbb{L}_m(S|G)$. Furthermore, we call $K \subseteq \mathbb{L}_m$ the specification (or requirements) the supervisor S implements.

Given the notion of *nonblocking supervisor*, we can express the *nonblocking controllability theorem*:

Theorem. Let G be an automaton describing a discrete event system with $\Sigma_u \subseteq \Sigma$ being the set of uncontrollable events. Let $K \subseteq \mathbb{L}_m(G)$ with $K \neq \emptyset$ be the specification we want to force on G. Then, it exists a *nonblocking supervisor* S for G such that: $\mathbb{L}_m(S|G) = K$ iff it holds that:

1. Controllability: $\overline{K}\Sigma_u \cap \mathbb{L}(G) \subseteq \overline{K}$
2. K is $\mathbb{L}_m(G)$-closed, i.e. $K = \overline{K} \cap \mathbb{L}_m(G)$

Condition 1 of the presented theorem states the requirement to consider a system controllable under a given specification K. If the condition does not hold, we must change our specification K to make it either stricter or loosen it. Starting from the specification K, we can define either

- the *supremal controllable sublanguage* $K^{\uparrow C}$, which represents the maximal subset of K which guarantees controllability over G
- the *infimal prefix-closed controllable superlanguage* $K^{\downarrow C}$, which represents the smallest superset of K which is prefix-closed and controllable.

Most importantly, these two languages are guaranteed to exist.

For the purpose of our work, we will focus only on $K^{\uparrow C}$. To compute such language, various algorithms can be applied to different levels of complexity. Here, we will provide a general idea of how the synthesis algorithm works:

```
1   Let G = < X^g, Σ^g, f^g, Γ^g, x_0^g, X_m^g > the sytem automaton
2   Let K be the specification for G
3   Let Σ_u be the set of uncontrollable events
4   Let H = < X^h, Σ^h, f^h, Γ^h, x_0^h, X_m^h > the automaton such that L(H) = K
5   Let P = < X^p, Σ^p, f^p, Γ^p, x_0^p, X_m^p > compurted as P = G×H

7   Remove unreachable states:
8   X^p= {x^p ∈ X^p | x^p --> x_m^p ∈ X_m^p}

10  Iteratively remove bad states.
11  Let i=0; X_i = X^p; B_i=∅
12  repeat:
13      foreach x = (x^g, x^h) ∈ X_i, x^g ∈ X^g, x^h ∈ X^h :
14          Let e^p= {e ∈ Σ^p | Γ^p(x) = False};
15          Let e^g= {e ∈ Σ^g | Γ^g(x^g) = True}
16          if e^p ∩ e^g ⊆ Σ_u:
17              B_i = B_i ∪ {x}
18      X_{i+1} = X_i \ B_i
19      i= i+1
20  until X_i = X^{i-1} or x_0^p ∉ X_i

22  if x_0^p ∉ X_i:
23      return false

25  return < X_i, Σ^p, f^p, Γ^p, x_0^p, X_m^p ∩ X_i >
```

The general idea behind the synthesis procedure consists first of computing the product automaton between the plant and the specification (line 5). Next, we eliminate unreachable states, i.e., those state combinations which naturally we would not be able to reach any sequence of possible events accepted by the automaton, represented in the code by the `-->` operator (line 8). Finally, we compute a list of "bad" states, i.e., those states where the original plant uncontrollable transactions were possible but are now disabled in the product automaton. To do this, we check for each state in the `P` automaton which events are disabled and compute the intersection with events that are instead allowed in the corresponding state of the system (lines 14-15). If the intersection contains at least one uncontrollable event, it means that our controller is trying to block it, and therefore, we add the examined state to the list of bad states. After all states have been examined and the list of bad ones computed, we remove them to the complete states set, since those are the ones the controller must avoid. After removing such states and related transitions, the algorithm loops back and performs a second check on the new states' set. The loop ends in two possible cases: if we reach a fixed point, and in that case, the new automaton is returned as the controller; otherwise, if the initial state is removed from the set of allowed states, then the procedure ends, as no controller can be synthesized.

Let us, therefore, consider the following example: figure 2.6-a depicts a sample plant automaton P, subjected to controllable events a, b and uncontrollable event x which generates the marked language $\mathbb{L}_m(P)$={"ab","ba", "bbb"}. Automaton P, however, suffers a flaw for which state P_5 can either end up in marked state P_3 or in state P_6, which is a deadlock. As such, we describe our specification K as in figure 2.6-b: basically, we described the wished behavior of our plant, which simply consists in all sequence of events leading to the marked state without never ending in the deadlock state. By examining the provided specification, we can immediately see how condition

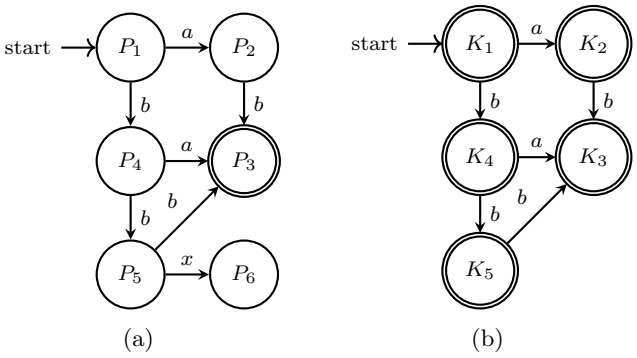


Figure 2.6: Plant (a) and Specification (b) automaton

1 of the theorem for the existence of the supervisor does not hold, since the specification misses a transition on uncontrollable event x present in the plant P. As such, we have to apply the previously described algorithm in order to compute the maximally permissive supervisor. Steps of the algorithm are shown in figure 2.7: first, we compute the product $P \times K$ (2.7-a); next step would be to remove unreachable states, but in such examples none exits, and therefore we skip this passage and move on to the next one. Following the synthesis algorithm, the next operation is to remove all states where an uncontrollable transition (Fig 2.7-b,c) was removed after the product, in our case this being state (P_5, K_5), that lost the transition on event x. Therefore, we remove such state, and all transitions entering the state. Subsequently, we examine again the product automaton, to check whether the previous removal operation caused the loss of transition on uncontrollable events; in this case no state are illegal and therefore the resulting automaton at the end of this iteration is the same as the previous one, meaning we reached a fixed point and that obtained automaton is our supervisor S (Fig. 2.7-d). Finally, we can compose together the supervisor and the original plant to realize wished specification. However, as we can note, implemented supervisor does not implement exactly our specification, as it did not satisfy the theorem of existence. Instead, the supervisor implements the maximum sublanguage possible which both satisfy our constraints while also avoiding situations where uncontrollable events may lead it to unwanted states. In particular, the new marked language of the product between plant and supervisor is only $\mathbb{L}_m(P \times S)$ ={"ab","ba"}.

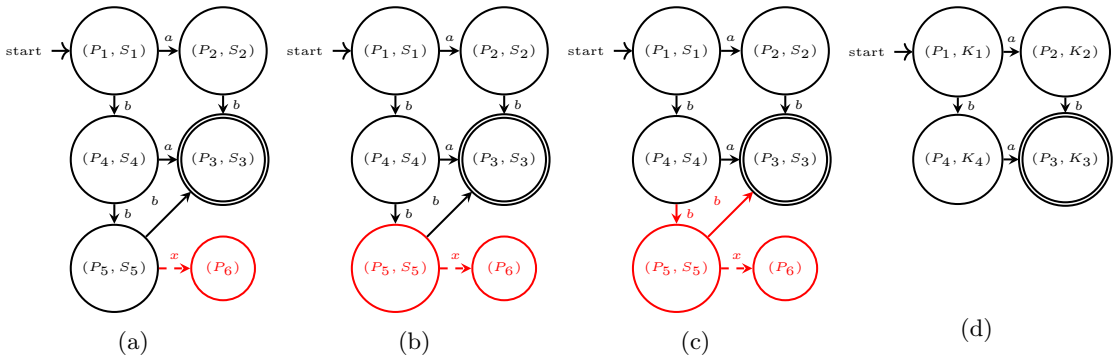


Figure 2.7: Supervisor synthesis applied to automaton $P \times K$. Red elements are the ones removed during the procedure.

2.9 The Eclipse Supervisory and Control Engineering Toolkit

The Eclipse Supervisory Control Engineering Toolkit (ESCET)[3] is an open-source project that aims to provide a comprehensive model-based engineering framework for the development of supervisory controllers developed by the Technical University of Eindhoven in collaboration with various industrial realities, such as Rijkswaterstaat, ASML, and Vanderlande. ESCET's purpose is to facilitate the engineering process for discrete-event systems by providing practical tools to automatically generate controllers that are correct by construction, as ESCET uses at its core supervisory controller synthesis techniques, therefore granting safety, controllability, and non-blocking behavior for synthesized controllers.

ESCET offers users a wide range of functionalities, including modeling, the above-mentioned supervisor synthesis, simulation-based validation and visualization, formal verification, real-time testing, and code generation [37]. Central to its operation is the CIF (Compositional Interchange Format) language [38], which offers an automata-based approach to modeling complex systems, as it allows to define plants, requirements, and supervisors through automata formalism, be it traditional finite-state automata, extended finite-state automata or even hybrid automata, supporting synchronization and invariant-based modeling. Furthermore, ESCET applies various techniques to scale as best as possible with larger and more complex systems such as industrial plants by optimizing memory usage and computation time. Examples of such techniques include the usage of Binary Decision Diagrams (BDD) data structures or the multilevel synthesis that allows the distribution of the control problem over smaller subsystems. Finally, ESCET allows for code generation in multiple programming languages, including Java, C99, Simulink, and some PLC standards.

[3] `https://eclipse.dev/escet/`

Chapter 3

State of the Art

In this Chapter, we will describe other works presented throughout the years related to the presented work, either because they faced similar problems or applied solutions similar to those in related fields.

3.1 Climatic heterogeneity within a greenhouse context

The problem of monitoring climatic conditions is well-known in agriculture, as even small changes in such conditions may significantly impact plants' growth and health status, even leading to irreversible damages if left uncontrolled [39]. Greenhouses can significantly help farmers in this regard, providing them more control over plants' microclimate [40]. Such innovation led throughout years to studying new research opportunities, such as finding the best values of climatic variables to either maximize yield [39] or reduce energy consumption [41], water waste [42], and pesticide usage [43]. A common factor among all these approaches is the requirement to model, in some way, the climatic behavior of the greenhouse itself in order to estimate the evolutions of monitored climatic variables and eventually act to modify it and avoid possible unwanted situations or, on the other hand, only to force a specific wished scenario.

Generally speaking, models can be divided into two main classes. The first ones are *mechanistic* models, which are described by equations derived from physical laws, such as energy or mass conservation. In the first part of their work, Singh *et al.* [44] sums up the history of mechanistic models, shortly describing the most important works since 1970. The second class of models groups instead *stochastic* models, which aim to find relationships between given sets of input and output data in order to minimize prediction error by using techniques from Statistics, such as regressions [45], or Artificial Intelligence, such as neural networks [46].

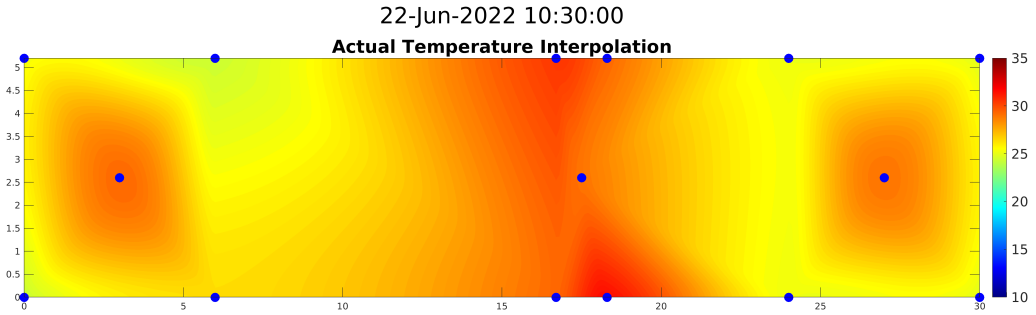


Figure 3.1: Temperature map obtained by interpolating readings from blue points.

Past works often assumed the environment to be uniform [47] to reduce model complexity and computational costs, while in truth climate heterogeneity can be easily observed inside the greenhouse [48], as figure 3.1 clearly shows. These differences in the climate may lead to irregular growth of crops and, if not correctly addressed, may hinder plant disease prediction. It is possible, in fact, that a particular disease may appear at a specific point with favorable microclimate and spread from there, even if the average climate behavior seems unfavorable. Nowadays, various models can predict the insurgence of specific diseases based on climatic conditions, e.g., in [49]; therefore, a fine-grain vision of the greenhouse microclimate can greatly help detect possible critical points and act preemptively.

To account for this heterogeneity, different monitoring strategies have been proposed, ranging from sensors scattering as in [50] to Computational Fluid Dynamics (CFD), which links temperature to its physical causes, e.g., sunlight absorption, reflection, and refraction ([51, 52]). In [53], authors took a step further and introduced virtual sensors inside the greenhouse via regression techniques using outside data as inputs and then generating a microclimatic model in real-time by using CFD. Such solutions produced accurate results to describe the greenhouse's microclimate heterogeneity. However, scattering sensors require many of them (and related infrastructure) to work correctly, leading to a quick growth in costs; CFD models, on the other hand, require the identification of hundreds of physical parameters. Furthermore, farmers are usually not interested in climate variable values for every point of their greenhouse but rather in particular *points of interest*, e.g., in correspondence of plant rows. In addition, not every greenhouse location can host a sensor because of spatial and working constraints.

3.2 Soft Sensors applications

Soft sensing is a well-known technique used in statistics. It is traditionally applied to process control, being used as a method to control industrial plants from the early '90s, as shown in [54], [55] and [56]. Generally, it focuses on developing approaches and algorithms to estimate or predict physical quantities in industrial processes based on the available measurements and knowledge, as defined in Jiang *et al.*'s review on the topic [57]. In agriculture, however, soft sensing has been employed only recently. In [58], the authors developed soft sensors to monitor crop transpiration, for which affordable sensors were not available, while in [53] soft sensors were used to run in real-time the CFD model developed in that work. Authors of [59] first created a complete CFD model of the greenhouse whose outputs were then used to train the soft sensors, oppositely to our approach, which is instead purely data-driven as we train soft sensors directly with data from persistent and temporary sensors.

3.3 Crop modeling

Throughout the years, many different crop models have been proposed, each developed to address a specific objective, be it a particular crop species or emphasizing a specific resource used by crops, such as water consumption, nitrogen, etc.

Outside of the TOMGRO model used in this document, other examples of crop models developed with such methodology include CERES [60] (Crop Environment Resource Synthesis), which models crop yield, aboveground biomass, harvest index, evapotranspiration, and many more variables for crops such as wheat (CERES-Wheat), rice (CERES-Rice) and maize (CERES-Mize); SUCROS [61] (Simple and Universal Crop growth simulator), another model which simulates wheat development given environmental factors; CROPGRO [62], a crop simulation that simulates crop's daily growth and development as a function of daily weather, soil properties, crop management, and cultivar/species parameters; TOMSIM [63], an alternative simulation model for tomatoes' dry matter production and LAI.

Despite all the differences among all crop models developed throughout the years, we can find a common aspect shared by all is the heavy parametrization of these model's equations, which allows us to tune models' behavior according to the different conditions surrounding examined crops without needing to re-implement anything or developing a new model. This particular trait is also one of the reasons that made and keeps such models popular up to the present. Because of such perks, more recent studies preferred to emphasize parameter variability rather than changing and/or developing new models. Using the TOMGRO model as an example, after its presentation paper in 1991 [19] the model has undergone only two significant changes, one in 1997 [64] and one in 1999 [21], and all subsequent manuscript shifted their focus on studying TOMGRO's parameters, in terms of how to properly calibrate them and how much the model is sensible to their variation. Some of these studies include Cooman *et al.* in 2006 [65] and 2007 [66] where the uncertainty of TOMGRO's outputs were studied when subject to variation in parameters and inputs values respectively; Bacci *et al.* [22] (2012) that calibrated TOMGRO's parameters to use the model under Italian climatic conditions; Vazquez-Cruz *et al.* [67] (2014) who performed sensitivity analysis to determine TOMGRO's most sensible parameters that impact on model's output; Shamshiri *et al.* [68] (2016) that evaluate TOMGRO's reduced model under boundary condition and in different geographical location concerning where the model was originally calibrated; Gong *et al.* [69] (2021), who used evolutionary algorithms to perform calibration onto TOMGRO's parameters.

Finally, during the last two decades, other works have also begun to appear, addressing the problem of modeling and control within a greenhouse environment. Such works do not focus on the crop models themselves but rather aim to integrate and use those models to solve problems of controlling the environment to achieve a specific objective, for example, to maximize yield or prevent disease. Examples of these works include Van Henten *et al.* [70], who performed sensitivity analysis regarding optimal control problem formulated to manage greenhouse climate, and the two works of Vanthoor *et al.* [71, 72], that performed model-based design methodology by using a greenhouse model and a crop model (tomato in their case) to optimize the design of their greenhouse.

3.4 Disease modeling

Despite *Oidium Neolycopersici* being an evident problem for the growing of greenhouse tomatoes and is of increasing significance for field crops [29], little research has been dedicated to modeling the synchronous interaction between such disease and its host plant under the most common and advisable environmental conditions for tomato growth. The model proposed in [30], which we will use in Chapter 6, describes the development of powdery mildew symptoms caused by *Oidium Neolycopersici* coupled with tomato plant development and is to our knowledge the only modeling interactions between the pathogen and the crop. Similar works also include [31], where controlled experiments in a glasshouse were conducted in order to determine the temporal progress of the powdery mildew disease by performing artificial inoculation of conidia of *Oidium neolycopersici* on tomato plants. The study focused on determining the difference between plants that are not treated with anti-fungal sprays, plants that are treated after 10 days, and plants that are treated after 20 days, from the perspective of host-pathogen interaction in various aspects such as disease severity, disease distribution based on leaf position, actual leaf area, defoliated leaf area, plant height, and healthy leaf area duration. The study also served as the precursor of [30] to define the model. Alternatively, in [73], authors monitored the expansion of powdery mildew epidemic in different greenhouses and found some typical expansion patterns: constant, linear, and a Gaussian bell surface. A deterministic long-range trend and a short-range random residual are observed in all three patterns. The primary sources of infection found are air currents from greenhouse doors and openings, although workers and machinery also spread spores since higher severity was found in the most frequently traversed pathways. In [74] and [75], instead, machine learning techniques were used to develop an automated spray prediction model for the powdery mildew disease to optimize the number of fungicide applications and the timing of application. Finally, in [26], which, however, focused on *Leveillula Taurica*, authors built and evaluated a forecasting model that aimed to minimize the number of fungicide spray applications based on climate data. The model was able to determine whether climatic conditions were conducive to high disease severity and throw a warning to alert the farmer that he should apply a fungicide.

3.5 Cellular Automata

Cellular Automata (CA) are formal models used to describe discrete dynamic systems that emphasize the interactions between multiple individuals living in a multidimensional space. Initially introduced by John von Neumann and Stanislaw Ulam in the '40s, CA have found their primary usage in modeling physical, biological and epidemiological systems, such as reaction-diffusion systems, self-reproduction models, epidemic spreading, forest fire diffusion [76].

Since cellular automata work in a multidimensional space, different definitions can be provided depending on the number of dimensions the modeled system has. In case of this work, the confrontation we face is against *bidimensional cellular automata*, which can be used to model the spatial spreading of a pathogen. Formally speaking, a bidimensional cellular automaton consists in a 2D, $r \times c$ grid of identical objects (cells) and is defined as a 4-tuple (C, Q, V, f) [77] where:

- $C = \{(i,j), 1 \leq i \leq r, 1 \leq j \leq c\}$ represents the cellular space,
- Q is a finite set of possible states for each cell.
- $V = \{(\alpha_h, \beta_k), 1 \leq k \leq n\} \subset Z \times Z$ defines the neighborhood of each cell. For each cell (i,j) we can define its neighborhood as $V_{i,j} = \{(i+\alpha_1, j+\beta_1), \ldots, (i+\alpha_n, j+\beta_n)\}$. Specifically, we also define here commonly-used Von Neumann neighborhood as $V = \{(0,0), (-1,0), (1,0), (0,-1), (0,1)\}$, considering the cell itself and the four adjacent one, and Moore neighborhood

as $V = \{(0,0),(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(-1,1),(1,-1),(1,1)\}$, which extends Von Neumann to include all surrounding cells.

- $f : Q^{|V|} \to Q$ is the local transition function that dictates how, at each timestep t, each cell updates its state $s^t_{i,j}$ based on its neighbors:

$$s^t_{ij} = f(s^{t-1}_{i+a_1,j+b_1}, s^{t-1}_{i+a_2,j+b_2}, \ldots, s^{t-1}_{i+a_n,j+b_n}).$$

The evolution of a cellular automaton occurs in discrete time steps: at each step t, the state of each cell is updated synchronously based on the transition function f and the values of its neighbors at time $t-1$. Figure 3.2 shows a graphic representation of the evolution for a simple CA, where each cell changes color if one of its neighbors had changed state at the previous time step.

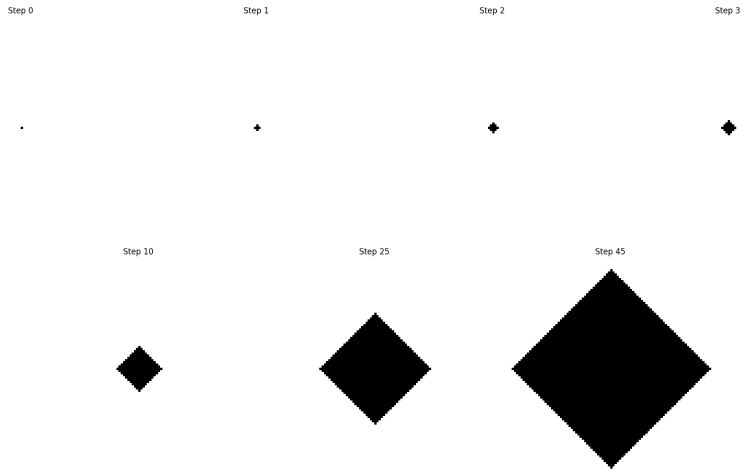


Figure 3.2: Evolution of the example Cellular Automata

Although CA posses a huge expressive power in terms of modeling spatial interactions, they also suffer from two big limitations which limits the kind models that can be described. First, traditional CA evolve their cells' state over time following an update function depending only on the state of neighbor cell; therefore, we can not to implement within traditional CA any kind of temporal dynamic and as such any kind of hybrid behavior. Secondly, each cell in bidimensional CA implements the same update function, meaning that we can not to differentiate cells such that part of the grid evolve according a certain rule and another one following a different one. As such, using cellular automata to model a system which evolves in time and space, such as a pathogen growing and spreading across plants, is not possible. Meanwhile, we will show throughout this thesis that using DTHA with opportune event synchronization, inspired by CA's update functions, we become able to model both temporal and spatial evolution for pathogen infesting crops inside a greenhouse.

3.6 Greenhouse Environmental Control

The task of controlling environmental variables in a greenhouse context is not a trivial one. As stated earlier, greenhouse control systems are very different from industrial ones, mainly due to dealing with many different variables and their unpredictable behavior. Consequently, the variable space grows exponentially with each new parameter we aim to control, making finding the optimal strategy harder. In addition, a big difference between industrial plants and greenhouse systems consists of the fact that control in greenhouses is not always fully automated; contrary, on many occasions, it is achieved by manual labor, i.e., control actions are provided by human operators who make decisions based on the greenhouse's current status. Consequently, the greenhouse control problem can be split into two possible subsets: Manual and Automatic Control.

3.6.1 DSS-Aided Manual Environmental Control

Manual control leaves the decision-making process of choosing the best environmental parameters for the greenhouse entirely in the hands of human experts. Such people, strong in their knowledge about crop growth and the greenhouse system and their experience, can use data coming from the

field coupled with direct observations and sampling to assess the system status and subsequently plan how control is performed by setting thresholds and set points for the installed equipment. In this context, *Decision Support Systems* (DSSs) may offer significative help in the decision-making process, collecting and elaborating data in real-time from the field to assist farmers and agronomists in optimizing agricultural practices [78]. Additionally, DSS systems may already offer some knowledge coded from experts or predictive models to further enhance the decision process. Various DSSs exist today, each one offering different tools to support farmers in their work. Among them, we cite ones described in [79]: DSSAT [80], a widely and well-know suite of models that simulate crop growth, soil conditions, and climate interactions to support decision-making; WOFOST (WOrld FOod STudies) [81], a simulation model developed for the quantitative analysis of annual crops' growth and production by modeling the physiological processes of plants; InfoCrop [82], a generic simulation growth model providing data on a daily basis on crop's growth parameters, yield, nitrogen requirements, greenhouse gas emissions, as well as water and nitrogen levels in the soil; AquaCrop [83], a crop model developed by FAO to assess environmental and managerial effects on crop production, and specifically for herbaceous crops in conditions of limited water; APSIM (Agricultural Production Systems sIMulator (APSIM)) [84], a highly advanced simulator of agricultural systems composed by a large suite of modules enabling simulation for systems made of plants, animals, soil, climate, and management interactions. Finally, we also mention vite.net® [78], a web-based DSS for vineyard management, integrating weather data, crop status, and disease models to provide real-time recommendations.

Although functional, however, nowadays DSSs suffer from what is called to what is called the *problem of implementation* [78, 85], i.e. the lack of sustained use for these systems in a way that they can actually produce results and be seen as beneficial, The causes of this under-performance of DSS systems can be tracked down to two significant factors: technical limitations and user resistance. Formers arise from the way the DSS system is built and presented, often tied to the overall quality of offered services and software, with six major key points creating the main divisive barrier between the tool and users:

- Considering the system in its entirety, instead of addressing single problems or sub-systems
- Quality of models built into the software
- User-friendliness
- Time required to by data processing and easiness of submitting inputs
- Software's output must come at the right timing, providing enough room for decision-making and acting
- Regular maintenance

In addition to such factors, DSS must also face users' resistance, especially if such systems are tuned to make decisions automatically and, therefore, replace human contributions. As a consequence, there is a rejection of such full-automatic systems in favor of ones that simply assist the decision-making process.

3.6.2 Full Automatic Environmental Control

Opposite to the previous section, automatic control aims to build an environment where decisions on how and when to operate the greenhouse's equipment are taken exclusively by a controller aiming to keep the greenhouse's conditions within a given range or to satisfy others requirements, such as energy consumption or waste production, provided by farmers according to their own desire.

Control techniques can be split into two main categories: open-loop and closed-loop. Open-loop systems, sometimes also called sequence control, operate simply by following a given schedule of actions according to the scenario currently playing in the greenhouse, meaning that multiple schedules can be issued for different conditions, and the controller will actuate the one corresponding to the greenhouse conditions. This type of control is probably the easiest to implement, relying only on a limited amount of input and mainly on the knowledge of the people who program the actions, thus being relatively cost-effective, although lacking in adaptability. On the contrary, closed-loop (or feedback) control continuously monitors greenhouse conditions through sensors and adjusts actuators dynamically to maintain optimal conditions to ensure greater precision and responsiveness to fluctuations in conditions. Obviously, this also comes with higher computation and monitoring equipment costs.

Various algorithms can be applied to achieve automatic control. In their work, Chen *et al.* [86] deeply analyzed the most used control strategies applied in modern greenhouses and related algorithms. Here, we will only report the ones we believe to be the most important ones. The first type of control strategy proposed in the review is the Proportional-Integral-Derivative (PID), one of the earliest and most used techniques in agriculture [87]. Based on a feedback loop schema, this strategy continuously adjusts the control system based on the deviation between a measured parameter and a desired setpoint: the proportional component reacts to the current error, the integral component accounts for accumulated past errors, and the derivative component predicts future errors. However, PID control is primarily effective for single-variable control but struggles with complex, multi-variable greenhouse environments. Following, we have fuzzy logic control [88], which is a nonlinear, model-less approach that processes inputs using a rule-based system to determine appropriate actuator responses, and Model Predictive Control (MPC) [89], which is an advanced closed-loop strategy that uses mathematical models to predict future greenhouse conditions and determine optimal actions sequence. Finally, we can find neural network-based control strategies, such as in [90], that utilize machine learning algorithms to model complex greenhouse environments and optimize control strategies by analyzing historical and real-time sensor data and predicting environmental trends to make control adjustments. Obviously, the presented strategies are not mutually exclusive, but they can be combined in a way that covers the weak points of another, thus strengthening the final strategy.

3.6.3 Speaking Plant Approach

All control strategies presented so far aim to optimize crop's environmental conditions using as input data those same variables they aim to control. Consequently, the system they aim to supervise is not the actual complete greenhouse system but rather only the environmental one, and crops can be considered a separate subsystem loosely tied to the controlled one only by sharing some variables. Absurdly, one could think of using such a control technique regardless of the crop's type or presence inside the greenhouse.

In recent years, a novel method to manage the greenhouse's environment has started rising to close the gap between control techniques and crops' growth status. This approach, known as *Speaking Plant Approach*, or SPA [91, 92], is an innovative concept in intelligent greenhouse management that aims to optimize crop cultivation by extending traditional controllers' inputs with crop's physiological status, either estimated by a model or directly sampled using special sensors, just like crops were "speaking" to it. Examples of data sensed from crops include chlorophyll fluorescence, stomatal conductance, and photosynthetic activity.

3.7 Hybrid Automata Simulation and Control Syntehsis

Due to their popularity in the modeling context within computer science and the many algorithms developed throughout the years to perform operations on them, many software tools have been developed to simplify the modeling process on automata. In matter of Hybrid System modeling, the Control Systems Society provides an extensive list[1] of available tools which can be used to model, simulate and verify hybrid systems. For modeling and identification, we cite CHARON [93], a tool for the hierarchical description of interacting and concurrent hybrid systems, where each component is described as a hybrid automaton with its continuous flow updates and which communicates using shared variables and evolves according to a hierarchical state machine; HyEQ [94], a simulation toolbox for Matlab/Simulink capable of computing approximations for system's trajectories given in terms of differential and difference equations with constraints; IOA [95], a toolkit supporting algorithm design, development, testing, and formal verification of models described using I/O automata [96]; Modest [97], a toolset which supports modeling and analysis of hybrid, real-time, distributed and stochastic systems using a modular framework centered around the stochastic hybrid automata formalism. Regarding verification, we mention Ariadne [98], a C++/Phython library that allows describing automata with nonlinear behavior while computing conservative over-approximations for automata's reachability problem; HyCreate, [99], a software tool for defining hybrid automata and computing over-approximations of reachable sets for systems with nonlinear, non-deterministic dynamics; HyTech [100], a C++ symbolic model checker for hybrid systems which computes the conditions under which system's parameters satisfy the safety and timing requirements for a given embedded system description provided as a collection of communicating automata; Finally, we mention UPPAAL [101], a toolbox for verification of real-time

[1] `https://ieeecss.org/tc/hybrid-systems/tools`

systems represented by timed automata extended with integer variables, structured data types, and channel synchronization developed by Uppsala University and Aalborg University. Originally developed for timed automata only, it has evolved to support evaluating performance properties of (stochastic) hybrid systems through UPPAAL SMC (Statistical Model Checking), which allows defining continuous behaviors expressed via stochastic and nonlinear dynamics. UPPAAL SMC focuses mainly on model checking by combining traditional methods and simulations to boost and evaluate the system's performance properties. Finally, in the matter of hybrid systems control, we mention AMYTISS [102], a C++/OpenCL tool for designing correct-by-construction controllers for large-scale discrete-time stochastic systems by building finite abstractions of the given original systems as finite Markov decision processes (MDPs) and synthesizing a controller from it using scalable parallel algorithms; CoSyMA [103], a tool for automatic control synthesis of incrementally stable switched systems, which generates a controller for the system enforcing a given safety or time-bounded specification starting from a description of a system as set of differential equations and sampling parameters used to define an approximation of the state-space generates; SCOTS, a software tool for automatic controller synthesis of nonlinear control systems which construct a symbolic model of the input system via Lipschitz-based estimation and discretization to perform controller synthesis for invariance and reachability goals using fixed-point computations. Finally, we also must mention the recent works regarding least-violating controller synthesis for reactive systems, such as in the work of Aminof *et al.* [104]. In such contexts, the focus shifts to developing control systems capable of enforcing a specification while also managing exceptional events for as far as possible, aiming to develop a winning strategy against all possible *expected* events and that provide the best-effort answer to unexpected ones. In this regard, we cite TuLiP [105], a software toolbox for the synthesis of embedded control software with respect to an expressive subset of linear temporal logic (LTL) specifications, combining finite-state abstraction of control systems, digital design synthesis from LTL specifications and receding horizon planning to build a controller that is guaranteed to be correct for any admissible environment profile.

It is interesting to point out how most of the listed tools for the controller synthesis for hybrid models rely on performing an abstraction step to reduce the input model into a more manageable form. Abstraction is a powerful instrument, as it allows simplifying a given system to a more manageable form where usage and development of algorithms are less demanding and, moreover, provide more guarantees on the results. For example, in the work Nilsson *et al.* [106], authors introduce *augmented finite transition systems* to abstract continuous dynamics via the encoding of liveness properties in order to apply high-level specifications described using (LTL). The abstraction follows an incremental synthesis approach, starting with a coarse abstraction and refining it via fixed-point structures according to LTL formulas. Similarly, the work of Girard *et al.* [107] presents an abstraction-based method for controller design, focused on safety and reachability specifications and based on approximately bisimilar models. The controller is synthesized for the abstract system and then refined to the original system through a concretization process, thus ensuring correctness by design. Finally, we also mention the work of Liu *et al.* [108], who present a method for synthesizing robust controllers from temporal logic specifications using finite abstractions with built-in robustness margins.

The last topic we want to discuss here regards the tool used for controller synthesis for regular FSA. While, obviously, such tools can not be used on DTHA, they can be applied to an abstraction of such systems to compute a controller capable of enforcing some specifications on the system. Regarding control-focused tools, the most popular ones can be found in another list, again offered by the IEE Control Systems Society, where software tools that can be used to model discrete-time systems using FSA[2] are reported. On a similar note, a portion of the work by Reniers *et al.* [109] discusses some of such tools' usage, investigating offered support within an actual modeling context for a non-trivial system. Among all tools the IEE Control Systems Society, we cite DESUMA [110], a software and educational tool used to model and control discrete event systems as FSA written in Java combining the UMDES C library with the GIDDES tool for FSA visualization, MDESpos [111], an open-source Python tool for analysis and control of discrete event systems modeled as finite-state automata, including also non-deterministic systems, Supremica [112], a tool for automatic verification and synthesis of FSA, and the Eclipse Escet [37] suite, a Java framework implementing CIF, a modeling language capable to specify discrete event, timed, and hybrid systems as a collection of synchronized automata. All presented tools implement the main algorithms and theories of the Ramadge framework presented in Section 2.8 to perform supervisory control of DES. In particular, the last two presented tools also allow for modeling systems not only via FSA but also Extended FSA and, in the case of Escet, Hybrid Automata. Moreover, Escet

[2] `https://ieeecss.org/tc/discrete-event-systems/resources`

also allows the specification of the model and performs supervisory controller synthesis, as well as validation, visualization, verification, testing, and code generation.

Chapter 4

Soft Sensors for microclimate control

As noted in section 3.1, the greenhouse's climatic conditions are far from uniform, meaning using a single instance of a model to assess crops' conditions is not the correct choice. In order to solve this problem without needing to develop overly complicated models or deploy an excessive number of sensors, we proposed a new approach that combines permanent and temporary sensors, external sources of information, and statistical regression to create a microclimatic map of the greenhouse and to extend the concept of *soft sensors* (also called "virtual sensors") with different categories depending on the modeling technique and their usage. As depicted in Fig. 4.1, the proposed infrastructure mainly consists of the following entities:

1. Persistent sensors, installed permanently in low numbers to reduce costs and encumbrance, provide accurate data regarding the inside of the greenhouse (red circle in the figure) and its outside (green box in the figure).
2. additional information sources, e.g., for weather forecasting.
3. *Sparse Soft Sensors*, each created by correlating data from persistent sensors to data sampled by a temporary sensor in the same location (black crosses in the figure).
4. A *2D-Sensor* is a soft sensor describing the 2-dimensional behavior of a climatic variable obtained by interpolating data from *Sparse Soft Sensors*.

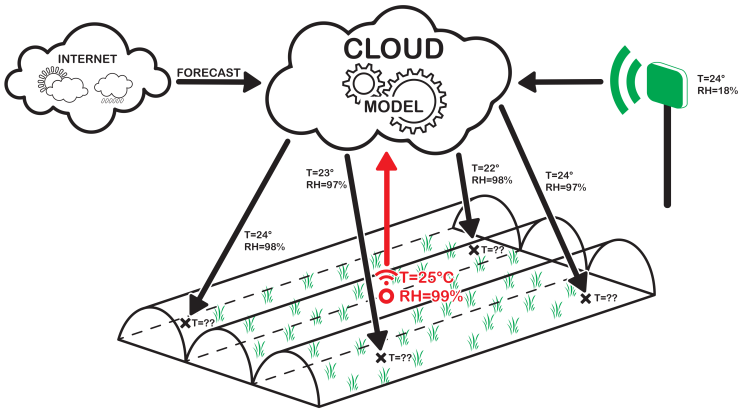


Figure 4.1: Proposed methodology for monitoring multiple points of interest based on a single sensor

Initially, only persistent and temporary sensors are placed, and sensing data forms the points of interest. Then, from those data, *Sparse Soft Sensors* can be modeled to substitute temporary sensors. Finally, a *2D-Sensor* is created, covering locations not directly considered in the sampling process. Additionally, removed temporary sensors can be placed again at different positions to restart the sampling process and increase the covered area and accuracy. Additionally, by changing

the input, this approach can also be used as a weather forecast system and a fault detection system for persistent sensors. We will label models that use forecast data as *Forecast Sparse Sensors*, while models for anomalies detection *Anomaly-Detection Soft Sensors*.

In particular, we applied our methodology specifically to compute air temperature at different locations since it is the main driving variable for all previous models.

4.1 Soft Sensors development

In order to give a proper order to previously defined soft sensors, a hierarchy was developed and is presented in Fig. 4.2. At the top are two main types of soft sensors, i.e., *Regressed (R) Soft Sensors*, obtained by using regression techniques, and *Interpolated (I) Soft Sensors*, obtained instead via interpolation. From there, *R-Soft Sensors* can then be divided as *Sparse Soft Sensors*, which air temperature value at specific locations, *Forecast Sparse Sensors*, that model future temperature using general weather forecast, and *Anomaly Detection Soft Sensors*, used to assess the status of a persistent sensor in function of other sensors' data. On the other hand, *2D-Sensor*s derives from *I-Soft Sensors* and computes temperature at any location through 2D interpolation for a single moment in time. In total, nine *Sparse Soft Sensors*, and the same number of *Forecast Sparse*

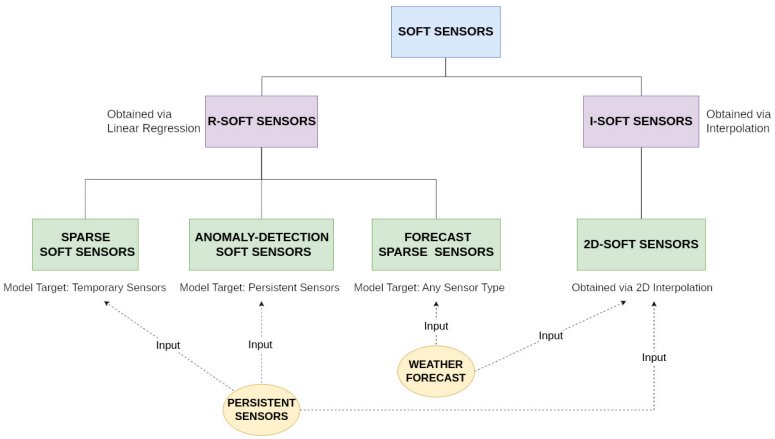


Figure 4.2: Hierarchy of Soft Sensors and their relationships with available data sources.

Sensors were created, one for each point of identified position (and later increased to fifteen each), more than ten *Anomaly Detection Soft Sensors* to cover all Evjas' and Davis' station sensors, and an unspecified amount of *2D-Sensor*, as with each new set of measures a new sensor had to be computed.

4.1.1 Regressed Soft Sensors

Regressed soft sensors allow for estimating the value of a climatic variable at a given location using input readings from sensors placed elsewhere, which are obtained as results of linear regression.

As in classical linear regression, the starting point to develop the models is the column vector $Y = \begin{bmatrix} y(t_1) & y(t_2) & \dots & y(t_n) \end{bmatrix}^T$ that stores the measurements taken from time t_1 to t_n by the sensor we want to model (defined as *target model*), and matrix

$$X = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_m(t_1) \\ \vdots & \vdots & & \vdots \\ x_1(t_n) & x_2(t_n) & \dots & x_m(t_n) \end{bmatrix} = \begin{bmatrix} x(t_1) \\ \vdots \\ x(t_n) \end{bmatrix}$$

which is an n by m matrix collecting measurements taken at the same n time instants from m other sensors (defined as *input sensors*). Generally, the X matrix can store any type of sensed data without restrictions. Therefore, the first step of the presented methodology is to use either the *lasso* and *stepwise* techniques to automatically remove variables that do not contribute to or may even reduce, regression accuracy. This step turns the X matrix into a smaller one, denoted as $\bar{X}$, where only functional components remain.

Although useful, removing useless elements from matrix X alone is insufficient to obtain a good model for the targeted sensor, as data from input sensors can only sometimes suffice to explain all phenomena that affect target models. Particularly, while the persistent sensors' readings allow

modeling of some recurring events (such as the day-night cycle of air temperature), they may not capture other phenomena, being "local" phenomena related to specific locations. However, if local phenomena are time-dependent (e.g., shades), they can still be included in the model by adding to the input set some variables explicitly representing the time at which each measurement has been collected. This leads to a new matrix $\tilde{X}$ defined as $\tilde{X} = \begin{bmatrix} M & | & D & | & H & | & \bar{X} \end{bmatrix}$ where M, D and H are binary maps defined as:

$$M = \begin{bmatrix} m_1(t_1) & \dots & m_{12}(t_1) \\ \dots & & \\ m_1(t_n) & \dots & m_{12}(t_n) \end{bmatrix}$$
$$D = \begin{bmatrix} d_1(t_1) & \dots & d_{31}(t_1) \\ \dots & & \\ d_1(t_n) & \dots & d_{31}(t_n) \end{bmatrix}$$
$$H = \begin{bmatrix} h_1(t_1) & \dots & h_{24}(t_1) \\ \dots & & \\ h_1(t_n) & \dots & h_{24}(t_n) \end{bmatrix}$$

were M, D, and H stand for Month, Day, and Hour, respectively. The i-th row of M, D, and H encodes the timestamp of the samples at the i-th row of the matrix $\bar{X}$, such that only the entry that has column index matching the month, day and hour of the timestamp is set to one, while other the row elements are set to zero. For instance, if the i-th row of $\bar{X}$ has the timestamp 2021-10-15 13:00, then the i-th rows of M, D ad H will have a 1 in column index 10, 15 and 13, respectively:

$$M_i = \begin{matrix} m_1 & \dots & m_9 & m_{10} & m_{11} & m_{12} \end{matrix}$$
$$\begin{bmatrix} 0 & \dots & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$D_i = \begin{matrix} d_1 & \dots & d_{14} & d_{15} & d_{16} & \dots & d_{31} \end{matrix}$$
$$\begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$
$$H_i = \begin{matrix} h_1 & \dots & h_{12} & h_{13} & h_{14} & \dots & h_{24} \end{matrix}$$
$$\begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

Having defined what the Y vector and the X matrix mean for *R-Soft Sensors* and which form they should have, it is now possible to give a more mathematical definition for such soft sensors. Assuming S_k to be a source of data (defined as *target*) placed in a point of interest that monitors the value of the variable V, then S_k can be described as the function

$$V(t) = S_k(t) \tag{4.1}$$

that for each instant of time t produces a value $V(t)$. The corresponding *R-Soft Sensor* $\hat{S}_k$, which generates the estimate $\hat{V}(t)$ of S_k, can then be described as

$$\hat{V}(t) = \hat{S_k}(\tilde{x}t) \tag{4.2}$$
$$= \tilde{x}(t)\hat{A} \tag{4.3}$$
$$= \tilde{x}(t)(\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T Y \tag{4.4}$$

Where $\tilde{x}(t)$ is a row related to the timestamp t of the matrix $\tilde{X}$ calculated before, and Y is the column vector described at the beginning of the section.

The following sections present the three types of *R-Soft Sensors*.

4.1.2 *Sparse Soft Sensors* and *Forecast Sparse Sensors*

We define s *Sparse Soft Sensor* as special *R-Soft Sensor* where the matrix $\tilde{X}$ is obtained from historical data gathered from persistent sensors, while the training vector Y consists of historical data coming from a temporary sensor placed at a specific a point of interest inside a greenhouse for a limited amount of time. Fig. 4.3 shows the complete set of operations to create the model following previously-explained steps, whereas Fig. 4.4 shows runtime operations to use such model to estimate the variable at time t in the corresponding point of interest.

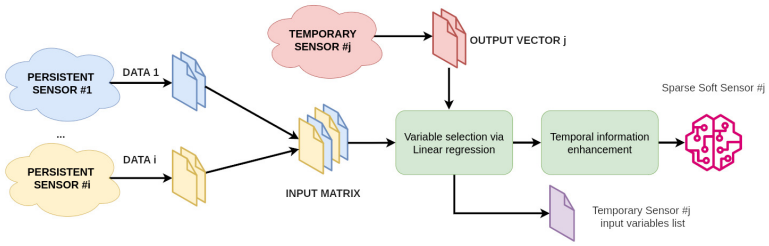


Figure 4.3: Creation flow of *Sparse Soft Sensors* starting from acquired data.

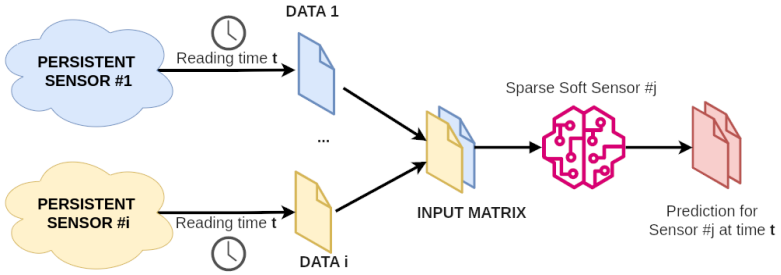


Figure 4.4: Runtime usage of *Sparse Soft Sensors* at time t.

Sparse Soft Sensors allow the modeling of greenhouse microclimate in real-time according to data obtained from persistent sensors. However, on different occasions, knowing the greenhouse's internal status may not suffice to choose which agricultural actions must be undertaken. On different occasions, in fact, information about the near future is preferable to assess crops' risk and which actions to take to avoid damages. Typically, information about climatic conditions for the near future is provided by weather forecasts, which often refer to a wide geographical area. However, we can slightly adjust the described regression methodology such that the X matrix is filled with the weather forecast data, and the Y vector contains data read from persistent or temporary sensors depending on the point of interest. With this change, we can obtain a model capable of localizing weather forecasts to specific positions within the greenhouse. We defined this new type of soft sensor as *Forecast Sparse Sensor*. Obviously, since weather forecasts typically have a lower time granularity than sensor readings, they cannot be directly related to the regression operation. To solve such a problem, sensor readings must before be averaged over the same time window forecasts refer to so that both sensed and forecast values have the same time granularity, and Eq. (4.4) can be used.

4.1.3 Anomaly Detection Soft Sensors

Since our methodology heavily depends on data gathered from sensors, anomalies generated by faulty persistent sensors introduce noise in soft sensors that use such data as input, thus resulting in wrong estimations.

To validate the measurements obtained from each persistent sensor, the proposed modeling technique can be exploited again. For each persistent sensor, we can use its historical data as our target Y vector and put it in relationship with readings from the other persistent sensors, which constitute the X matrix. We can then proceed to create a regression model, thus obtaining a soft sensor capable of estimating readings of the modeled sensor. We defined this type of soft sensor as an *Anomaly Detection Soft Sensor*. As explained in Fig. 4.5, these soft sensors can be used at runtime to estimate the climatic variable associated with each persistent sensor regarding readings from the others. Then, the estimation is compared to the actual sensor reading, and a difference is interpreted as an anomaly. It is worth noting that this mechanism can implement reciprocal supervision of the persistent sensors, and models can be continuously re-trained with new incoming data.

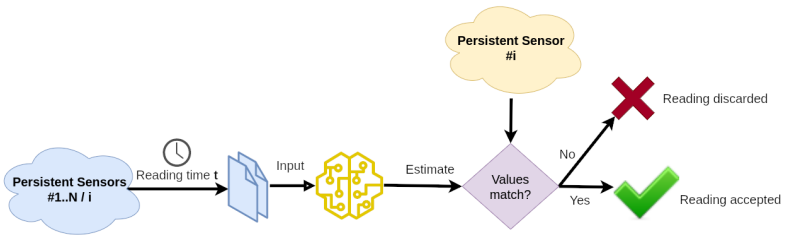


Figure 4.5: Runtime assessment of a persistent sensor's status.

4.2 *2D-Sensor*

Sparse Soft Sensors, *Forecast Sparse Sensors* and *Anomaly Detection Soft Sensors* allow monitoring a fixed number of locations without needing physical permanent sensors. However, in some occasions, it would be more desirable to monitor those same variables in arbitrary locations of the greenhouse area without introducing other temporary sensors and training new *Sparse Soft Sensors*. The 2D-interpolated estimation of a physical variable, given by Eq. (4.5), can be used to cover for such a task, as it estimates the variable values $\hat{V}_t^{p_x,p_y}$ for a given location (p_x, p_y) at time t using the interpolation function S_{2D}. In this case, the function itself is the soft sensor, as it allows the estimation of a climatic parameter for a specified position. We named this type of soft sensor as *2D-Sensor*.

$$\hat{V}_t^{p_x,p_y} = S_{2D}^t\left(p_x, p_y\right) \tag{4.5}$$

The *2D-Sensor* takes two parameters as input, i.e., the spatial coordinates of the point to monitor (p_x, p_y). It is worth highlighting that the sensor is bound to a time parameter: the interpolation process uses, in fact, data from sensors (of any type) to create a 2D surface and compute the value for the desired position. However, since climatic variables evolve over time, the interpolation function must be recalculated every time new data are available and can only be used to estimate data for that specific time frame.

Finally, the interpolation mechanism is flexible and can be combined with the previously described *R-Soft Sensors*. A *2D-Sensor* can create a microclimatic map referring to either the current time if it is fed by persistent sensors and *Sparse Soft Sensors* or a future time if it is fed by *Forecast Sparse Sensors*.

4.3 Experimental Setup for Applications

The proposed methodology has been applied using data from a real-world scenario. Within such an experiment, we

- modeled a microclimatic map via *2D-Sensor*fed by different *Sparse Soft Sensorss*;
- modeled *Forecast Sparse Sensors* for air temperature by using weather forecast obtained from an online web service;
- designed an *Anomaly Detection Soft Sensor* for each persistent sensor for fault detection purposes.

The research was conducted in greenhouse tunnels of two farms located near Verona (45° 20′ 25.9″N 11° 05′ 12.3″E and 45° 20′ 47.13″N 11° 1′ 46.4″E respectively) in north-eastern Italy and spanned from late August 2021 to early July 2022. Each tunnel is 275.6 m^2 (50 m x 5.3 m) northeast-southwest oriented and covered with transparent plastic film.

The primary data sources used as persistent sensors consisted of two microclimatic stations by *Evja s.r.l.* installed in each farm as well as a Davis Vantage Pro2 weather station installed relatively close to the greenhouses. The microclimatic stations were equipped with sensors to capture data for air temperature, relative humidity, solar radiation, soil humidity, soil temperature, and soil electrical conductivity. In contrast, the weather station monitored barometric pressure, temperature, relative humidity, wind speed, rainfall, and solar radiation, as well as computed dew point, wet bulb temperature, and heat index. Sampling time was set to fifteen minutes for all persistent sensors, and data were stored in the cloud through a SIM-based Internet connection and

Figure 4.6: The microclimatic station (Evja) hosting persistent sensors and the weather station.

accessed using MQTT protocol. Figure 4.6 shows the two data sources and their position in the greenhouse.

Within the two greenhouses, nine points of interest were chosen for each farm and positioned as in figure 4.7: in positions 1 to 8 HOBO S-TMB-M0xx air temperature sensors have been placed and connected to two UX120-006M loggers providing four channels each; in position 9 a HOBO UX100-011A Data Logger was used to monitor both air temperature and relative humidity. Additionally, six more temporary sensors were later introduced to cover larger gaps between installed sensors.

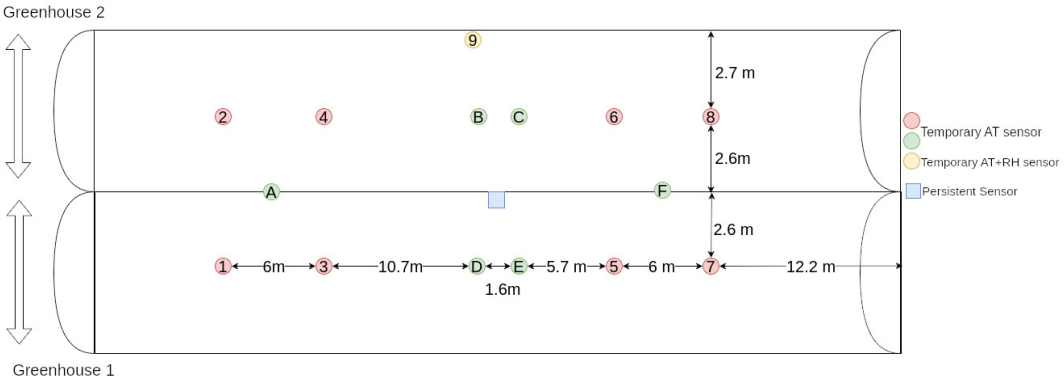


Figure 4.7: Position of persistent and temporary sensors across the two greenhouses. Sensors denoted by letters were added in a second moment for validation purposes.

4.4 *Sparse Soft Sensors* design and validation

Data from temporary sensors were used to create nine vectors Y_k to apply the methodology described in previous sections. The input matrix X collected all data produced from the Evja microclimatic station and the weather station for a total of 24 different input variables and more than $14,000$ readings over the observation period. Furthermore, in each row, we collected the measurements of the three previous time instants (hence up to 45 minutes in the past) from the weather station and Evja station

$$X = \begin{bmatrix} x_1(t_3) & \dots & x_m(t_3) & x_1(t_2) & \dots & x_m(t_2) & x_1(t_1) & \dots & x_m(t_1) \\ \vdots & & \vdots & & & & & & \vdots \\ x_1(t_n) & \dots & x_m(t_n-1) & x_1(t_n-1) & \dots & x_m(t_n-1) & x_1(t_n-2) & \dots & x_m(t_n-2) \end{bmatrix}$$

bringing the total input matrix's size up to 72 columns.

A MATLAB script was used to perform the regression automatically. First, we divided the X matrix and the Y vectors and split them into a training set and a test set, the former taking 80% (first 40% and last 40%) of the total measurements, while the latter the remaining 20%. After the division, we used MATLAB's `stepwiselm` function to compute a stepwise regression, which applies a forward selection and a backward selection in sequence to add all needed variables to the model and to later eliminate those that were not statistically significant, to minimize a given cost function. In our case, the cost function set was the adjusted R-squared. Before computing the regression, the input matrix X and output vector Y were normalized to feature zero mean and unitary variance.

Following component selection, the matrix was enhanced with temporal information as described in Section 4.1.1; the resulting matrix $\tilde{X}$ matrix is used in a final regression process to obtain the coefficients of the models representing the *Sparse Soft Sensors*.

To evaluate the resulting models' performance, the following three evaluation metrics have been used:

- adjusted-R-squared

$$R^2_{\text{Adj}} = 1 - \frac{n-1}{n-k-1}\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{1}^{n}(y_i - \bar{y})^2},$$

- Adjusted Root Mean Square (RMSE_{Adj}), calculated as the ratio between the Root Mean Square Error (RMSE) and the Root Mean Square Error(RMS) of y

$$\text{RMSE}_{\text{Adj}} = \frac{\text{RMSE}(Y, \hat{Y})}{\text{RMS}(Y)} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} y_i^2}};$$

- Relative Root Square Error (RRSE), calculated as the ratio between the Root Mean Square Error and the standard deviation σ

$$\text{RRSE} = \frac{\text{RMSE}(Y, \hat{Y})}{\sigma(Y)} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}}.$$

Where n is the number of samples, k the row number of coefficient vector A, y_i and $\hat{y}_i$ are elements belonging respectively to vectors Y and $\hat{Y}$, and $\bar{y}$ is vector Y's mean value.

Tables 4.1 and 4.2 list the metrics for air temperature of all the nine sensors placed in each farm, with the addition of relative humidity for the two points of interest further away from the Evja station (number 9 in Fig. 4.7). As it is possible to see in tables, each soft sensor reaches a precision of 85% or more both for training and testing, with most of them reaching an accuracy of 98-99%, meaning that *Sparse Soft Sensors* can estimate very precisely temperatures at the given points in space. The time series for the temperature *Sparse Soft Sensor* of the First Farm is reported in Fig. 4.8 and compared to the actual value of the temporary sensor used for training and to the values provided by the microclimatic station. Estimated values follow the actual ones as desired, whereas the microclimatic station does not, serving as another proof of the heterogeneity of the greenhouse's microclimate.

Sensor	R^2	Adj RMSE	RRSE
Temp #1	0.98	0.04	0.14
	0.96	0.05	0.19
Temp #2	0.99	0.03	0.11
	0.99	0.03	0.12
Temp #3	0.99	0.03	0.09
	0.99	0.03	0.10
Temp #4	0.99	0.03	0.10
	0.99	0.02	0.10
Temp #5	0.99	0.03	0.11
	0.98	0.03	0.12
Temp #6	0.99	0.04	0.14
	0.97	0.04	0.16
Temp #7	0.98	0.04	0.13
	0.97	0.04	0.16
Temp #8	0.98	0.04	0.13
	0.98	0.03	0.15
Temp #9	0.99	0.03	0.10
	0.99	0.03	0.12
RH #9	0.98	0.04	0.15
	0.94	0.05	0.24

Table 4.1: Accuracy results for each sensor in the first farm.

Sensor	R^2	Adj RMSE	RRSE
Temp #1	0.98	0.02	0.08
	0.99	0.02	0.07
Temp #2	0.99	0.02	0.08
	0.99	0.02	0.08
Temp #3	0.99	0.02	0.08
	0.99	0.02	0.07
Temp #4	0.99	0.02	0.08
	0.99	0.02	0.08
Temp #5	0.99	0.02	0.08
	0.99	0.02	0.09
Temp #6	0.99	0.02	0.09
	0.99	0.02	0.10
Temp #7	0.99	0.01	0.06
	0.99	0.02	0.07
Temp #8	0.99	0.02	0.08
	0.99	0.02	0.10
Temp #9	0.98	0.07	0.15
	0.87	0.11	0.31
RH #9	0.97	0.04	0.16
	0.94	0.05	0.21

Table 4.2: Accuracy results for each sensor in the second farm.

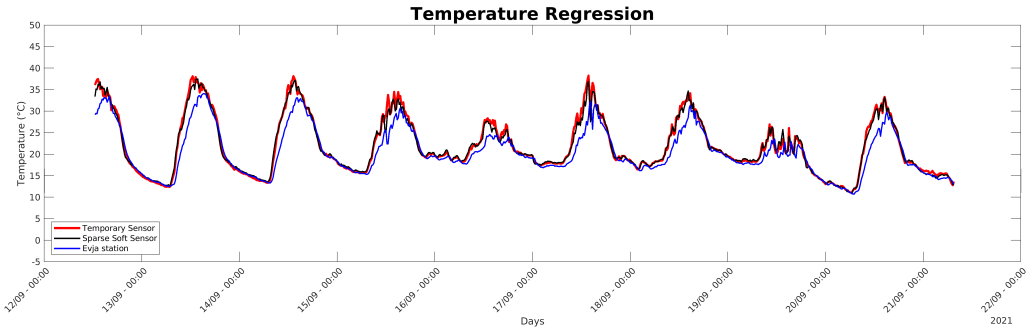


Figure 4.8: Plot for validation phase of regression for sensor #9.

4.5 Impact of temporal information on the model

In order to test how much the time information impacts the regression model, two tests were conducted. In the first one, named *Unmodified-output test*, we aimed to assess the difference between a traditional regression model obtained using the available dataset and a regression model computed with the addition of temporal information. In the second test instead, named *Modified-output test*, the same operations from the first one were performed, only with outputs altered in order to introduce a local daily event, in our case by forcing a temperature decrease of 5°C only between the 16:00 and 18:00 every day. In this way, an important recurring event, although synthetic, was introduced in the output dataset to see how well-timed and un-timed regression models were capable of handling it.

Table 4.3 reports the values of the accuracy metrics for each test and each model. In the case named "Modified-output", the timed model is more accurate since it allows for coping with the variation introduced by the presence of the local phenomenon, which persistent sensors cannot detect. Interestingly, even for the unmodified output case, the timed model performs better than the un-timed one with an average accuracy increase of 2% (0.99 against 0.97). Such difference can be explained by the presence of local phenomena, which may also affect the persistent sensors and which the timed model still manages to capture differently from the un-timed one, thus resulting in an increase in accuracy. Figure 4.9 compares the behavior of the two models in the Modified-output case: the timed model (red line) is much closer to actual values (black line) with respect to the un-timed one (blue line), with a difference in accuracy of 4%.

Output	Metric	Timed Model	Un-timed model
Unmodified-output	R^2_{Adj}	0.99	0.97
	Adj RMSE	0.03	0.04
	RRSE	0.11	0.16
Modified-output	R^2_{Adj}	0.99	0.95
	Adj RMSE	0.07	0.07
	RRSE	0.21	0.21

Table 4.3: Metrics for each model and test. The timed model is always more accurate than the un-timed one.

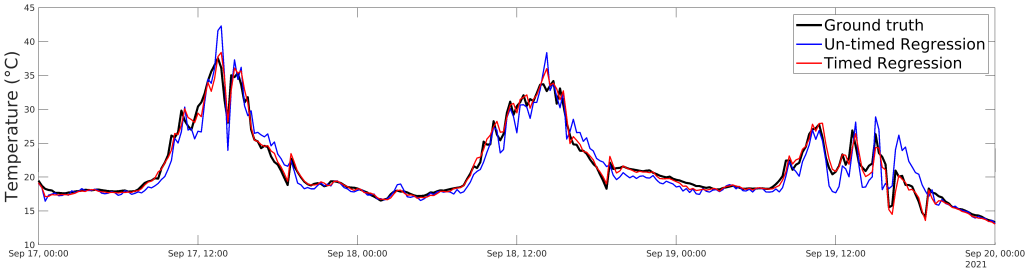


Figure 4.9: Accuracy comparison for un-timed and timed models when a local recurring event occurs.

4.6 Integration of weather forecast

Weather forecasts for the geographical zone where the experiment was conducted were retrieved from a specialized website, which provided an estimation for temperature, air pressure, and relative humidity for each hour of the day, assumed to be average values for the one hour the forecast refers to. Therefore, we modified the target vector Y to store hourly average readings, taken as the mean between all measurements done within a -30 and $+30$ minutes span from a given timestamp. For example, if a forecast refers to 10:00 am, the corresponding value in the Y vector was computed as the average between the sensors' readings from 9:30 a.m. to 10:30 a.m.

Regression results for conducted tests show that model accuracy reached an R-squared value greater than 0.8 for both the calibration and validation phases. Fig. 4.10 shows the actual temperature measurements (in blue) of the weather station near the greenhouse, the temperature forecast used (taken the prior day) related to a nearby town (in black), and the corresponding predicted greenhouse temperature forecast (in red). The obtained model provides a good estimation of future conditions near the greenhouse for most of the time, even if we used a free forecast web service, which had limited accuracy. Therefore, we believe a better forecast service can easily surpass the reported performance.

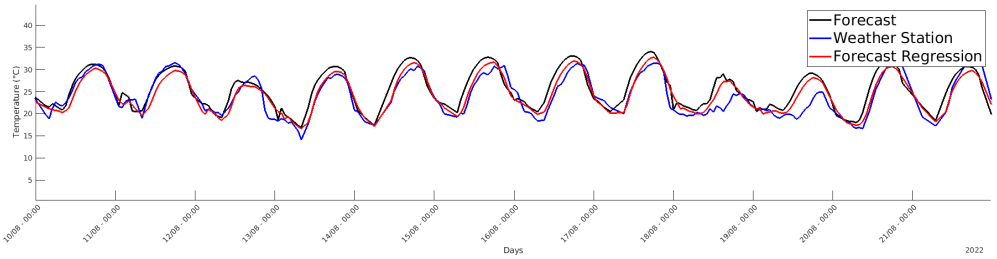


Figure 4.10: Validation of the forecast model against the original forecast and actual sensed values.

4.7 2D map generation and validation

After modeling fourteen temperature points of interest as *Sparse Soft Sensors*, the area delimited by them was split into a grid whose points were used to design the 2D map. The *2D-Sensor* was implemented by using MATLAB's `griddata` interpolation function fed by positions and estimated values of all the *Sparse Soft Sensors*. This function is called every time a new set of measurements is available for the *Sparse Soft Sensors* (i.e. when new samples from the persistent sensors are available).

*2D-Sensor*s validation was done by comparing, for each of the fourteen monitored spots, the actual sensed value, the estimation from the corresponding *Sparse Soft Sensors*, and the *2D-Sensor*'s estimation obtained by interpolation of all *Sparse Soft Sensors*s, but the one in examined position. Fig. 4.11 shows the values for a central location (upper plot) and for a corner location

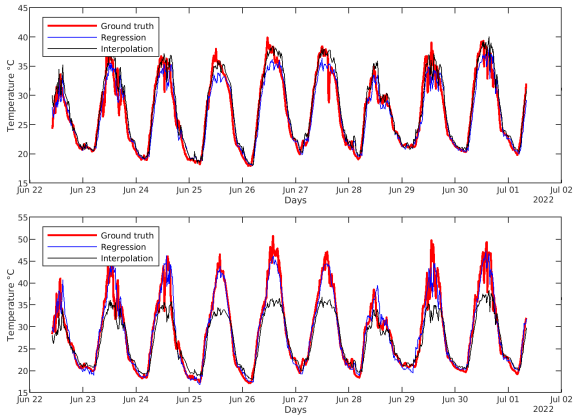


Figure 4.11: Comparison between estimated values with *Sparse Soft Sensors* and interpolation for two locations in the map.

(bottom plot). Results show that the difference between the *Sparse Soft Sensors* and interpolated

values grows more prominent for points closer to the border, while near the center, interpolation manages to be even more accurate than the *Sparse Soft Sensors*. We can, therefore, conclude that *Sparse Soft Sensors* are necessary for peripheral points of interest, while for central points of interest, the *2D-Sensor* may be sufficient, and the creation of specific *Sparse Soft Sensors* is not needed. Finally, Fig. 4.12 gives instead an idea of how the final result of the interpolation process should look like, basically creating a heat map of the monitored area to let farmers know at first glance how the situation inside the greenhouse is.

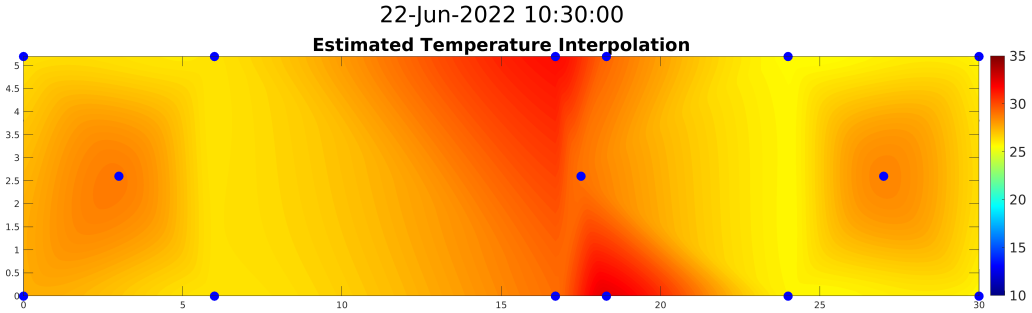


Figure 4.12: Temperature map obtained by combining estimation from *Sparse Soft Sensors* and the *2D-Sensor*.

4.8 Validation of anomaly detection

To validate the anomaly detection methodology previously explained, a model of each persistent sensor was built based on data coming from the weather station and vice versa. Results showed that for physical variables that both stations monitored, prediction accuracy managed to be very accurate ($R^2 > 0.8$), whereas the prediction of variables sensed only by one station (e.g., the soil moisture not present in the weather station) was less precise. Fig. 4.13 shows the original behavior of the persistent temperature sensor, its fault-injected version, and the predicted behavior by using data from the weather station. The anomalous behavior of the sensor is denoted by the difference between its current output and the corresponding prediction, and therefore, it is possible to continuously check the validity of a data source simply by using incoming data from the other one.

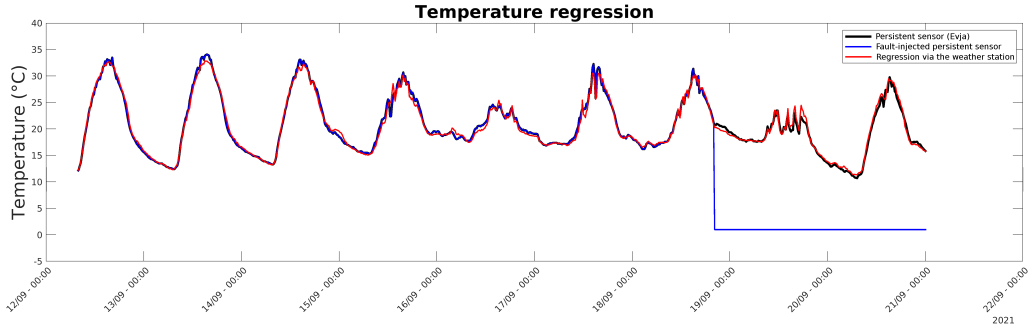


Figure 4.13: Comparison between the faulted sensors and model's predicted values.

4.9 Improving model robustness

The presented methodology has as its founding element the use of linear regression to develop models capable of estimating present or future greenhouse conditions, be they climatic or related to sensors' status. Throughout linear regression, we manage to obtain a model capable of combining various inputs to estimate a certain quantity with relatively good accuracy, as the presented results show. Although simple to use, however, linear regression models suffer from severe limitations in the sense of not being very robust. Despite the fact that the presented methodology can be used to detect faults, a noisy or missing value for even a single input variable may lead to unpredictable behavior, making the model unusable. While there are ways to work around these problems, such as using redundant models with fewer input variables to cover for cases of missing readings, these

solutions may not be practical. Therefore, it could be beneficial to move on from linear regression to a more robust technique for creating models capable of dealing with unexpected input behavior. In this regard, Recurrent Neural Networks (RNNs) offer a promising alternative, as they can handle the absence of measurements and outlier values much better than traditional linear regression. In particular, the usage of *Long Short Term Memory* (LSTM) networks appears to be a good solution, as they handle better long-term dependencies between input variables, are less susceptible to the vanishing gradient problem, and are very efficient at modeling complex sequential data. In addition, the introduction of the concept of *context* [113, 114], which enriches the measure with elements such as position and acquisition time, allows to boost the model's effectiveness furthermore.

Chapter 5

Crop Model Description and Calibration

The proposed extended automaton implements the reduced TOMGRO model. However, to take advantages the most out of automaton composition, the model was break down into its core components, and for each one an ad-hoc automaton was developed, allowing us to avoid the explosion in states number that we would have obtained by describing with a single automaton all model's possible behaviors. In this section we will provide only the key information for each automaton, while all formal specification are reported in table 5.1,appendix A

The first step to convert the model was to re-write it using only discrete-time variables, which allow for an easier conversion later on. This was made possible due the fact that all equations presented previously were either daily-based or hourly-based, with no actual continuous time involved. Therefore, the transition from the formulation given by original authors and the one we propose here, which uses discrete time, does not introduce any error, as we simply rewrite the equations to exhibit their innate property of being at discrete-time. In each developed DTHA, constructed following rules expressed in Section 2.7, we introduce the temporal variable c that evolve in continuous time over a give period T_p, after which it is reset to zero, thus acting as a sort of clock for the system and allowing to separate time into discrete moments. The total period T was set to be one-hour long, meaning that once c reaches the value of one, an hour as passed and we reset it. T_p was chosen to be hourly-based since one hour is the minimum granularity required by all TOMGRO's equations.

The decomposition process ended up in three distinct models, each one representing a core element for crop's growth: node, LAI and fruits development. Each model possesses its inputs, that can be shared across models, and eventually may depends on values computed by other models. This last statements may seem contradictory with the idea of working using separated models, but since the dependency graph of the model's equations turns out to be a DAG, it means that it is possible to put the models in a cascade formation, where for each level an automaton depends only on external inputs and value from automata at the upper value, without any circular dependency. This permit us to model automaton stating from the top, where no dependencies are presents, and then to move to lower level without worrying of changes propagating backwards. Figure 5.1 gives a visual representation of what stated so far.

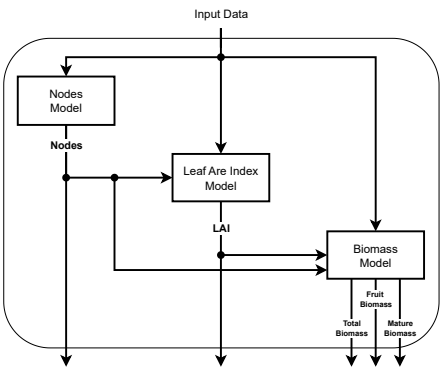


Figure 5.1: Diagram representing dependencies within the TOMGRO model.

Before proceeding with discussing the actual sub-models, we report here table 5.1, which reports information for each variable and parameters which will appear in equation we will discuss throughout this chapter.

Table 5.1: Legends of state variables, functions and parameters for the TOMGRO model.

Name	Meaning	Reference Eq.	Source(s)
	State Variables		
N	# of crop's nodes	2.1,5.2	[21]
LAI	Leaf Area Index (m^2leaf/m^2 ground)	2.2,2.7,5.3	[21, 22]
W	Aboveground dry weight (g/m^2)	2.3,5.7	[21]
W_f	Total fruit dry weight (g/m^2)	2.4,5.8	[21]
W_m	Mature dry weight (g/m^2)	2.5,5.9	[21]
P_g	Gross photosynthesis ($g[CH2O]/m^2$)	2.8,5.10,5.12	[19]
R_m	Maintenance respiration (($g[CH2O]/m^2$))	2.9,5.11,5.13	[21]
	Inputs		
T	Hourly mean Temperature (°C)		
T_m	Daily mean temperature (°C)	5.14	
T_{dm}	Daily daytime mean temperature (°C)	5.15	
CO_2	Hourly average CO_2 concentration		
PPFD	Hourly average photosynthetic photon flux density (μmol/sm^2)		
	Parameters and Auxiliary functions		
N_m	Maximum rate of node appearance at optimal temperature	[21]	
N_b	Coefficient in expolinear equation	2.2	[21]
f_N	Function to modify node development depending of temperature	5.16	[19]
ρ	Plant density per m^2	2.2,2.2,2.3,5.7	[21]
σ	Maximum leaf area expansion per node	2.2	[21]
λ	Function to reduce rate of leaf area expansion	2.2	[21]
β	Coefficient in expolinear equation	2.2	[21]
a	Parameter in sigmoidal equation	2.7,5.3	[22]
b	Parameter in sigmoidal equation	2.7,5.3	[22]
LAIMAX	Maximum leaf area expansion	2.7,5.3	[21]
d	Parameter in sigmoidal equation	2.7,5.3	[22]
GR_{net}	Net aboveground growth rate	2.6	[21]
$p1$	Loss of leaf dry weight per node after LAIMAX is reached	2.3,5.7	[21]
V_{max}	Maximum increase dry weight per node	2.3,5.7	[21]
α_f	Maximum partitioning of new growth to fruit	2.4,5.8	[21]
F_f	Function to modify partitioning to fruit w.r.t temperature	5.20	[115]
θ	Transition coefficient between vegetative and full fruit growth	2.4,5.8	[21]
N_{ff}	Nodes per plant when first fruit appears	2.3,2.5,5.8,5.9	[21]
T_{Crit}	Mean daytime temperature above which fruit abortion starts	2.4,5.8	[21]
K_{ff}	Nodes from first fruit to first ripe fruit	2.55.9	[21]
D_f	Rate of fruit's development w.r.t average daily temperature	5.21	[115]
E	Ratio of biomass to photosynthate available for growth	2.6	[19]
f_R	Function to modify partitioning to roots w.r.t nodes number	5.18	[19]
cfgt	Parameter in equation	5.20	[115]

5.1 Nodes

The first implemented automaton refers to the Node state variable, being it the only one without any dependencies except from external input and also being the one all other variables depend on.

TOMGRO's original formulation for node development was given in (2.1) and is reported here for simplicity. Here f_n is a reduction factor that can be defined either as in equations 5.16 and 5.17, while N_m is the maximum daily node appearance rate.

$$\frac{dN}{dt} = N_m f_N(T_m)$$

The discrete version of equation (2.1) is given by equation (5.1):

$$N(d) = N(d-1) + N_m f_N(T_m) \tag{5.1}$$

where d is a discrete-time variable representing the current day, while all other parameters retain their original meaning. Since, however, we decided to have an hourly-based time step, switching from a daily-based increment as the one proposed to an hourly-based one could prove beneficial, as working with hourly input may help in describing better plant's response to big variations in temperature, otherwise, hidden when by average values. Luckily, the formulation of equation (5.1) allows us to easily change the time scale for the increment by changing the N_m parameter from expressing maximum *daily* node appearance to maximum *hourly* node appearance. Moreover, since the original N_m parameter had to be calibrated in any case, changing it to be hourly-based does not add additional work. Therefore, the final formulation for the node's growth in our model is:

$$N(h) = N(h-1) + N_{mh} f_N(T) \tag{5.2}$$

where h is a discrete variable that represents the current *hour*, N_{mh} is the hourly maximum node appearance, and f_N is the reduction function. To stick with the original TOMGRO, we decided to implement such a function using its original formulation (eq. 5.16), and from now on, we will only refer to it when talking about the f_N function. T represent instead the *hourly* mean temperature, to distinguish from T_m being the *daily* mean temperature

Translating equation (5.2) into a discrete-time hybrid automaton is relatively straightforward, as it can be represented with a single state with a self-loop, updating the state variable. Figure 5.2 depicts the resulting automaton, stutter transitions are omitted to simplify the reading process.

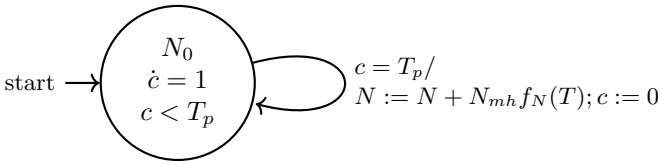


Figure 5.2: Node Automaton

5.2 Leaf Area Index

Following node modeling, we moved to model the Leaf Area Index (LAI). Back in section 2.3, we described that such a variable was computed in two different ways by two studies. In the TOMGRO model, LAI evolution is given by an expolinear function represented by the differential equation shown in (2.2); on the other hand, Bacci *et al.*described the LAI evolution via a sigmoidal function that binds LAI and nodes number directly as depicted in 2.7. Out of the two possibilities, we decided to implement our model using Bacci's formulation, as the sigmoidal growth better matches the actual evolution of leaf area for tomato crop [116].

$$\mathrm{LAI}(N) = \frac{ab + \mathrm{LAIMAX} \cdot N^d}{b + N^d}$$

While chosen formulation could be already implemented as-is, however, as previously appointed, Bacci's formulation provides computes the LAI value directly from nodes's number, meaning LAI becomes a stateless variable directly tied to the nodes. Because of this, any changes in nodes will directly and inevitably reflect on LAI's value even when not necessary, and, even worse, it becomes impossible to change the LAI value, for example to reset it as a consequence of a direct observation, without changing the node's value first. To solve this limitation, we decided to derive Bacci's formulation over $\frac{dN}{dt}$, turning the equation from a closed solution over the node's number into a differential equation over time:

$$\begin{aligned}\frac{d\mathrm{LAI}}{dt} &= \left(\frac{ab + \mathrm{LAIMAX} \cdot N^d}{b + N^d}\right)\frac{dN}{dt} \\ &= \frac{bd(\mathrm{LAIMAX} - a)N^{d-1}}{(b + N^d)^2}\end{aligned}$$

With this change now the LAI becomes a proper state variable, and its dependence from nodes is weakened. Now, in fact, it is the LAI increment that is computed using nodes rather and not its actual value, thus allowing us to easily solve the problems we previously addressed.

Following our choice, the next step would be to turn such equation into a discrete form. Moreover, the derivation process returned an equation expressed on a daily basis, meaning that if we

wish to express such a formula on an hourly basis, the result must be divided by a factor of 24. This leads to the final formulation for the finite difference equation expressing the hourly evolution of the LAI state variable, expressed in equation (5.3).

$$\mathrm{LAI}(h) = \mathrm{LAI}(h-1) + \frac{bd(\mathrm{LAIMAX} - a)N(h-1)^{d-1}}{\left(b + N(h-1)^d\right)^2} \tag{5.3}$$

Thanks to the derivation process, we managed to obtain a finite difference equation to express LAI change over time, but in doing so, we also lost track of a crucial LAI behavior. According to the original formulation, in fact, zero growth in nodes causes the LAI to not change at all due to the original function receiving two identical inputs. Applying such logic to the derived function, however, does not work since two identical inputs would lead to two identical increments, which would cause the LAI to grow even in conditions where it should not be possible. Therefore, the special case of zero growth had to be handled to recreate TOMGRO's original behavior in full. To do so, we decided to change equation (5.4) to (5.6).

$$\mathrm{LAI}(h) = \mathrm{LAI}(h-1) + \Delta\mathrm{LAI}(h) \tag{5.4}$$

$$\Delta\mathrm{LAI}(h) = \min(\delta\mathrm{LAI}, \max(0, \mathrm{LAI}_c(h) - \mathrm{LAI}_c(h-1))) \tag{5.5}$$

$$\delta\mathrm{LAI}(h) = \frac{bd(\mathrm{LAIMAX} - a)N(h-1)^{d-1}}{\left(b + N(h-1)^d\right)^2} \tag{5.6}$$

Basically, alongside regular LAI, we also introduce a second variable used for control, LAI_c, which is updated following Bacci's formulation. For every time step, we first compare the new control variable with its value at the previous time step to compute the increment Bacci's formula would provide. Secondly, we take the minimum between such increment and the one computed thanks to the equation derived in (5.3). In this way, we can guarantee LAI's update will only occur when conditions allow while also allowing LAI to grow without being strictly tied to nodes.

The resulting automaton is shown in figure 5.3 (stutter transitions are omitted), which consists of a single location, similar to the nodes' one, with a self-loop that updates the LAI variable every time c reaches the given time step, synchronously with the update of nodes.

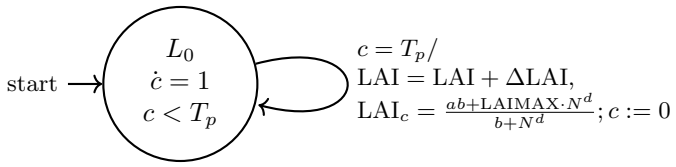


Figure 5.3: Automaton for Leaf Area Index

5.3 Fruits and Biomass

While nodes' number and leaf area index allow us to have a general idea of how the crop is growing, these variables do not permit us to estimate how much the crop has produced in terms of fruit weight. For this reason, alongside previous state variables, TOMGRO computes three more state variables on a daily basis: W, the *total dry weight* (i.e. biomass weight after removing all the water) of the crop, root excluded; W_f, which accounts the dry weight of all fruits, independently of their maturation; W_m, which holds information about the dry weight only for fruits that have reached full maturation and are therefore ready for harvest. Equations expressing the dynamic of such variables have already been shown in section 2.3, equations (2.3) to (2.5), but are also reported down below for ease of readability.

$$\frac{dW}{dt} = \mathrm{MIN}(\mathrm{GR}_{\mathrm{net}} - p_1\rho\frac{dN}{dt}, \frac{dW_f}{dt} + (V_{\max} - p1)\rho\frac{dN}{dt})$$

$$\frac{dW_f}{dt} = \begin{cases} 0 & \text{if } N < N_{\mathrm{FF}} \\ \mathrm{GR}_{\mathrm{net}}\alpha_f F_f(T_m)(1 - e^{-\theta(N-N_{\mathrm{FF}})}) & \text{if } T_{dm} \le T_{\mathrm{crit}} \\ \mathrm{GR}_{\mathrm{net}}\alpha_f F_f(T_m)(1 - e^{-\theta(N-N_{\mathrm{FF}})})(1 - 0.154(T_{dm} - T_{\mathrm{crit}})) & \text{if } T_{dm} > T_{\mathrm{crit}} \end{cases}$$

$$\frac{dW_m}{dt} = \begin{cases} 0 & \text{if } N \le N_{\mathrm{ff}} + K_{\mathrm{ff}} \\ D_f(T_m)(W_f - W_m) & \text{if } N > N_{\mathrm{ff}} + K_{\mathrm{ff}} \end{cases}$$

Since all equations represent the crop's biomass evolution throughout different growth stages, the three state variables were condensed into a single automaton representing the biomass change in tomato crops. In particular, from the given equation, it is possible to divide the crop's growth into three separated phases, depending on nodes' number, which dictates how the three state variables evolve: in the first phase the plant is still relatively young, therefore only the total dry weight increase, indicating a growth in the stem and other organs; in the second phase the crop has reached adulthood and begins to develop fruits, and the correlated state variable begins to evolve accordingly; finally once the first fruits have reached maturity, the last state variable starts evolving alongside the other two. Figure 5.4 visually represents the three phases the tomato plants' growth was split into.

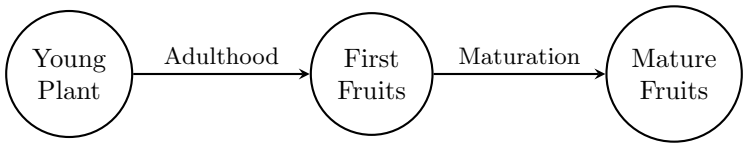


Figure 5.4: Phases of tomato's growth

Alongside the equations ruling biomass accumulation, the TOMGRO model also specifies the equations needed to compute the crop's photosynthesis, which is then converted into actual biomass. The three equations ruling such process are eq. (2.6),(2.8),(2.9), and are reported here for simplicity.

$$\begin{aligned}
\mathrm{GR}_{\mathrm{net}} =& E(P_g - R_m)(1 - f_R(N)) \\
P_g =& \int \left(\frac{D\tau CO_2 \mathrm{PgR}(T_m)}{K} ln \left(\frac{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD}}{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD} \cdot e^{-K \cdot \mathrm{LAI}}} \right) \right) dh \\
R_m =& \int \left(Q_{10}^{\frac{T-20}{10}} \mathrm{rm} \cdot (W - W_m) \right) dh
\end{aligned}$$

In such equations, T_m is the daily mean temperature, T_{dm} is the daily daytime mean temperature, CO_2 is the hourly concentration of CO_2, and PPFD is the hourly average photosynthetic photon flux density.

As already explained in past sections, the first step to rewrite TOMGRO's equations in a discrete-time form. This can be done almost trivially since such formulations were already discrete from the start, as the equations evolved at a fixed time step, either daily or hourly. Therefore, the discretization process for given formulas simply substitutes the derivative operator with finite differences and the integral one with a summation. The result of such a procedure is shown in equations (5.7) to (5.11).

$$W(d) = W(d-1) + \mathrm{MIN}(\mathrm{GR}_{\mathrm{net}} - p_1 \rho \frac{dN}{dt}, \frac{dW_f}{dt} + (V_{\max} - p1)\rho \frac{dN}{dt}) \tag{5.7}$$

$$W_f(d) = \begin{cases} 0 & \text{if } N < N_{\mathrm{FF}} \\ W_f(d-1) + \mathrm{GR}_{\mathrm{net}} \alpha_f F_f(T_m)(1 - e^{-\theta(N - N_{\mathrm{FF}})}) & \text{if } T_{dm} \leq T_{\mathrm{crit}} \\ W_f(d-1) + \mathrm{GR}_{\mathrm{net}} \alpha_f F_f(T_m)(1 - e^{-\theta(N - N_{\mathrm{FF}})})(1 - 0.154(T_{dm} - T_{\mathrm{crit}})) & \text{if } T_{dm} > T_{\mathrm{crit}} \end{cases} \tag{5.8}$$

$$W_m(d) = \begin{cases} 0 & \text{if } N \leq N_{\mathrm{ff}} + K_{\mathrm{ff}} \\ W_m(d-1) + D_f(T_m)(W_f - W_m) & \text{if } N > N_{\mathrm{ff}} + K_{\mathrm{ff}} \end{cases} \tag{5.9}$$

$$P_g(d) = \sum_{h=1}^{24} \frac{D\tau CO_2 \mathrm{PgR}(T)}{K} ln \left(\frac{(1-m)\tau CO_2 + Q_e \cdot K \cdot 5\mathrm{PPFD}}{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD} \cdot e^{-K \cdot \mathrm{LAI}(h-1)}} \right) \tag{5.10}$$

$$R_m(d) = \sum_{h=1}^{24} Q_{10}^{\frac{T-20}{10}} \mathrm{rm} \cdot (W(d-1) - W_m(d-1)) \tag{5.11}$$

Note that equation (2.6) is not present in the list of adapted formulas. This is due to the fact that equation (2.6) is simply an algebraic formula and is not tied to time dynamics, meaning no changes are required to it. It is instead important to underline that the last two adapted equations, namely the total daily gross photosynthesis P_g and the maintenance respiration r_m, are computed

as the sum over 24 hours of the hourly value of photosynthesis and respiration. This allows us to break down such variables into hourly steps rather than daily-based, meaning that equations (5.10) and (5.11) can be rewritten as seen in (5.12) and (5.13).

$$P_g(h) = P_g(h-1) + \frac{D\tau CO_2 \mathrm{PgR}(T)}{K} ln\left(\frac{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD}}{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD} \cdot e^{-K \cdot \mathrm{LAI}(h-1)}}\right) \tag{5.12}$$

$$R_m(h) = R_m(h-1) + Q_{10}^{\frac{T-20}{10}} \mathrm{rm} \cdot (W(d-1) - W_m(d-1)) \tag{5.13}$$

Therefore, the complete discrete model representing TOMGRO's biomass growth is given by combining the previous two equations with equations (5.7) to (5.9). It must be noted, however, that the two sets of equations evolve with different granularity: in the case of dry weight, variables evolve over a daily interval, whereas nutrient production evolves on an hourly basis and accumulates over 24 hours before resetting for the next day. From this consideration, it follows that the resulting automaton must use the finer granularity between the two possible, i.e., evolve on an hourly basis. Therefore, the resulting automaton still works using an established time-step, albeit this time coupled with the variable h responsible for counting the elapsed hours and detecting when a full day has passed ($h = 24$) to perform the daily update and reset such variable to zero. Simultaneously, *average daily temperature* and *average daytime temperature* are also computed alongside updating the h variable. Formulas accounting for the update of the two temperatures are shown in equations (5.14) and (5.15), with T being the mean hourly temperature.

$$T_m(h) = \frac{T_m(h-1)(h-1)}{h} + \frac{T}{h} \tag{5.14}$$

$$T_{dm}(h) = \begin{cases} \dfrac{T_{dm}(h-1)(h-1)}{h} + \dfrac{T}{h} & \text{if } 8 \leq h \leq 18 \\ T_{dm}(h-1) & \text{otherwise} \end{cases} \tag{5.15}$$

The resulting automaton is pictured in figure 5.5, with table 5.2 acting as a legend for the used labels. Stutter transitions are omitted to simplify the reading process.

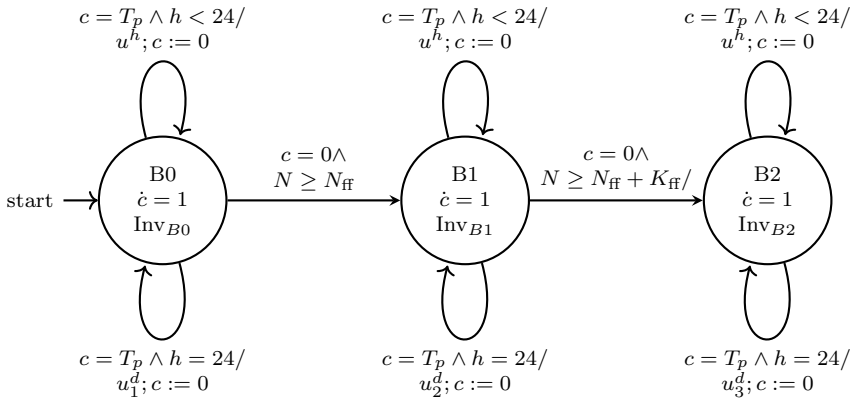


Figure 5.5: Automaton computing Biomass growth

Table 5.2: Legend expanding the labels pictured in fig. 5.5.

Label	Expression
u^h	$h := h + 1, P_g := (5.12), R_m := (5.13), T_m := (5.14), T_{dm} := (5.15)$
u_1^d	$W := (5.7) h := 0, P_g := 0, R_m := 0, T_m := 0, T_{dm} := 0$
u_2^d	$W := (5.7), W_f := (5.8) h := 0, P_g := 0, R_m := 0, T_m := 0, T_{dm} := 0$
u_3^d	$W := (5.7), W_f := (5.8) h := 0, W_m := (5.9), P_g := 0, R_m := 0, T_m := 0, T_{dm} := 0$
Inv_{B0}	$c < T_p \vee (c = 0 \wedge N < N_{\mathrm{ff}})$
Inv_{B1}	$c < T_p \vee (c = 0 \wedge N < N_{\mathrm{ff}} + K_{\mathrm{ff}})$
Inv_{B2}	$c < T_p$

5.4 Algebraic functions

Previous sections focused solely on equations ruling the dynamic of state variables. Such formulas, however, heavily rely on other functions to describe limiting factors or crop's response to climatic changes, for example, the reduction in nodes' growth under sub-optimal temperatures. These functions are computed at each step, and are time-independent and agnostic of the model's state; i.e., they are pure, stateless, algebraic functions. In this section, we will describe them by providing their formulation.

The very first function used appears in equation (5.2) as f_N, the function responsible for reducing node development under sub-optimal temperatures. According to [19], this function is computed by interpolating points given in table 5.3, resulting in the formula presented in 5.16. A different formulation for f_N was instead given by Bacci *et al.*in [22], where the function was computed as in 5.17.

Table 5.3: Value used for Node reduction function from original TOMGRO.

Temperature	$f_N(T)$
0.0	0.0
9.0	0.0
12.0	0.55
28.0	1.0
50.0	0.0

$$f_N(T) = \begin{cases} 0.18\bar{3}T - 1.65 & \text{if } 9 < T < 12 \\ 0.028125T + 0.2125 & \text{if } 12 \leq T \leq 28 \\ \dfrac{2}{11} - \dfrac{T}{22} & \text{if } 28 < T < 50 \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

$$f_N(T) = \text{Cfnm} \cdot \min\left(\min\left(0.25 + 0.025T, 2.5 - 0.05T\right), 1\right) \tag{5.17}$$

Figure 5.6 shows graphically the differences between formulation given in 5.16 and 5.17: the former equation penalizes development under low temperatures much more with respect to the second one, which instead penalizes higher values of temperature.

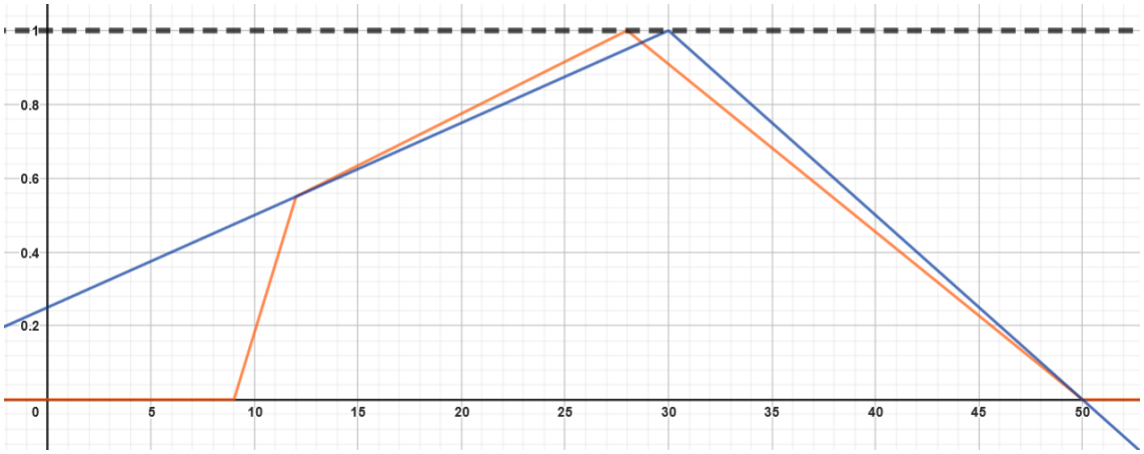


Figure 5.6: Curves for the two formulation of the f_N functions: orange is 5.16 and blue is 5.17

In this work, we choose to implement the f_N function according to the original TOMGRO's implementation, as it is more flexible with respect to Bacci's. Moreover, the four temperature points (that in the original table corresponded to $9, 12, 28, 50°C$) were set to be parametric to allow for better calibration during experiments. This means that giving a direct formulation of the f_N function is impossible, as it can only be interpolated by using the four temperature parameters after they have been set.

Proceeding with the equations, the second algebraic formula used is found in equation (5.7) in the form of $\mathrm{GR_{net}}$. Such formulation was already presented in section (2.6), and is computed as follow:

$$\mathrm{GR_{net}} = E(P_g - R_m)(1 - f_R(N))$$

Here, E represents the growth efficiency, while f_R indicates the fraction of biomass destined to roots depending on the nodes' number. Such function can be obtained from [19], which computes it by interpolating data we report here in table 5.4, therefore obtaining the formula (5.18)

Table 5.4: Value used in the f_R function in the original TOMGRO.

Nodes	$f_R(N)$
1	0.2
12	0.15
21	0.10
30	0.07

$$f_R(N) = \begin{cases} 0 & \text{if } N < 1 \\ 0.205 - 0.0045N & \text{if } 1 \leq N < 12 \\ 0.217 - 0.006N & \text{if } 12 \leq N < 21 \\ 0.17 - 0.003N & \text{if } 21 \leq N \leq 30 \\ 0.07 & \text{otherwise} \end{cases} \tag{5.18}$$

Similarly, the function $\mathrm{PgR}(T)$ that appears in equation (5.12) and computes the reduction of photosynthetic rate due to sub-optimal temperature values and is given too by interpolating data represented in table 5.5 (obtained again from [19]), leading to the formula as in (5.19)

Table 5.5: Value used for the photosynthetic rate reduction function in the original TOMGRO.

Temperature	$\mathrm{PgR}(T)$
0.0	0.0
9.0	0.67
12.0	1.0
28.0	1.0
35.0	0.0

$$\mathrm{PgR}(T) = \begin{cases} 0.074T & \text{if } 0 \leq T < 9 \\ 0.11T - 0.32 & \text{if } 9 \leq T < 12 \\ 1 & \text{if } 12 \leq T \leq 28 \\ 5 - \dfrac{T}{7} & \text{if } 28 < T \leq 35 \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

Moving on, the next function to consider is the one responsible for modifying biomass partitioning to fruit with respect to the average daily temperature, $F_f(T_{dm})$, appearing in equation (5.8). Of such function, however, no formulation was given in the considered paper [21], and a correspondent one could not be found in the original 1991 TOMGRO model.

Because of this, the function was instead adapted from the SGx software [115], which also runs using the TOMGRO model as its core. Therefore, the formula computing $F_f(T_{dm})$ was set as in (5.20), where $CfNFGT$ represents the reduction coefficient for tomato dry matter production.

$$F_f(T) = \max\Big(0, \min\big(1, 0.0625\,(T - \mathrm{cfgt})\big)\Big) \tag{5.20}$$

On a similar note, we have the function $D_f(T)$, which appears in equation (5.9) and computes the rate of development of fruit under daily temperature. Since such a function could not be established from either the TOMGRO model or its reduced version, such function was also imported from the SGx software as it appears in (5.21)

$$D_f(T) = \max\Big(0, \min\big(1, 0.0714 * (T - 9)\big)\Big) \tag{5.21}$$

5.5 Parameter Calibration

Previous sections described in detail all changes made to the original TOMGRO's equations in order to develop automata for the three main model's components. During such work, however, we focused only on the structure and the meaning of the equations themselves while leaving behind another crucial aspect of the model itself: its parameters. As with many other agronomical models, in fact, TOMGRO heavily relies on the value of such parameters to model crops' development and estimate their evolution characteristics that our formulation also inherits. Despite such importance, computing the correct value for all parameters is not a trivial task, mainly due to the fact that all the parameters are obtained from the field and hence related to crops, which are by nature heterogeneous. Therefore, computing the exact value for a specific parameter is almost impossible, as we can only estimate it from crops' behavior, which is driven by their genetics and growth conditions, two very fluctuating elements. As such, the only possible way to proceed is to search for the best parameters set which minimizes the total error and variance across all crops in the calibration sets. However, there is no guarantee that the same crop grown in different places, time or conditions will behave in the same way. Because of this, parameters' values are often left apart in publications, meaning the only way to access their values is to compute them by ourselves.

In this work, four datasets were made available. The first two refer to tomato cultivation of the Caramba tomato variety (*Solanum lycopersicum*, cv. Caramba) set in the facilities of DiSAAA-a, Pisa, from April to June 2005 and August to November of the same year. The other two datasets, instead, refer to a cultivation of tomato of the Jama variety (*Solanum lycopersicum*, cv. Jama), this time in the heated greenhouse of the Department of Crop Biology at the University of Pisa, from March to June and July to November 2001 respectively. Of the four, we decided to use the first two to estimate the parameters, as they contain more precise data related to nodes, LAI, and biomass over time, while we left the last two for validation. The estimation procedure followed the dependencies between the three automata, as we started by estimating parameters only for the nodes to move later to LAI and biomass. For nodes, estimation focused on the four temperature values delimiting the different reduction functions plus the maximum appearance rate N_{mh}; for LAI, the four parameters of the sigmoid were computed, and for biomass, all parameters for which a known value could not be found had to be estimated.

Parameter estimation was performed using a brute-force algorithm, which tested all possible parameters' combinations, selected within a given range, by setting them in the corresponding automaton and by running it. From the simulation's results, average Root Mean Square Error (RMSE) and variance were computed across all test sets. In the end, the combination of minimum RMSE and variance was chosen as the best possible values to use for parameters, as it provided us with a model providing the lower error and variability,therefore constantly performing well across the cases instead of extremely well in specific case and bad in others. Once estimation returned the parameters for a specific group of crops, the algorithm moved to the next one until all groups had been analyzed. To speed up the process, the algorithm was parallelized, allowing the split combination testing among different cores and processors. Subsequently, models were validated using the remaining two datasets. For each model, three metrics were used to estimate prediction goodness: *Root Mean Squared Error* (RMSE), the *Relative Error* (RE,[117]), Willmott's index of agreement d [118] and the Nash and Sutcliffe's model efficiency index NSE [119], defined as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x_i})^2}{N}}$$

$$\text{RE} = \frac{RMSE}{\bar{x}}$$

$$d = 1 - \frac{\sum_{i=1}^{N}(x_i - \hat{x_i})^2}{\sum_{i=1}^{N}(|x_i - \bar{x}| + |\hat{x_i} - \bar{x}|)^2}$$

$$\text{NSE} = 1 - \frac{\sum_{i=1}^{N}(\hat{x_i} - x_i)^2}{\sum_{i=1}^{N}\hat{x_i} - \bar{x}}$$

where x_i refers to predicted values, $\hat{x_i}$ refers to expected outputs and $\bar{x}$ is the mean value for expected outputs. Results for both calibration and validation are depicted in table 5.6, where c_1 and c_2 represent the two calibration sets from April-June and August-November 2011, respectively, while v_1 and v_2 are the two validation set from March-June and July-November respectively.

Figures 5.7 to 5.11 show gives a visual representation for validation phase results, specifically for the second validation set.

Dataset	RMSE	RSE	d	NSE
Nodes prediction accuracy				
c_1	0.89	0.05	0.99	0.98
c_2	0.27	0.01	0.99	0.99
v_1	1.20	0.05	0.99	0.98
v_2	0.94	0.04	0.99	0.99
LAI prediction accuracy				
c_1	0.04	0.07	0.99	0.99
c_2	0.14	0.20	0.98	0.91
v_1	0.12	0.21	0.97	0.86
v_2	0.24	0.26	0.96	0.88
Total Biomass prediction accuracy				
c_1	25.7	0.18	0.99	0.96
c_2	37.8	0.34	0.97	0.88
v_1	34.2	0.20	0.99	0.96
v_2	22.1	0.15	0.99	0.98
Fruit's Biomass prediction accuracy				
c_1	12.5	0.16	0.99	0.97
c_2	16.7	0.27	0.99	0.95
v_1	28.8	0.28	0.98	0.93
v_2	16.8	0.23	0.99	0.96
Mature Fruit's Biomass prediction accuracy				
c_1	18.9	0.34	0.98	0.93
c_2	19.9	0.59	0.97	0.88
v_1	43.8	0.75	0.93	0.81
v_2	11.2	0.21	0.99	0.97

Table 5.6: Model's accuracy using the four error indexes.

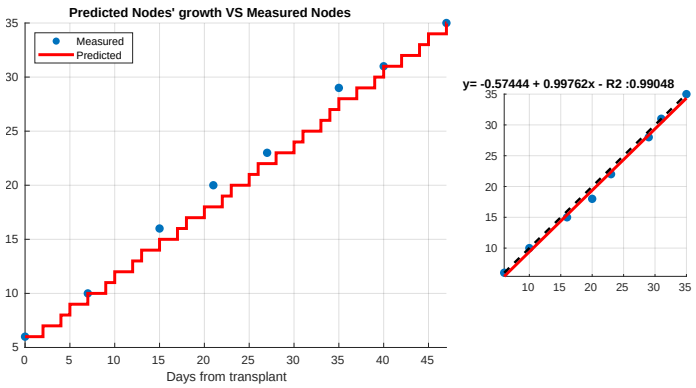


Figure 5.7: Predicted vs Estimated values for nodes' model

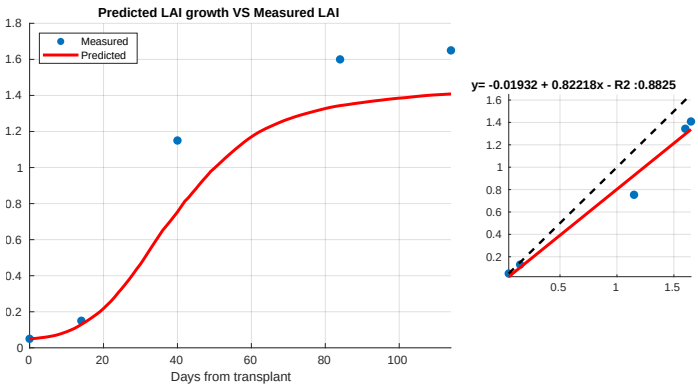


Figure 5.8: Predicted vs Estimated values for the LAI's model

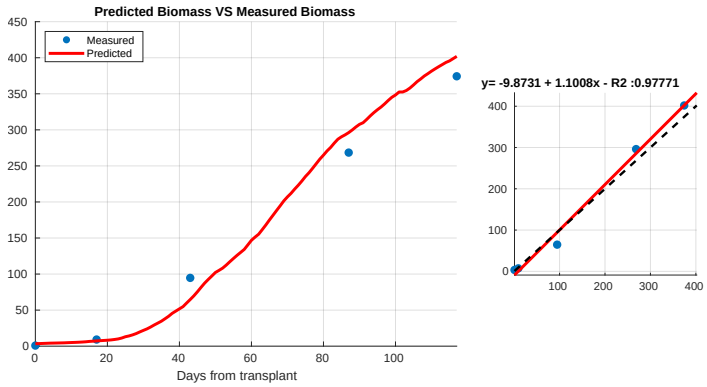


Figure 5.9: Predicted vs Estimated values for the total biomass' model

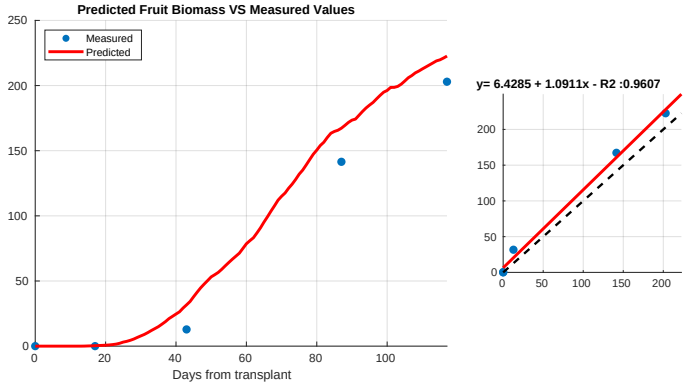


Figure 5.10: Predicted vs Estimated values for the fruit biomass' model

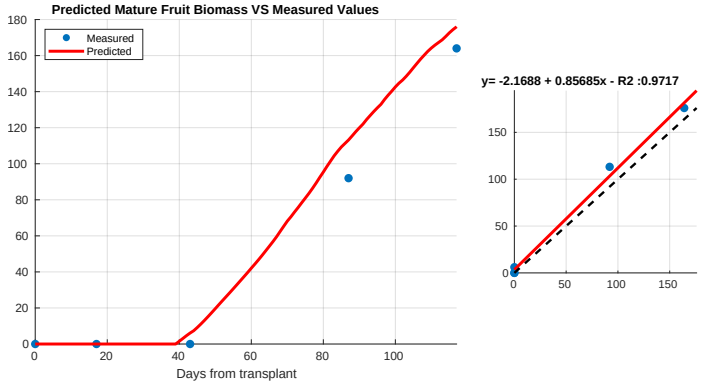


Figure 5.11: Predicted vs Estimated values for the mature-fruit biomass' model

Chapter 6

Disease Modeling

The previous chapter showed how automata formalism can be used to convert the popular tomato model TOMGRO from its original description to a more formal one based on automata. Furthermore, in the process, we also gave a small showcase of how automata composition works by using it to compose the three sub-models derived from the original description to obtain the simulation of the complete system. However, such an example does not show the true power of automata composition, as we simply re-assembled the original model we first divided. Therefore, alongside the tomato model, a disease model based on *Oidium Lycopersici* was adapted from literature and rewritten as a Discrete-Time Hybrid Automaton to be composed later with the crop model.

The model we decided to implement has already been explained in section 2.5 and is directly taken from [30]. The adaptation procedure followed the same rules dictated for the TOMGRO model, resulting in a set of finite-difference equations dictating the disease's progress, which were later used to develop the automaton implementing the pathogen's model. Furthermore, in order to better compose the two models later on, we also decided to change the pathogen's time-step by switching it from a daily-based one to an hourly one, therefore synchronizing it with all the crop's automaton, especially the one accounting for LAI development. To do so, we divided the Rate_Y term in equation 2.14 by 24 and used the hourly increment of sane leaf tissue instead of the daily one. Finally, just like the tomato model, the Oidium DTHA also shares the same time granularity T and clock variable c, dictating when updates must occur. Equations 6.1 to 6.3 shows the pathogen's discretized equation with hourly update:

$$H(h) = H(h-1) + \Delta H_s(h) - \Delta Y(h) \tag{6.1}$$

$$Y(h) = Y(h-1) + \Delta Y(h) \tag{6.2}$$

$$\Delta Y(h) = \begin{cases} 0 & \text{if } 0 \leq h < \text{IP} \\ r_{\text{LINmax}}\text{DE}(T, \text{RH}) & \text{if IP} \leq h < \text{LP} + \text{IP} \\ r_{Y\max}\text{DE}(T, \text{RH})Y(1 - \dfrac{Y}{H+Y}) & \text{otherwise.} \end{cases} \tag{6.3}$$

$$\text{DE}(T, \text{RH}) = \begin{cases} d(T - T_{\min})^n (T_{\max} - T)^m (1 - e^{-a\text{RH}})^b & \text{if } T_{\min} \leq T \leq T_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

where h denotes the hourly time step, ΔH_s is the hourly update to the sane leaf tissue, and IP and LP are the incubation and latent period respectively, expressed in hours. ΔY represents instead the rate at which the infection spread and is basically a renaming of the term Rate_Y, sharing the same formulation described back in equation 2.15. Finally, the *Disease Efficiency* function DE remains the same as in the original model, limiting pathogen growth under suboptimal conditions. Remaining parameters are listed all in table 6.1.

An important consequence of switching from a daily-based model to an hourly-based one can be observed when we consider the relationship between the LAI (Leaf Area Index) computed by the TOMGRO model and the H (absolute leaf area) variable computed by the disease model. By definition, LAI is obtained as the ratio between the absolute area covered by leaves and the ground surface, or, in other words, as the product between the plant density ρ and the absolute covered leaf area H. Therefore, we have that $\text{LAI} = \rho H$, while we can also rewrite the update to sane leaves as $\Delta H_s = \Delta\text{LAI}\rho$, with ΔLAI being computed as in the TOMGRO model, assuming it to

Table 6.1: Legends of state variables, functions and parameters for the Pathogen model.

Name	Meaning	Reference Eq.	Source(s)
	State Variables		
Y	Ill leaf area (m^2)	2.146.2,	[30]
	Inputs		
T	Hourly mean Temperature (°C)		
RH	Relative humidity (%)		
LAI	Leaf Area Index		
	Parameters and Auxiliary functions		
IP	Incubation Period	2.15,6.3	[30]
LP	Latent Period	2.15,6.3	[30]
DE	Disease Efficiency function	2.16	[30]
a	Parameter in function DE	2.16	[30]
b	Parameter in function DE	2.16	[30]
d	Parameter in function DE	2.16	[30]
n	Parameter in function DE	2.16	[30]
m	Parameter in function DE	2.16	[30]
Tmin	Parameter in function DE	2.16	[30]
Tmax	Parameter in function DE	2.16	[30]
rho	Crop Density	6.5	

be the LAI maximum possible hourly expansion. Subsequently, we can rewrite equation 6.1 as 6.5

$$\begin{aligned} H(h) &= H(h-1) + \Delta H_s(h) - \Delta Y(h) \\ \mathrm{LAI}(h)\rho &= \mathrm{LAI}(h-1)\rho + \Delta\mathrm{LAI}(h)\rho - \Delta Y(h) \\ \mathrm{LAI}(h) &= \mathrm{LAI}(h-1) + \Delta\mathrm{LAI}(h) - \frac{1}{\rho}\Delta Y \end{aligned} \tag{6.5}$$

meaning we can establish a connection between the TOMGRO model and the disease. Following this consideration, two separate automata were built starting from the pathogen model: one responsible for modeling the damage dealt to leaves by the pathogen and a second one modeling the life cycle of the disease. The choice to separate the life cycle from the damage it deals was driven by the fact that, thanks to composition, in the future, it will be possible to swap the pathogen life cycle with another one without the need also to replace the component responsible for computing damages dealt to leaves, which now becomes independent of the pathogen's life. Figure 6.1 gives a visual representation of the described concept.

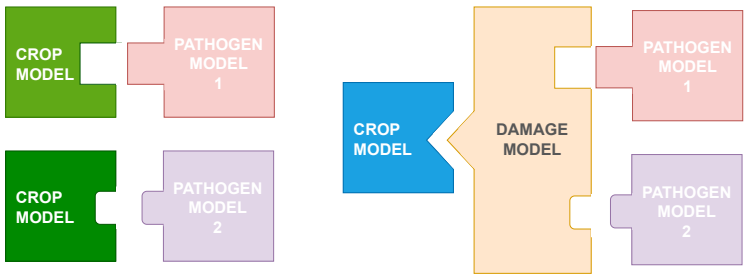


Figure 6.1: Benefits of separating the damage model and the pathogen life cycles. We can use the damage model as conjunction point between crop and disease without changing the plant model each time.

6.1 Oidium Lycopersici infection cycle

The model for the pathogen itself was derived from equations 6.2 and 6.3, coupled with considerations from the pathogen life cycle such as the distinction between latent and infective period or

primary and secondary infections. As a result, the five-state automaton presented in figure 6.2 was modeled, with table 6.2 expanding the model's invariant in each location. For simplicity, stutter transition were not reported in the figure.

The automaton starts in location P_0, which represents the absence of a pathogen, and remains in such location until the plant's LAI reaches a given threshold LAI_t, indicating the plant has reached a stage where leaves have grown enough in dimension to allow the pathogen's spore to land and start developing. This is, obviously, a simplification of what happens in reality, where the probability of spores landing depends on the presence of such spores in the air, their concentration, and lastly on the leaf tissue available, and even if all previous conditions are satisfied, there is no guarantee of them landing and starting an infection. However, to keep the model simple while avoiding stochastic behavior, we decided to put ourselves in a condition where spores are always present and are guaranteed to land on plants as long as enough leaf tissue is available, symbolized by the threshold LAI_t. Once the starting threshold has been exceeded, the automaton moves to location P_1, and the pathogen starts its life cycle. From location P_1, a counter starts to measure elapsed hours and detect the end of the incubation period, which coincides with when damages start to appear on leaves. The incubation period lasts until IP hours have passed, but within this time frame two separate phases can be detected: one which spans up to LP, coinciding with location P_1 where the pathogen grows without any effect, and a second one, represented by location P_1I, spanning from the end of LP to the end of IP, where the plant becomes actually infective for its neighbor. While the pathogen itself does not change its way of growing, we still decided to split up the two phases to emphasize the moment a plant may become contagious and, therefore, act as a starting point to infect its neighbors. As the incubation period (IP) ends, the plant starts showing the disease's symptoms-i.e., a white cover on the leaves. As a consequence, the automaton moves from location P_1I to P_2, which computes damage done by the primary infection. Location P_2 estimates damage dealt by the pathogen on leaves according to the first case of equation 6.3, meaning the pathogen follows a linear growth during the entire first infection cycle. As the incubation period ends and the spores start germinating, new infections begin on the same plant as consequences of released spores, meaning that following a second latent period a new infection will start. This is represented in the automaton with the last location, P_3, which is entered only after the total time from the infections begins to surpass another $IP + LP$ time period. Here, the infection dynamic changes, moving from a linear growth to a sigmoidal one, which better represents the damage done by secondary infections.

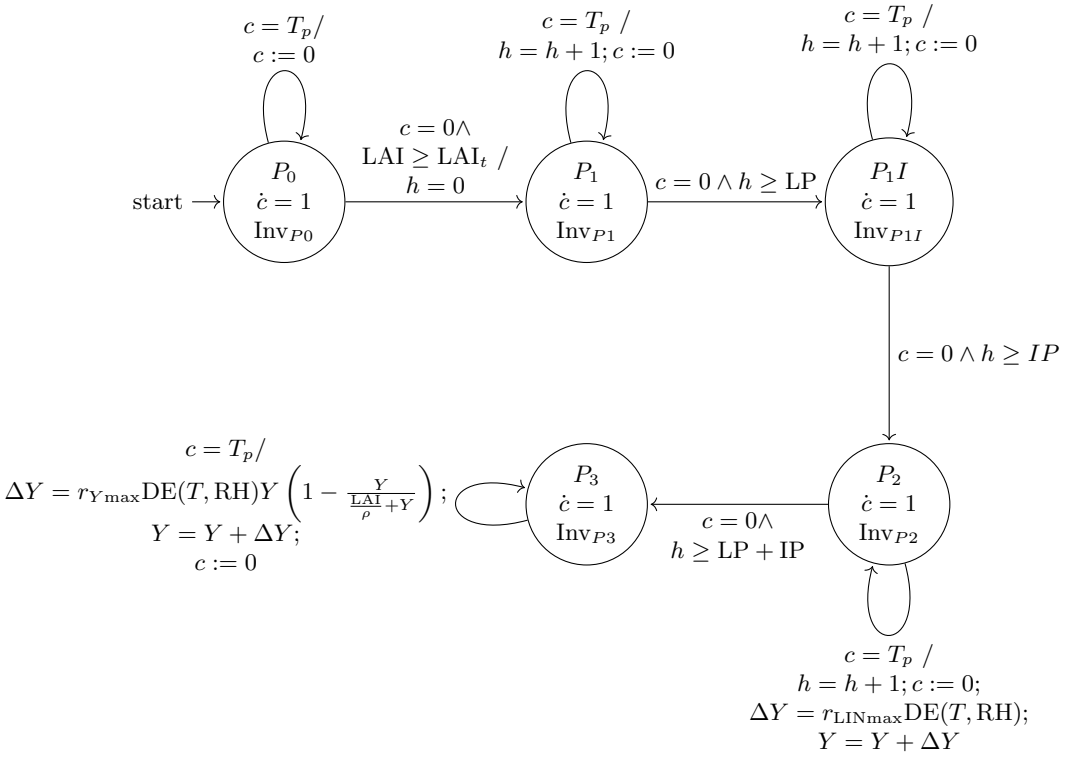


Figure 6.2: DTHA representing the infection cycle caused by *Oidium Lycopersici*.

Table 6.2: Invariant for automaton in fig. 6.2.

Label	Expression
Inv_{P0}	$c < T_p \vee (c = 0 \wedge \text{LAI} < \text{LAI}_t)$
Inv_{P1}	$c < T_p \vee (c = 0 \wedge h < \text{LP})$
Inv_{P1I}	$c < T_p \vee (c = 0 \wedge h < \text{IP})$
Inv_{P2}	$c < T_p \vee (c = 0 \wedge h < \text{LP} + \text{IP})$
Inv_{P3}	$c < T_p$

6.2 Damage Model Automaton

For how the pathogen automaton has been modeled, composing it as is with the adapted TOMGRO model does not produce the correct simulation behavior. As said earlier, in fact, in order for the two models to synchronize, we must introduce a third automaton capable of enabling communication between the crop model and the pathogen one. The automaton in question must handle two tasks: first, it must pass information about the LAI status to the pathogen needed by its growth function; on the contrary, it must also transmit information about ill LAI to the crop model. Figure 6.3

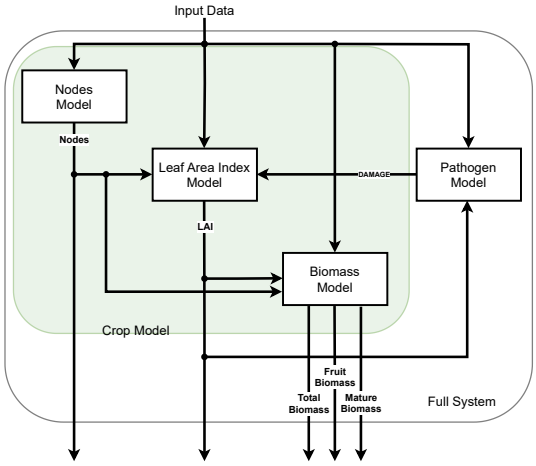


Figure 6.3: Updated diagram representing dependencies between the crop model and the pathogen model

clearly shows the relationship between the pathogen and the crop model, clearly evidencing how LAI is the conjunction point between the pathogen and the crop. As a consequence, our disease model can be derived by simply modifying the LAI automaton, with the caution of keeping it compatible both with the TOMGRO model and with the pathogen's automaton. The result of such an operation is shown in figure 5.3. Initially, this automaton behaves precisely like the original one, looping in the initial location and updating the LAI variable following the original model to maintain backward compatibility. However, as soon as the infection starts and damages begin to appear, the automaton switches location, moving into a new one where the only difference with respect to the first one lies in the LAI formulation, which is changed with equation 6.5. Therefore, while the disease increases, LAI's net growth is reduced up to the point it starts to decrease, meaning that the disease is consuming the sane tissue and causing, as a side effect, a reduction in biomass accumulation, strictly tied to LAI.

The new LAI automaton was inserted into the TOMGRO model, and the new model was composed alongside the pathogen model. Simulation results are depicted in figure 6.5 and 6.6, which plots both the simulation output for a healthy crop and for a crop infected by the pathogen. Plots clearly show damages dealt by the pathogen to the leaves and, as a side effect, on produced biomass, with a decrease of about 10% with respect to the maximum production, slightly below average estimated damages.

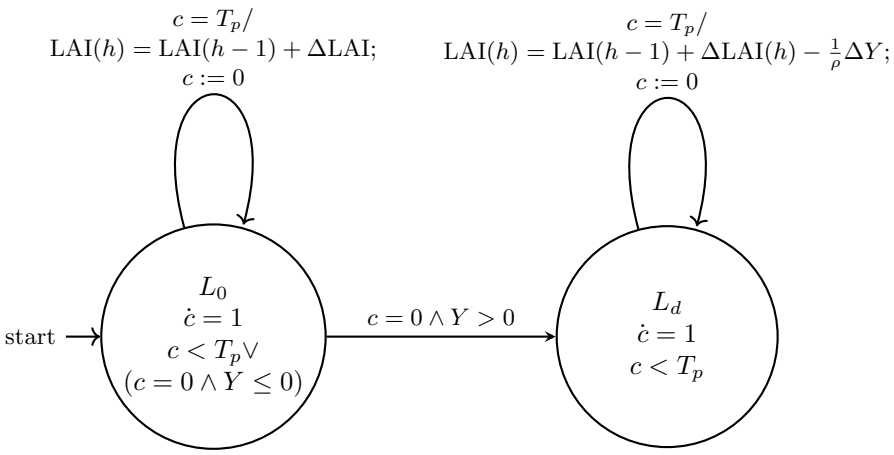


Figure 6.4: Automaton for Leaf Area Index

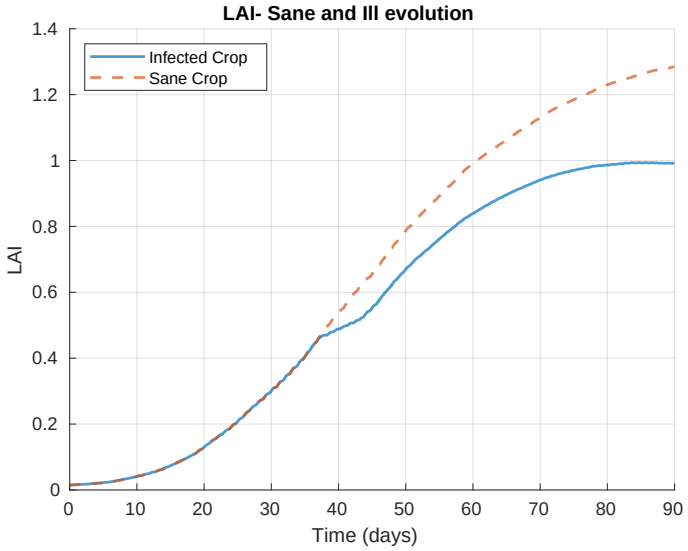


Figure 6.5: Simulation of an ill plant's LAI vs an healthy one

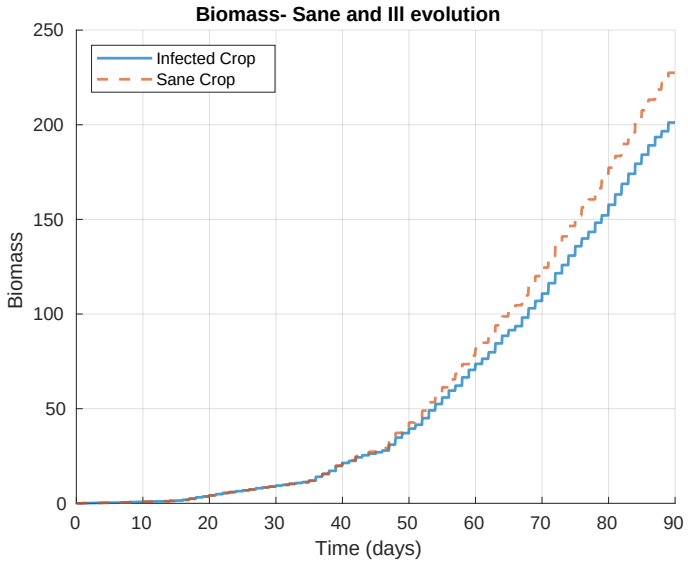


Figure 6.6: Simulation of an ill plant's biomass vs an healthy one

6.3 Modeling Interactions between Infected Crops

Results presented so far refer only to simulations of a single ill plant. In reality, however, multiple crops are present inside a greenhouse, and the pathogen can spread from a plant to nearby ones following traditional epidemic dynamics. Nevertheless, our models currently do not permit us to describe such a phenomenon, as even by composing multiple pairs of disease-crop, each pair would evolve independently with respect to others since all pathogens would activate as soon as the paired crop reaches sufficient leaves surface, meaning that, since all crops reach such threshold more or less at the same time, all crops would become infected almost at the same time, without the disease actually spreading.

To better understand how to model such behavior, we can consider the example shown in figure 6.7. In the picture, nine pairs of crops/pathogen (P_0 to P_8) are displaced in a 3x3 grid representing their spatial position, with only one of them, pair P_0 at position $(0, 0)$ being already infected and with the pathogen actively growing. After some time, cell $(0, 0)$'s pathogen reaches the point where it becomes infective and spreads the pathogen to its cardinal neighbors, this being pairs in position $(0, 1)$ and $(1, 0)$, causing their pathogen to grow and starting its cycle, which will prolong the infection chain and spread the disease even more.

Upon first consideration, using bidimensional cellular automata to model the described scenario may seem a good solution, as the example can be easily mapped onto such formalism. However, as we already mentioned at the end of section 3.5, cellular automata do not allow to properly describe a model's dynamics, meaning the pathogen life cycle can not be expressed. It is important to remember, in fact, that each pair depicted in Fig. 6.7 is actually the composition of the pathogen hybrid automata with the crop one, which itself is the composition of the Nodes, LAI, and Biomass automata. As such, every single cell in the grid evolves according to its dynamics, and are those specific dynamics that dictate whenever a pathogen can spread or a crop can be infected. Therefore, our main task now becomes to find a formulation that allows us to keep both the temporal dynamics of the systems described so far but also implement some mechanisms inspired by how cellular automata evolve spatially.

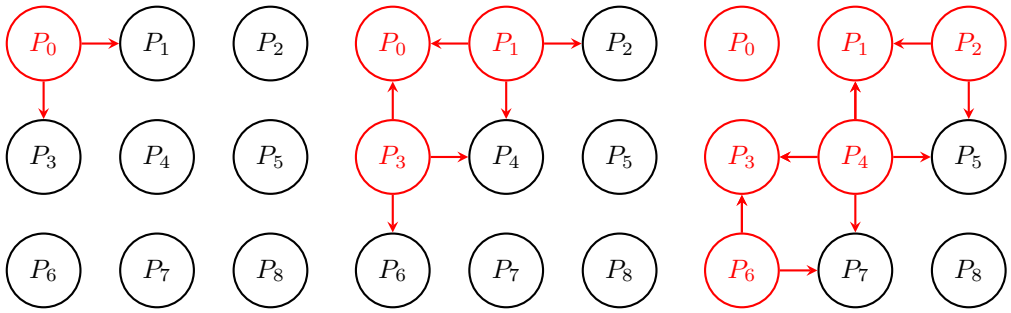


Figure 6.7: Example used to model pathogen spreading across a field

The solution to such a problem can be found by noticing how, in figure 6.7, arrows indicating the epidemic spread can also be seen as events triggered by automata. Therefore, we can model the field with an array of DTHA, each one capable of triggering a specific event indicating the pathogen's contagious status to its neighbors while also being capable of receiving the same event type from those same neighbors. For example, should pair P_0 become infected, it will trigger event I_0 on which its neighbors, pairs P_1 and P_3, will synchronize to preemptively exit their initial state, potentially skipping the condition on the LAI threshold and starting their life cycle, thus emulating the pathogen "jump" from one crop to nearby. Similarly, newly-started pathogens will trigger their own event I_i on which neighbors will synchronize, spreading the infection through the entire cultivar. Figure 6.8 graphically represents how synchronization between automata happens.

The scenario depicted in figure 6.7 has been implemented in the ESCET tool. Nine pairs of crops (made respectively by Nodes, LAI, and biomass) and pathogens were instantiated using the ESCET's template feature, which allowed for easy reuse of the automata's source code. Next, simulation was performed and data plotted, resulting in figures 6.9 and 6.10. The two plots in the first figure show pathogen development over time, showcasing how each crop develops the pathogen at a different time and with different growth depending on the available leaf tissue, with even cases where a crop infected later develops symptoms much faster than another infected earlier;

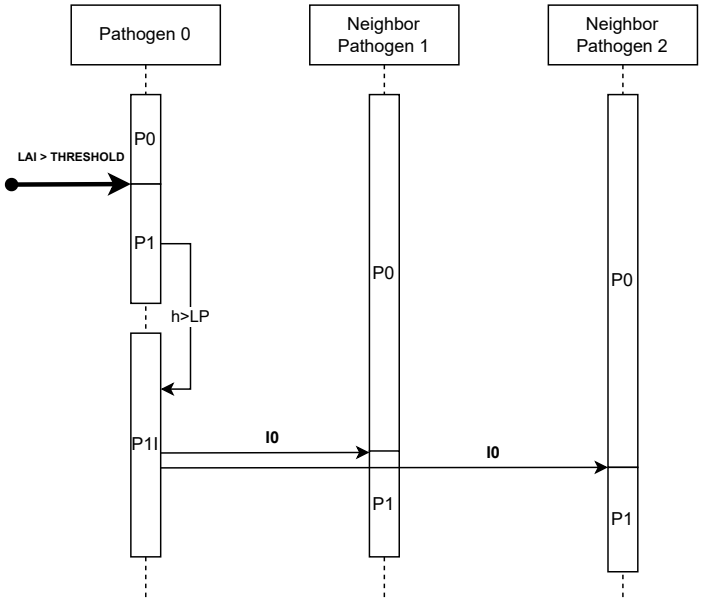


Figure 6.8: Flow diagram explaining how a pathogen automaton can trigger nearby ones by using an event to allow neighbors to exit their initial location preemptively

conversely, the second figure shows how the pathogen spread, starting from the top-left corner, the only one where the infection condition is tied to LAI, to its neighbors. As the figure shows, we successfully managed to add the spatial dimension to our problem, becoming able to simulate not just the life cycle of a single crop-pathogen pair but possibly an entire field.

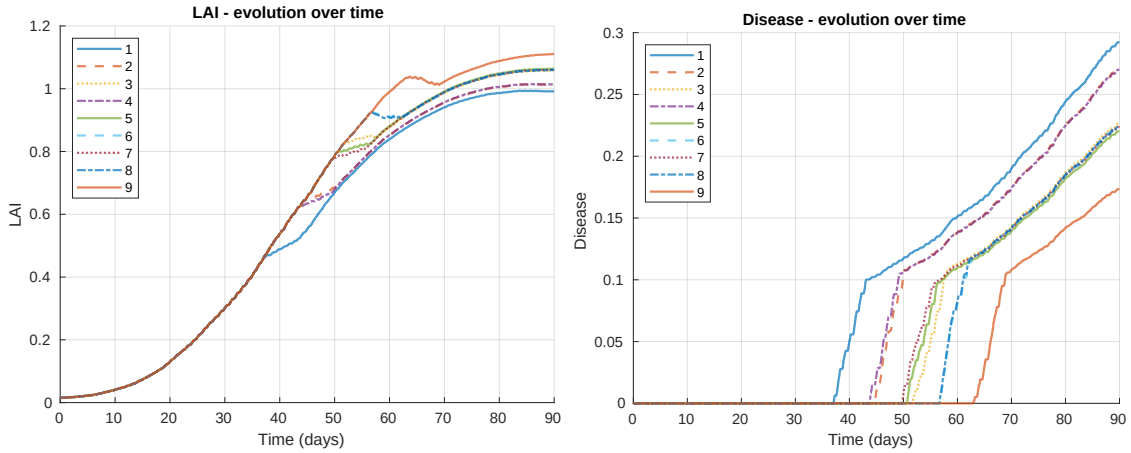


Figure 6.9: LAI development over time for nine infected crops

Following presented experiment, other ones were conducted in order to test the scalability of our solution by increasing the grid size of considered crops. Since such tests were, however, conducted within a different and more complex simulation setup, we remand to Section 8.5 for a full analysis on the scalability of our approach.

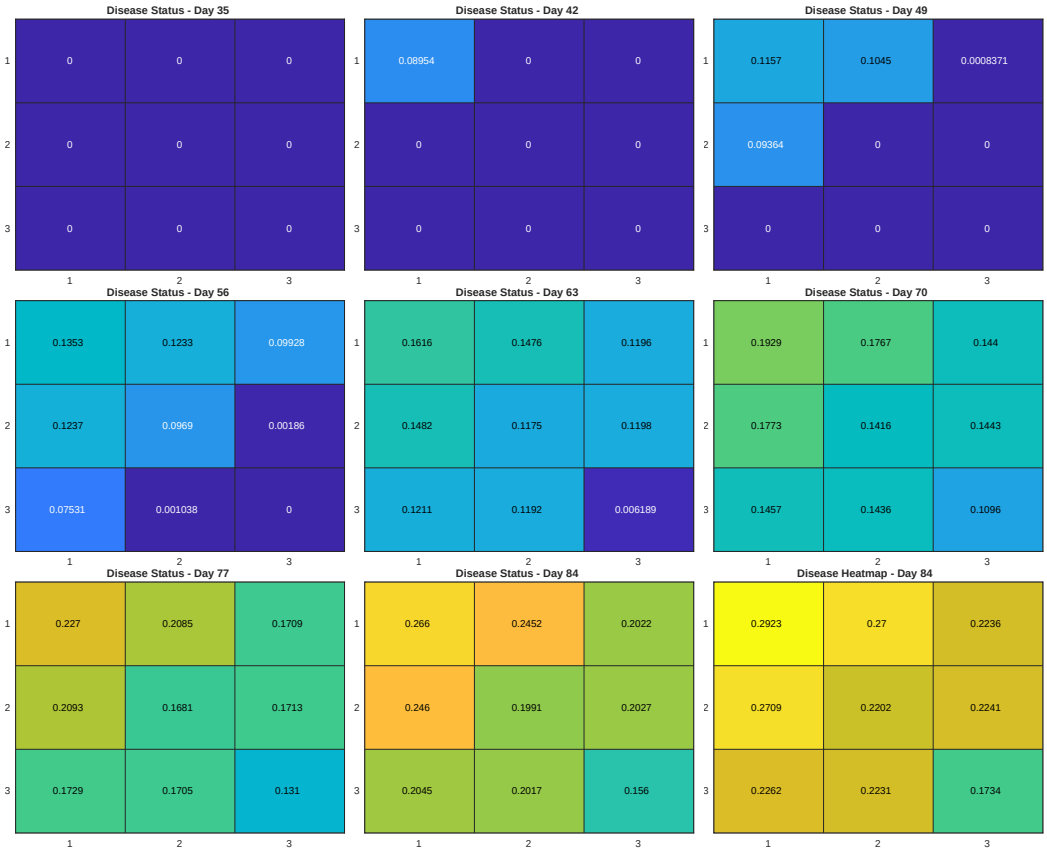


Figure 6.10: Disease spreading across nine crops in the field

Chapter 7

Supervisor Synthesis for Greenhouse Environmental Control

In this chapter, we apply the *Control Supervisory Theory* onto previously obtained models in order to enforce certain requirements onto the systems by automatically developing a supervisor capable of satisfying given needs. The controller synthesis is performed using the ESCET tool, which implements algorithms mentioned in section 2.8.

Due to the nature of their models, both the implemented tomato model and the pathogen can not be controlled directly in any kind, as their evolution is tied only to the passage of time, which is, by definition, not controllable. As a consequence, no possible control actions are available on the models themselves, and therefore, indirect control must be performed by manipulating the models' driving variables. Therefore, we first need to expand our modeled system by adding a simple model of the temperature inside a greenhouse, which instead can be influenced, for example, by the controlling windows' opening/closure to regulate the airflow. Specifically, we decided to focus only on modeling such type of control and not other types for two main reasons: first, it helps us keep the greenhouse model as simple as possible, whereas other forms of control would require a more complex greenhouse model to be developed, and this is not inside the scope of this work; secondly, it is one of the first type of control used in most greenhouses regardless of their technological level, meaning that found solution could be applied almost in any context.

Since modeling greenhouse systems is a complex task, we also make the following assumptions to ease the modeling process:

- The system sampling time is much greater than the controller processing speed. This ensures that the supervisor will always be able to react to system changes and output corresponding commands before any other change in the system occurs.
- The system sampling time is big enough to ignore small fluctuations in temperature. This allows us to assume that the temperature inside the greenhouse will be at least greater or equal to the outside.

We believe these assumptions to be plausible: traditionally, sampling time for sensors within greenhouses works at best in the order of seconds; on the contrary, the CPU processing speed of supervisors is in the order of microseconds. Additionally, fluctuations in external temperature usually do not last for long and, therefore, can be ignored with a sufficient long sampling time. Obviously, the obtained model will be far from an actual complete description of how temperature evolves inside a greenhouse, but we simply need it to faithfully describe its variations and how it responds to control actions so that we can use its outputs as inputs for crop models, allowing to produce trustworthy outputs.

Following the completion of the greenhouse model, we proceed to generate an abstract representation of our entire system by using the Finite-State Automata formalism, which does not possess any kind of dynamic, and tag it with requirements we want to enforce. The reason we are required switching to regular FSA instead of keep using DTHA is due ESCET's synthesis algorithm for supervisor and control, which only works with this specific kind of automata, meaning that any other kind, even if openly supported by the tool, can not be used in the synthesis algorithm. Following the conversion, we synthesize the supervisor and put it in parallel to our models, to simulate the system and assess its effectiveness.

7.1 Simple Greenhouse model

A simple model to estimate greenhouse conditions has been developed in order to estimate the greenhouse's inside temperature as a function of the external one and solar radiation. Moreover, this model also takes as inputs the state of windows' opening, which regulates the thermal difference between the inside and outside and is directly under our control, thus giving us the possibility to change it and influence temperature's evolution.

The model's dynamic computing greenhouse temperature is presented in equation (7.1), obtained by taking inspiration from the capacitor dynamics in RC circuits.

$$T'_{\text{in}}(t) = \tau \text{Rad}(t) S - \frac{(T_{\text{in}}(t) - T_{\text{out}}(t))S}{r + r_k \cos(\theta)} \tag{7.1}$$

T_{in} and T_{out} represent the temperature inside and outside the greenhouse, Rad is the solar radiation, and θ is the window's opening angle. Finally, S represents the greenhouse surface, r and r_c are the thermal resistance, either fixed and in the case of open windows, and τ is a reduction factor for the incoming radiation.

Obviously, the described model does not integrate very well with previous ones, as this one is described by a pure continuous dynamic, whereas other ones followed a discrete-time formulation. In addition, this formulation needs continuous inputs in the form of T_{out} and Rad, which, however, are sampled with a fixed, discrete time step. Therefore, we discretized equation (7.1) by solving it first and implementing the obtained solution, shown in equation (7.2), allowing us to estimate inside temperature at a given time step h as a function of external temperature and radiation at the same time.

$$T_{\text{in}}(k) = \tau \text{Rad}(k)(r + r_k \cos(\theta)) + T_{\text{out}}(k) + c \cdot e^{\frac{-kS}{r + r_k \cos(\theta)}} \tag{7.2}$$

After obtaining the presented equation, we finalize our model by wrapping it into the DTHA shown in figure 7.1 (stutter transitions are omitted) to be used later for simulation. In the figure, T represents the automaton's time step, which is shared with all other automata and set to be one hour.

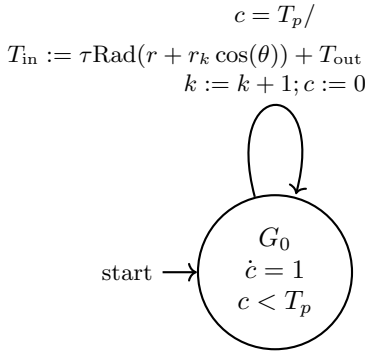


Figure 7.1: Automaton for Greenhouse temperature evolution

The described greenhouse model alone can not be used to develop the controller since, in order to complete the entire system description, we must add information about the opening and closure of the windows and shades to explicate how such operations change the θ parameter. Therefore, we completed the greenhouse system description by adding automata shown in figure 7.2. Initially, the windows are still and do not move, waiting for a command to be issued. As soon as one of the two possible commands, either OPEN or CLOSE, arrives, the model changes in the corresponding location, resetting the clock in the process. Here the opening angle changes either by increasing or decreasing of three degrees at every discrete time-step, defined by the time constant T_c set to be at 5 seconds. The model keeps increasing/decreasing the angle, bound to never grow higher than 90° and lower than 0°, until a STOP command is issued, which forces the model to return to the HALT location, where the opening angle remains constant until a new command is received.

7.2 System Abstract Description

In order to operate the supervisor synthesis within the ESCET framework, we need first to abstract the system we aim to control so that it is only composed by FSA. In our case, we decided that

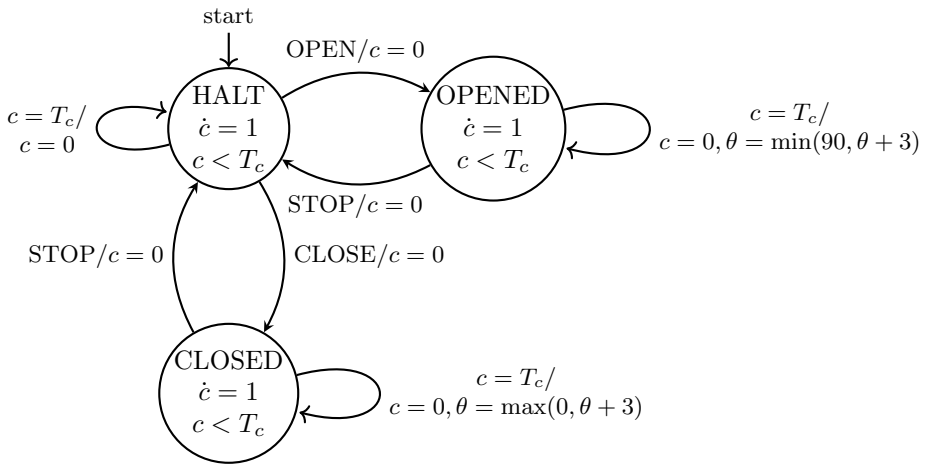


Figure 7.2: Automaton describing windows opening and closure.

our system is composed of the pair crop-greenhouse, leaving the pathogen aside for the moment. Therefore, our first objective is to find a way to rewrite such models using FSA formalism in a way that allows them to interact and synchronize using nothing but shared events while also giving us the freedom to express properties we want to enforce on it that can actually be synthesized without resulting into an empty controller. Figure 7.3 gives a visual representation of the system we aim to abstract.

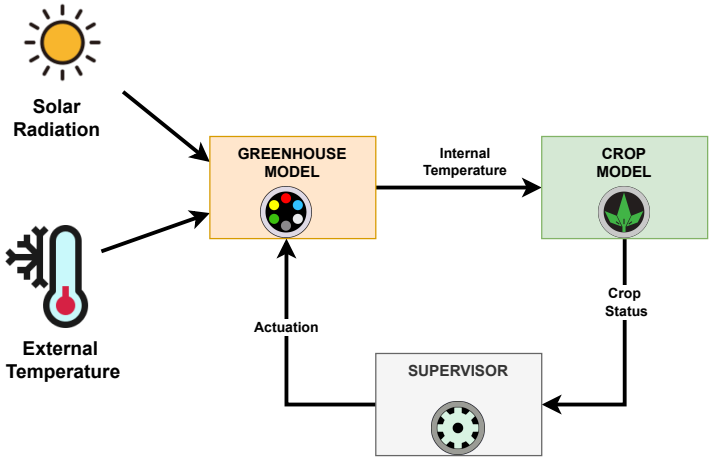


Figure 7.3: Control loop for the system we want to abstract.

The first step in abstracting our system is to find a correct representation of the crops. However, abstracting automata shown in Chapter 5 would not be enough, as those descriptions heavily rely on values from internal variables to change state, greatly limiting possible interactions with other automata. As such, we decide to slightly modify and expand automata for Nodes and Biomass in order to better express the relations the models have with the temperature. Expanded automata can be seen in figures 7.4 and 7.5: for nodes we now fully express the various cases that compose equation 5.16; meanwhile Biomass automata now openly depicts cases related to the critical temperature T_{crit} appearing in equation 5.8. Importantly, we want to underline how the formulation presented here are completely equivalent to ones used in Chapter 5, as underlying equations remains the same-the only difference being that in Chapter 5 we kept cases for equation implicit when designing the automata. In figure 7.4, the four parameters T_0, T_1, T_2, T_3, T_4 are the same used to define the intervals in equation 5.16, here generalized and expressed as parameters. Notably, locations N_3 and N_4 do not update the state variable, as they are the cases where temperature is too low or too high and therefore nodes' growth is null. Meanwhile, in figure 7.5, updates follows rules depicted in table 5.2, with the only note being that in locations $B1_c$ and $B2_c$ we applied the case $T > T_{\text{crit}}$ of equation 5.8 (marked as updates u^d_{2c} and u^d_{3c}), while in location $B1$ and $B2$ the other case is applied.

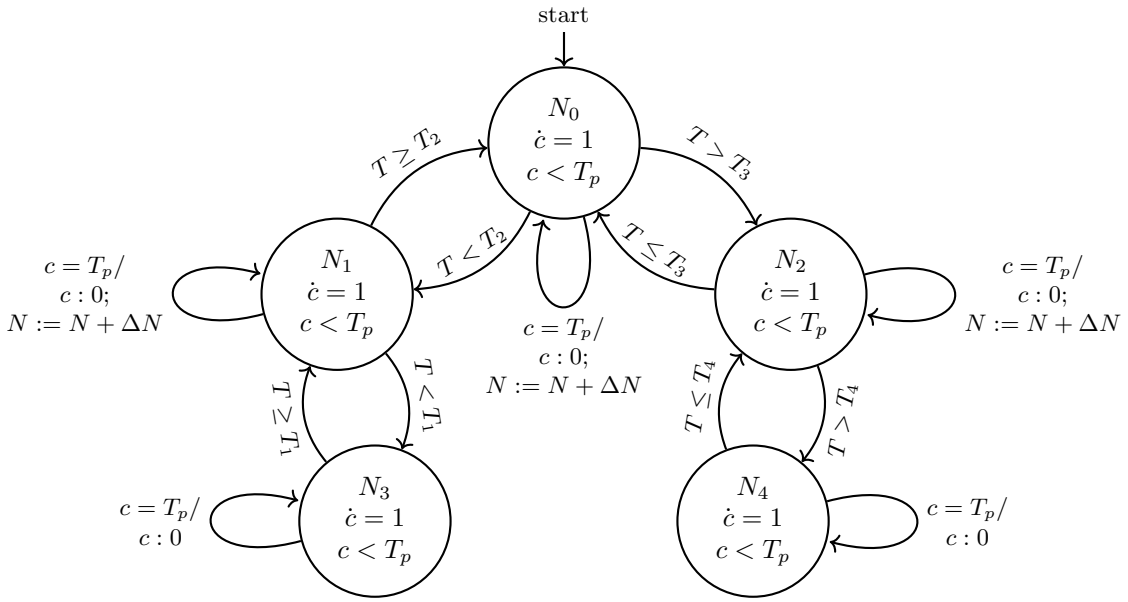


Figure 7.4: Expanded version of the Nodes automaton.

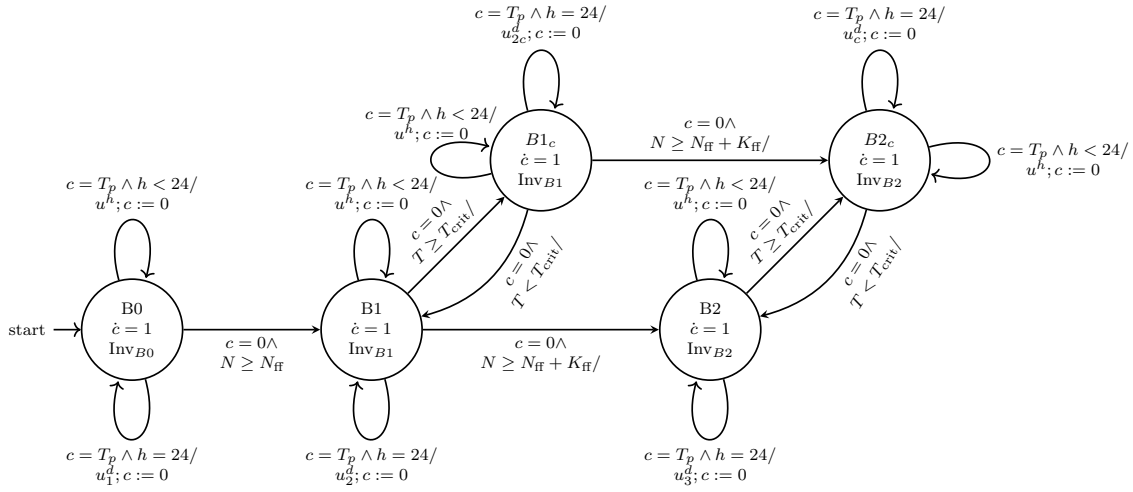


Figure 7.5: Expanded version of the Biomass automaton

Following such redesign, we abstract those designs and turn them into FSA to use them within ESCET's synthesis process. In our case such process is trivial: since our interest is focused only on which state the two automata are in, our abstraction process can be performed by simply removing all dynamics and self loops from each location, and turning state-switching edges into ones bound to specific event triggered when the original switching condition is satisfied. Therefore, after applying such process we obtain FSA shown in figure 7.6. The abstract version for the Nodes' automaton behaves the same as its hybrid counterpart minus the updates del loops: the crop begins its growth in the OPTIMAL state and changes it whenever an event signals a change in the temperature: in state OPTIMAL, the crop has the best conditions for nodes' developments, which subsequently positively affects both LAI and Biomass; in states COLD and HEAT the temperature is out of the optimal range, but still enough for the crop to keep growing; finally, in states COLD_STRESS and HEAT_STRESS temperature goes completely outside allowed range, hindering crop's growth and potentially damaging it. Meanwhile, the Biomass automaton starts in state GROWING, where the crop is still young, and change states following the event FRUIT_ON, symbolizing the crop's age passage to adulthood, represented as state FRUITS. Here our attention shifts on controlling whether the sensed temperature is above or below a critical threshold. In the former case crops' fruit will grow without problem, while in the latter fruit's growth will be hindered. The passage between state FRUITS and CRIT is regulated by events CRIT_UP and CRIT_DOWN. Notably, it is possible to see that the structure of this last automaton is slightly different from its DTHA counterpart, as states $B3$ and $B3_c$ are absent. The reason of such absence comes from the consideration that this abstraction only focus on indicating whether environmental conditions are hostile to crops; since thresholds for such triggers are the same across both states $B2$ and $B3$, we decided to merge states $B2$ with $B3$ as well as $B2_c$ with $B3_c$.

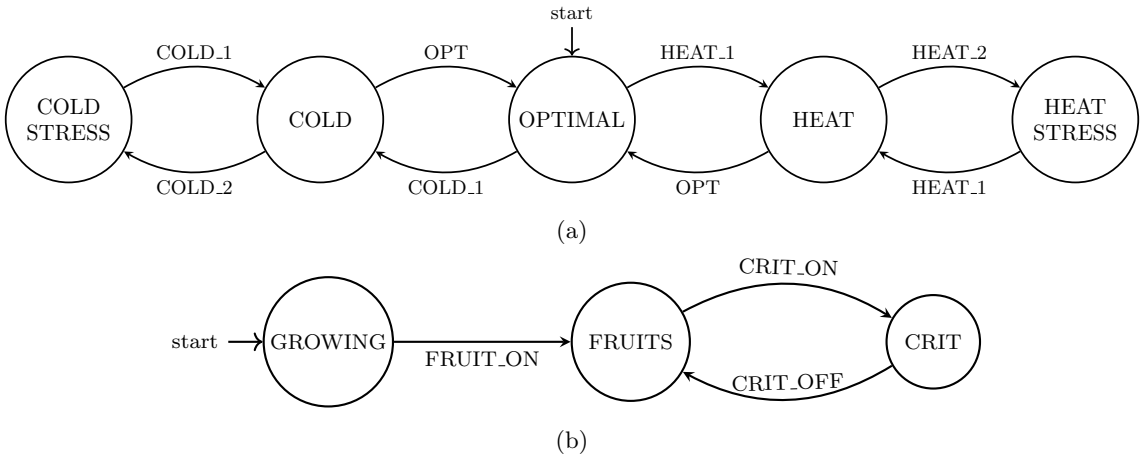


Figure 7.6: FSA obtained after abstracting Nodes (a) and Biomass(b) automata.

All events depicted in figure 7.6 (OPT, HEAT_1, HEAT_2, COLD_1, COLD_2, FRUIT_ON, CRIT_ON, CRIT_OFF) are all assumed to be generated by sensors detecting when given thresholds are crossed, and are therefore considered *uncontrollable*. Finally, to give our system more reactivity, we coupled such a sensor with another one detecting whether the temperature is increasing or decreasing, triggering events RISING or FALLING as consequence.

Next, we must add to our abstract description of the system the windows' opening and closure model. Such process is immediate, since we simply need to remove variables and updated from the automaton to obtain its abstract version. This process is trivial, and therefore resulting automaton is not depicted.

Finally, the last automaton we must implement serves to represent the system's reactivity to events. This automaton is based directly on the one shown in Diekmann's work [120] and is depicted here in figure 7.7. State F_0 represents the system in its processing phase, where we can issue commands and trigger *controllable* events at will, which, in our case, happen to be only the commands regulating windows. After all control actions needed have been issued, event τ brings the automaton from the processing state to the listening one, where we wait for any event σ to trigger, with Σ_G defined as the union of all events set of automata described in this section minus τ.

$$\Sigma_G = \{\text{COLD\_2}, \text{COLD\_1}, \text{OPT}, \text{HEAT\_1}, \text{HEAT\_2}, \text{FRUIT\_ON}, \text{CRIT\_ON}, \text{CRIT\_OFF}$$
$$\text{OPEN}, \text{CLOSE}, \text{STOP}, \text{RISING}, \text{FALLING}\}$$

This particular model allows us to express the first assumption we made, which states that our controller processing time is smaller than our system time. This means that every time an event triggers, our controller has enough time to capture it, process it, and respond accordingly with an appropriate control action before any other event is generated by the system.

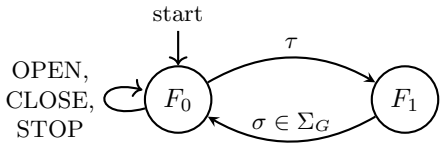


Figure 7.7: Automaton modeling system reactivity to other events.

7.3 Supervisor synthesis

Automata presented in the previous section have been put together to obtain an abstract model of the entire crop-greenhouse system capable of going through the control supervisory synthesis in the ESCET framework. Therefore, the next step is to express some requirements we want to enforce in order to apply CST algorithms for the supervisor synthesis subsequently.

Despite the simplicity of the described system, specification of requirements is not an easy task. Since, in fact, the system is moved almost exclusively by uncontrollable events related to the temperature, which we can only affect indirectly via windows control, requirements we can enforce are minimal, causing, in most cases, the synthesis to fail and produce an empty supervisor. For example, preventing temperature from exceeding a certain threshold will always fail to produce a supervisor, as it is impossible to prevent the external one from growing above given thresholds, and the modeled greenhouse does not possess a cooling system. Therefore, we limited ourselves to specifications that restrict possible combinations of states to only safe ones for the system. This, obviously, poses a limitation in our approach, but we still believe our methodology to be correct, as such limitation is not caused by the method itself but rather by the system we modeled and the decisions we made, preferring to keep the model as simple as possible and, therefore, limiting our control actions. Certainly, future works may address and solve such limitation, by employing different abstraction techniques or tools.

Traditionally, system specifications that are used in the supervisor synthesis are expressed using FSAs, which are composed with the system to compute the controller automaton. Since we are using the ESCET framework instead, we can skip this passage as the tool allows us to express such requirements as boolean predicates that are internally interpreted and used by the synthesis algorithm. Therefore, we simply had to formulate wished requirements using a common language and then convert them into boolean expressions understandable by the software to proceed with the synthesis. In total, two requirements were established and formulated in common language, these being:

1. If temperature rises, windows should start opening.
2. If temperature falls, windows should start closing.
3. If one of the two crop automata is in a critical state (either CRIT or HEAT_STRESS), windows must open.
4. If the Nodes' automata is in state STRESS_COLD, windows must close.
5. If the temperature is within the optimal range, windows should stop moving.

Stated requirements have been converted into logical expressions compatible with the ESCET framework, which treats states' and events' names as boolean functions expressing whether the corresponding automaton is in such state/the event is enabled (*true*) or not (*false*). Furthermore, ESCET introduces the "*needs*" keyword (here represented as $\rightarrow$), which specifies that an event or state can be true only if the boolean expression which follows is also true. Following such statements, the two requirements were reformulated as follows:

1. OPEN $\rightarrow$ (HEAT_STRESS $\vee$ RISING $\vee$ FRUIT_CRIT)
2. CLOSE $\rightarrow$ (COLD_STRESS $\vee$ FALLING)

3. STOP → OPTIMAL

Obtained statements were incorporated alongside system description in a single `.cif` file and submitted to ESCET's synthesis algorithm. Obtained supervisor consisted in a Finite-State Automaton consisting of 360 states and 1321 transitions, although it is possible not all of them are actually used since, due to the abstraction process, all events are possible at the same time, whereas in reality only some of them may be possible at a particular time. For example, only one temperature-related event may be enabled at a given time, whereas such limitation is absent in the abstract model, and all events are enabled at each step.

The obtained supervisor has been integrated into the actual system by composing it alongside the greenhouse model described earlier and the crop model. Integrating the supervisor is a

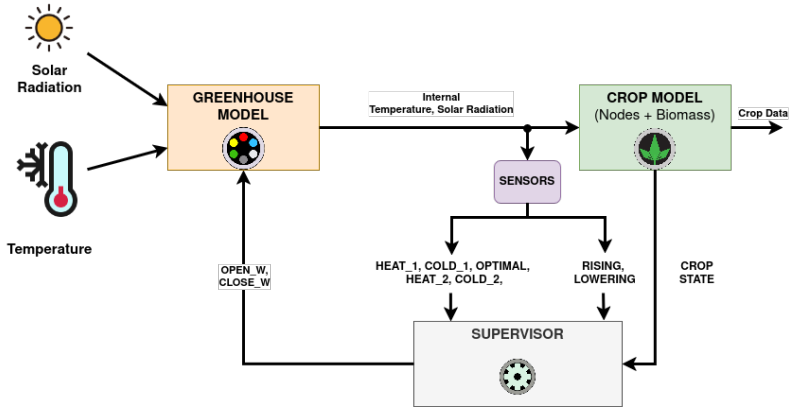


Figure 7.8: Simulated systems made of the greenhouse model, the crop model, the supervisor, and sensors generating events.

straightforward operation, even if it refers to an abstract version of our system since all events used are still present in the actual system thanks to sensors triggering them. In the end, the fully controlled system resulted in the setup shown in figure 7.8.

The obtained system was simulated in the ESCET framework, and data related to internal temperature, solar radiation, crop nodes, LAI, and biomass were extracted and plotted in MATLAB. Plots from figure 7.9 to 7.11 show simulation results compared against cases where the greenhouse control is absent and the windows are always fully opened or closed. The results show that the control system manages successfully to keep the temperature within the best range possible at all times, how the figure 7.12 shows. In particular, we can notice from figure 7.12 that implemented supervisor appears to implement a proportional rule to operate the windows, by increasing or decreasing the angle accordingly to the temperature variations. Additionally, figures 7.13 and 7.14 showcase how states for the abstract versions of Nodes and Biomass automata (and consequently their hybrid counterpart) respond to chages in temperature. Notably, we can see how, since the threshold for critical temperature of biomass is lower than the one causing stress on nodes, we never enter the state STRESS_HEAT, since the controller act to reduce temperature as soon as we enter state FRUIT_CRIT. Consequently, we can see how biomass' automaton changes states more frequently than nodes one, exactly in virtue of having lower thresholds. Conversely, Node automaton changes state less frequently, usually only during sunset and sunrise phases.

While at first, obtained control policy may appear too simplistic, we must consider that the obtained control strategy is very close to how actual greenhouse controllers are implemented nowadays. In fact, following discussions with experts in the greenhouse control sector, we discovered that proportional control is one (if not the most) of the used control strategies related to controlling windows in the greenhouse. Therefore, we successfully implemented a controller that could actually be considered for use within a greenhouse, proving our methodology can produce actual results.

Still, we must also note that there are many other paths to explore regarding the presented subject. First, we must address the fact that the obtained results came from a manually abstracted model for which regular supervisor synthesis was applied. Obviously, checking whether such a problem can be solved differently by using tools working directly on HA or performing a more rigorous synthesis process is a task we aim to perform to check if better results could be achieved. Secondly, introducing other elements from the greenhouse context, such as heating or humidity control, would lead to a more complicate model to study, but also provide us with more interesting scenarios for control strategies.

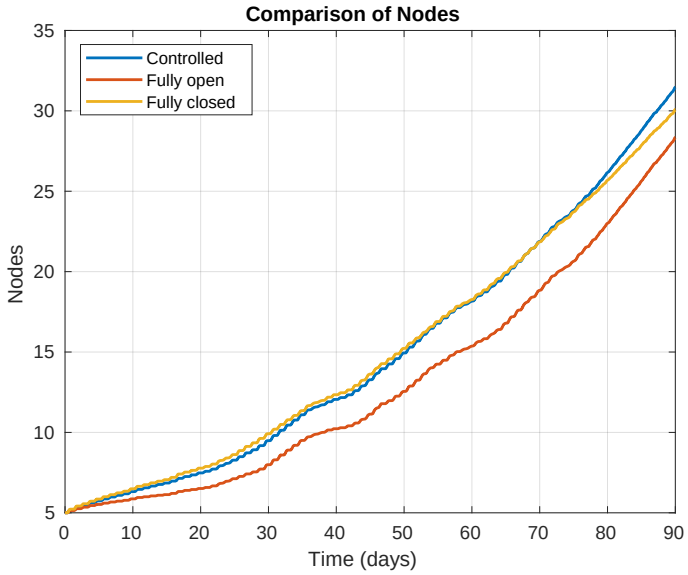


Figure 7.9: Simulated internal nodes' growth for a fully closed, fully opened, and supervised greenhouse

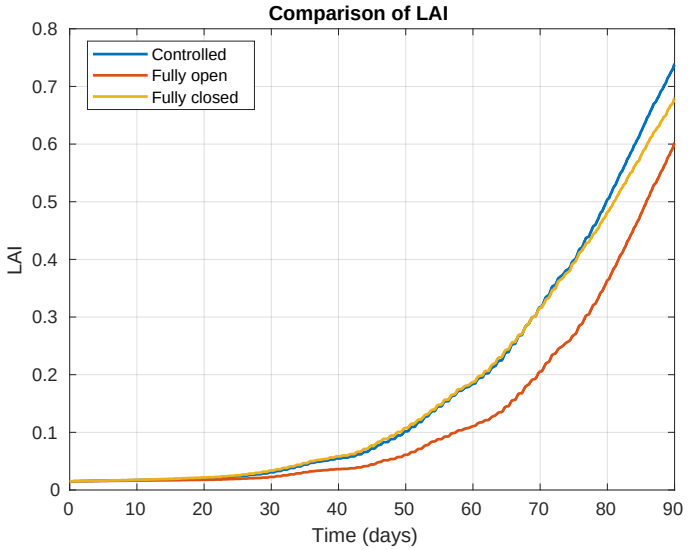


Figure 7.10: Simulated internal LAI for a fully closed, fully opened, and supervised greenhouse

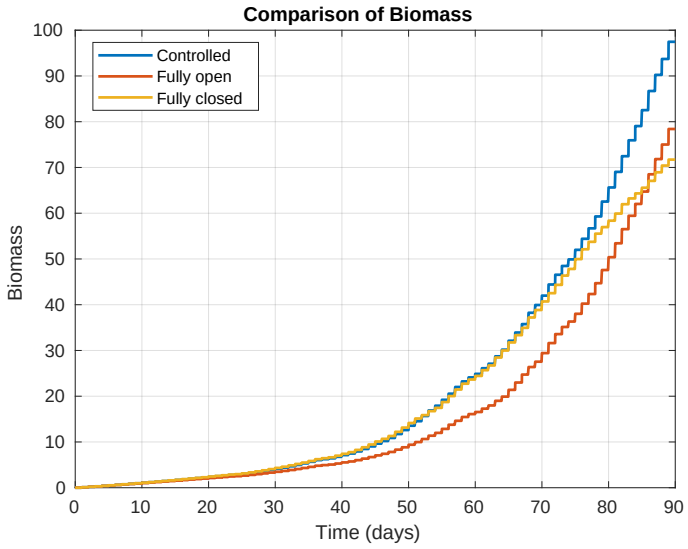


Figure 7.11: Simulated internal total biomass for a fully closed, fully opened, and supervised greenhouse

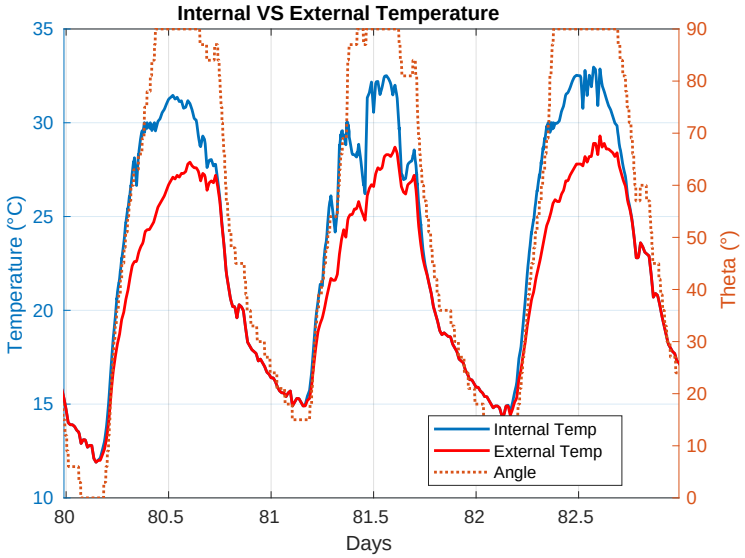


Figure 7.12: Extract of internal and external temperature evolution under implemented supervisor. Plots shows also the opening angle

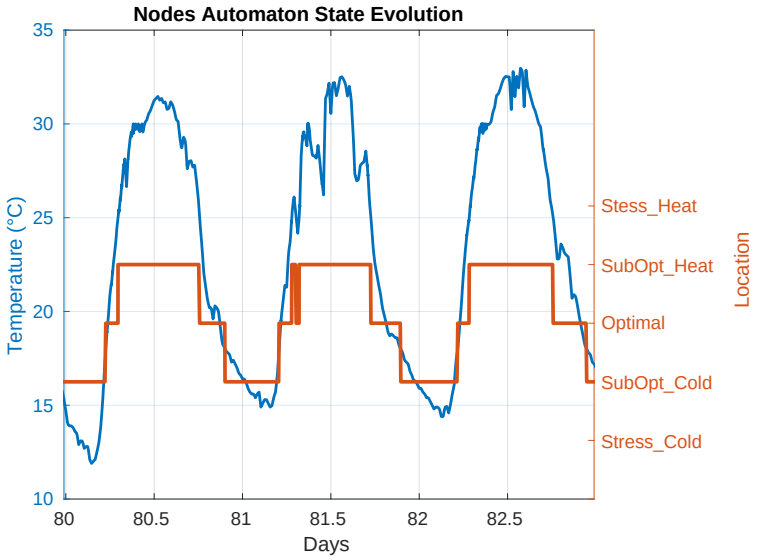


Figure 7.13: Temperature against Nodes' automaton states

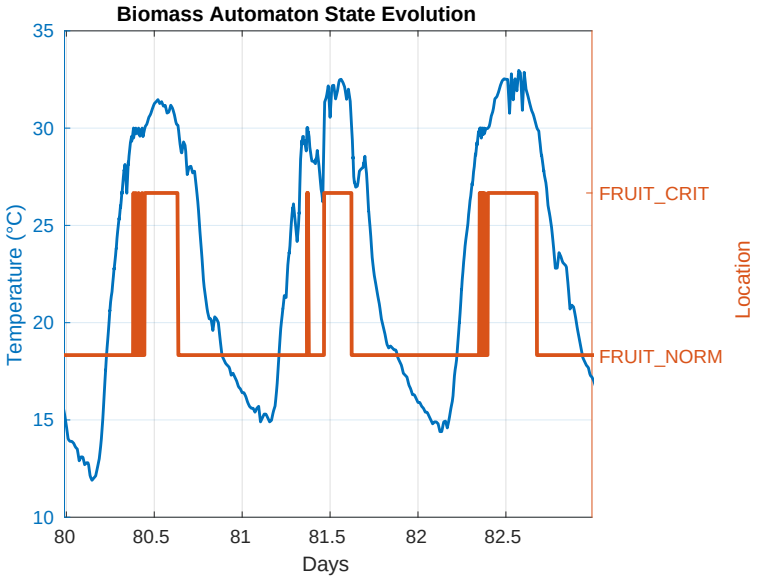


Figure 7.14: Temperature against Biomass' automaton states

Chapter 8

Parameters Assessment for Existing Control Strategies

In the previous chapter, we shown an attempt of automatically obtain a control procedure to find the best strategy to regulate windows opening. In this chapter, instead, we move our focus from developing our own control strategy to improving an already existing one by using our model to perform simulations and search for the best parameters set to improve control results, either by finding the best one for a given time period or the one who generally produces best results. Furthermore, we also show how such methodology can be used also to find parameters at runtime in an MPC approach.

8.1 Actual Controller Modeling

In order to optimize a control strategy, we need first to model the actual controller as a DTHA so that we can compose it with the system developed so far. In particular, we focused on implementing a P-type controller, following results obtained in previous chapter and experts' advices which informed us that such strategy is the used in most commercial products. However, we decide not to use previously obtained supervisor as basis for our tests, but rather implement the controller itself as a DTHA, to simplify its description and have more control on its behavior and parameters.

The proportional (P) controller possesses a simple behavior, represented in figure 8.1. The

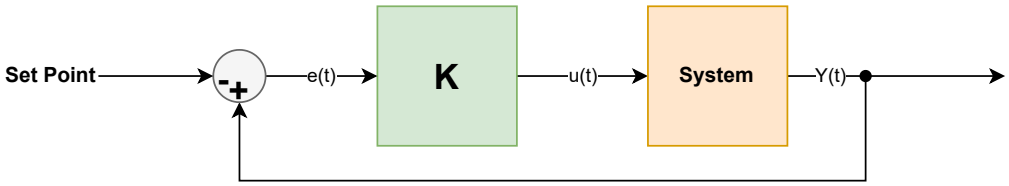


Figure 8.1: Typical structure of a P controller

measured value is compared with a given set point, and the error is multiplied by a gain to obtain the corresponding control action, which is then applied by the controller, producing a new measure to close the loop. However, in our case, the controller works by using two temperature points rather than one, known as *Full-Close Temperature* (FCT) and *Full-Open Temperature* (FOT), that represent the temperatures at which greenhouse's windows must be entirely closed or opened respectively. Subsequently, the windows' opening angle is adjusted according to how internal temperature poses between the two points. For example, if we pose FCT and FOT respectively to be 20 and 30°C, we will have that each single degree Celsius over the FCT will correspond to an opening of about 9°. Therefore, we can express the controller's output θ by simply using equation (8.1), where T represents the greenhouse's internal temperature.

$$\theta(t) = \max\left(\min\left(K \cdot (T - \mathrm{FCT}), 0\right), 90\right) \tag{8.1}$$

$$K = \frac{90}{\mathrm{FOT} - \mathrm{FCT}} \tag{8.2}$$

In the equation, the *max* and *min* terms serve only to restrict possible results within our admissible range of $[0, 90]$ degrees.

From the given equation, we can easily construct the corresponding DTHA, shown in figure 8.2, constituted by a single state computing the angle at each time step by using the previous equation. However, a big change we must underline in the presented automaton is the time-step it uses, here depicted as T_c, to differentiate it from previously used one T. According to experts who suggested the strategy, in fact, the controller must perform in the order of seconds, and therefore, the automaton must run at a much lower time step than all previously implemented automata. As

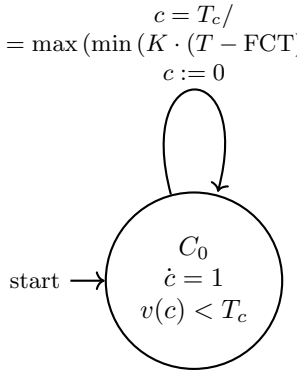


Figure 8.2: Automaton implementing the windows' controller. Stutter transition is omitted.

a consequence, we decided to set the modeled controller's time step T_c to be five seconds.

8.2 Parameter Assessment for the Control Strategy

An interesting point to note is that the proportional component of the modeled controller is heavily dependent on the FOT and FCT parameters, whose values are, however, left to be decided by end users based on their own knowledge and experience of the greenhouse and the crops growing inside. While this solution may work and produce good results, it is clearly improvable using more scientific methods, capable of providing end users with more information and suggestions on which value to assign for parameters based on data analysis and simulation that can subsequently be used to boost user's knowledge and therefore enhance the overall quality of the decision-making process. As such, we set up an experiment to prove how, using our methodology, we can find the best combination for parameters FCT and FOT across multiple seasons and time periods.

The experiment was conducted by using the climatic dataset already presented in section 4.3, made of data sampled every 15 minutes for about a year and a half, from August 26, 2021, to January 19, 2023. From this dataset, 17 different input traces were obtained, each one covering 90 days and with a temporal shift between a trace and the next one of 30 days. Furthermore, data were augmented using interpolation to reduce the time step from the above-mentioned 15 minutes down to the 5-second step used in the control loop. Obviously, this operation does not help us to reconstruct actual data in such time frame, but we believe the approximation provided to us by interpolation to be sufficiently precise, especially considering missed fluctuations would be ignored in any case due to the larger time step of the original model. Next, a model of the entire system was created by putting together the previous chapter's greenhouse model, Chapter 5's tomato model, and this chapter's control model using ESCET's tools. Again, we left out the pathogen since we prefer to focus only on improving crops' growth.

The composed model was simulated within the ESCET framework using obtained traces as input. However, it turned out that even a single trace's size was too much for the tool to handle, causing memory problems and aborting the simulation. Therefore, we could not proceed with using ESCET to complete our task but were forced to find another way to implement and test the model. Luckily, ESCET itself provided us with a solution by implementing code generation algorithms capable of synthesizing executable code from models in various programming languages. Among available possibilities, we decided to set on the C99 language, which is better suited for memory control and fast computation, two elements much needed for massive simulation with significant input traces. As such, we ended up with a C99 implementation of the composed system with an already delineated entry point for traces' inputs and parameter values, meaning we only had to write down the main code responsible for looping over traces and collecting results. Importantly, we want to underline how we did not implement any part of our model since the synthesis procedure constructed the source code correctly *by construction*, thus relieving us from the task of

demonstrating the equivalence between the implemented and formal model. Figure 8.3 graphically shows the procedure described so far.

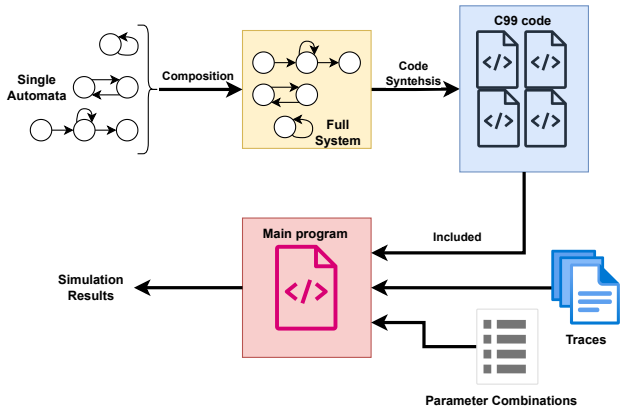


Figure 8.3: Steps to integrate the formal model into an executable source code for faster simulations

Working now in the C99 standard, an algorithm to search for the best parameter combination was developed. Since the set of values assignable to each FOT and FCT are limited only to actually meaningful values, we decided to opt for a simple brute-force algorithm trying all possible parameters' combinations onto a trace to find the best result before moving to the next one and restarting the process. More complex search techniques could also be applied, but since searched values are most likely to be in the integer domain due to the limited precision of sensors responsible for reading them, we believe that the brute force approach is sufficient. Parameter FOT was given a possible range of $[25 \dots 32]$, while for FCT, the set range was $[15 \dots 20]$, and each possible combination was numbered for a total of 48 possibilities.

Upon simulating each combination, the result for crop nodes, LAI, and the three biomass' outputs were put together in a linear combination, resulting in the total simulated outcome. All

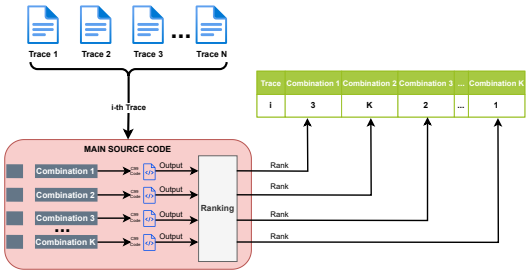


Figure 8.4: Schema depicting how ranking for different combinations is performed

outcomes for a single trace were subsequently compared, and each combination was given a rank equal to its position in the ordered list of outcomes: the combination with the highest outcome obtains rank one, the second highest a rank of two, and so on. Figure 8.4 provides a general schema on how the developed code operates to test combinations for a single input trace i.

All 48 combinations were tested against all 17 test traces, and all outcomes were gathered and analyzed. The results can be seen in Figure 8.5, which shows a color map where the lowest ranks (better results) are colored in yellow, whereas the higher ranks are in blue. Even without further analysis, we can already retrieve useful information from the presented figure. For example, we can clearly see how the *optimal* strategy does not exist, since no combination consistently holds the best ranks across all traces, with certain parameters' combinations even switching from beneficial to negative in different situations. Moreover, we can also see a trend in how ranks are distributed across traces, such as in traces 1-7, where ranks tend to decrease as the FOT parameter grows, which happens exactly every six combinations, i.e., every time all possible FCT combinations have been tested. Meanwhile, this trend becomes inverted from traces 9-14, where combinations with lower values of FOT and higher values for FCT produce good results. In addition to information retrieved from figure 8.5, we also performed some statistical analysis to gather more information, presented here in figures 8.6a and 8.6b. The first figure helps us find the best overall combinations

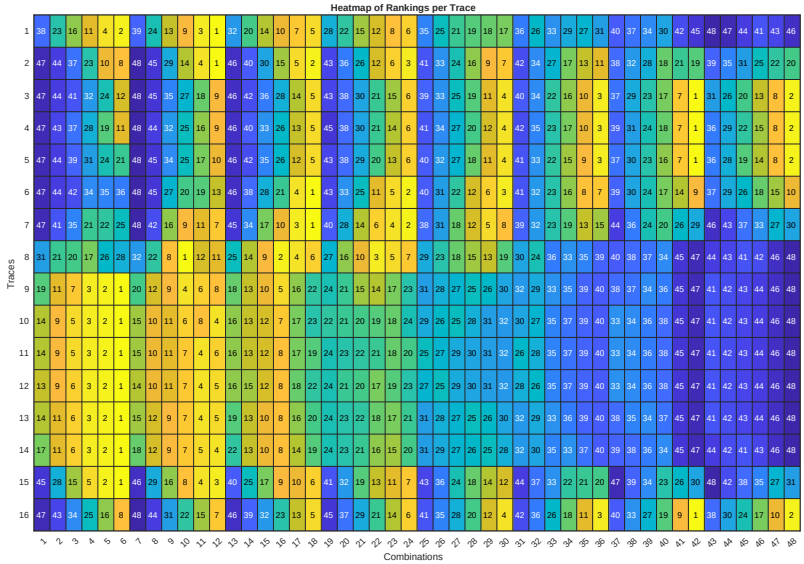
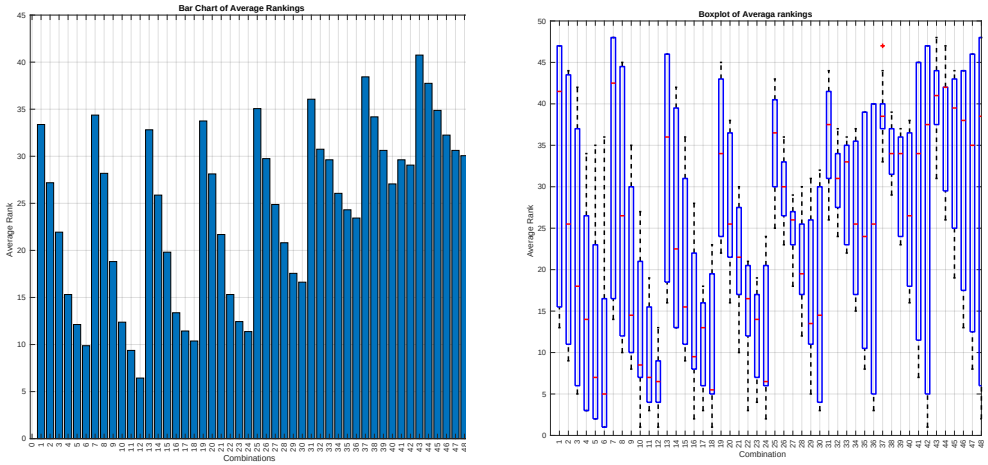


Figure 8.5: Colormap showing rankings for each parameter combination with respect to simulation traces

by plotting mean ranks, meaning lower values indicate a better average rank; similarly, the second figure also plots mean ranks but with the addition of their variance, in order to properly assess how much a combination's score varies over simulations. From the presented plot, we can easily



(a) (b)

Figure 8.6: Average rank (a) and rank + variance (b) per combination

see how combination twelve (FOT 26, FCT 20) scores low ranks consistently, resulting in a low average and low variance, while the second-lowest average rank is held by combination six (FOT 25, FCT 20), which however possesses much higher variance, and therefore must be considered less consistent in producing good results. Moreover, the presented figures also help us to delineate the trend already noticed in figure 8.5 and understand it better. We can, in fact, notice not one but actually two different trends in how ranks change in response to parameters' values, one for each parameter. Starting from combination 1, we can notice how the average rank decreases consistently until reaching combination 7, where the rank immediately ramps up. From here, however, ranks return to decrease until combination 14, where it rises again. This trend repeats itself across the entire plot, and if we tie it to how combinations are computed, i.e., trying all possible values for parameter FCT with a fixed FOT before incrementing it, it allows us to deduce that higher values of FCT positively affect production. On the contrary, the second trend we can identify is a general increment in average rank across all combinations, which ties with the constant increment of the FOT parameter, meaning that using higher values negatively affects overall production.

Concrete results for simulations can be seen in figure 8.7, showcasing for each of TOMGRO's state variables the two best and worst simulation outcomes for the trace that achieves the highest

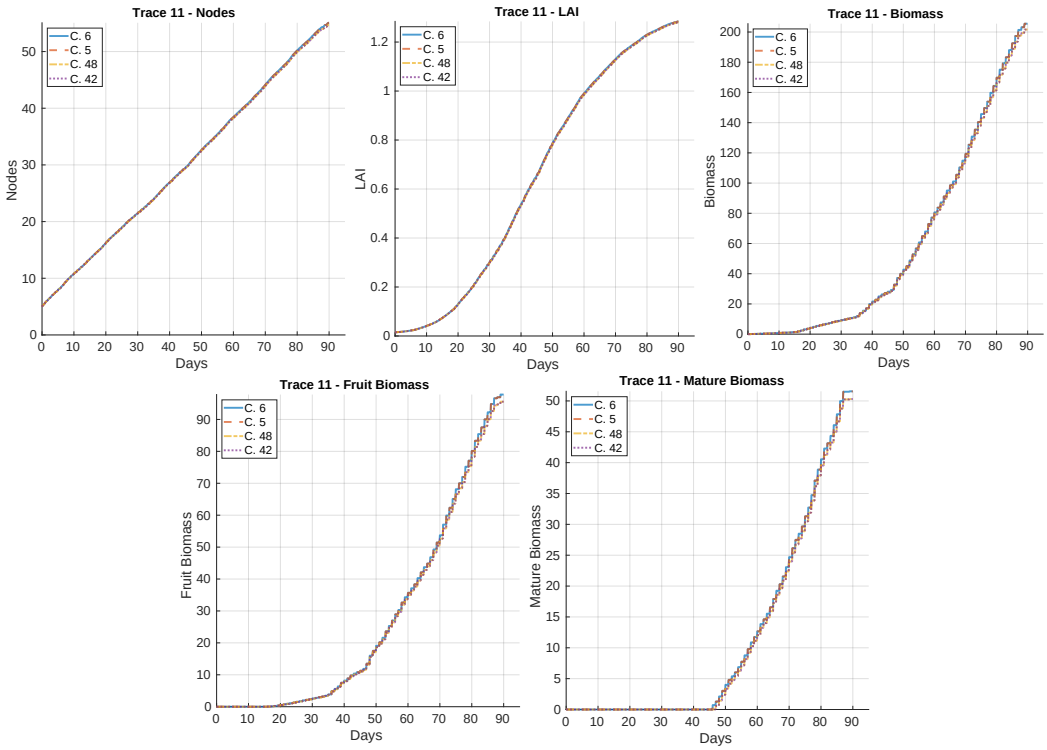


Figure 8.7: Simulation results for highest-yield trace: top two and bottom two strategies

yield. From plots, we can see that while all strategies reach approximately the same outcome for nodes and LAI, results for the three produced biomass clearly show a difference between a good set of values for parameters FOT and FCT against a bad one, resulting in a 1.2%, 1.6% and 1.7% percentual loss in total, fruit, and mature biomass. Such difference becomes even higher if we decide to change the considered input trace, reaching up to a 2% point difference in both fruit and mature biomass between best and worst possible combination (figure 8.8) form only managing how windows' opening policy.

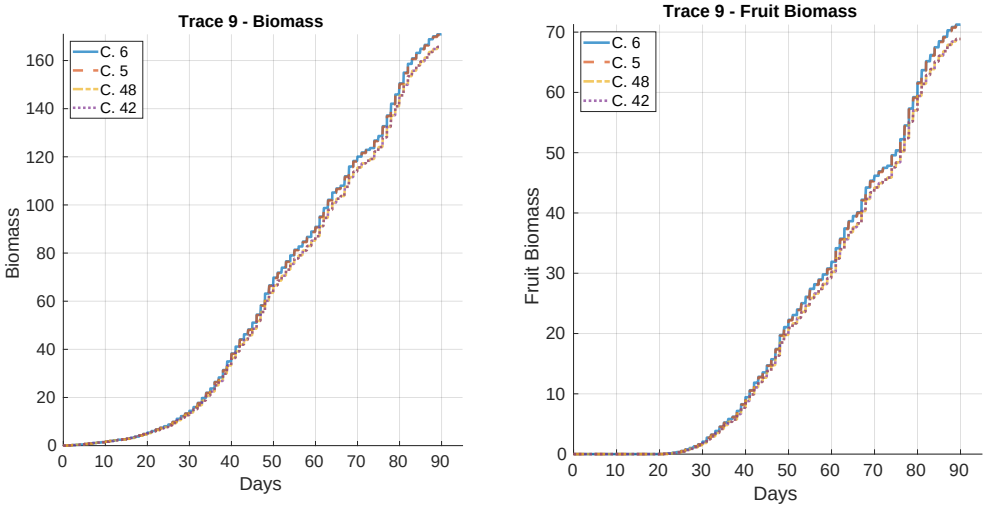


Figure 8.8: Highest difference obtained between best and worst strategy for a single input trace

The presented results clearly showcase how a common and straightforward control technique for greenhouse climatic regulation can be easily enhanced throughout simulations using a correct-by-construction source code, allowing us to search for the best parameters' value to implement either an overall solid strategy or an ad-hoc strategy for specific cases and conditions where a generic approach would lead to suboptimal outcomes. Moreover, we also showed that even enhancing a simple control strategy such as regulating windows opening can lead to an increase in yield of up to 2%, meaning that applying such strategy to more complex systems with more control options could lead to an even bigger increase.

8.3 Parameter Control via an MPC approach

In the previous section, we presented a methodology that allowed us to estimate the best combination for parameters FOT and FCT for different simulation traces. Nevertheless, with this methodology, we can only produce *static* and a-priori solutions by limiting ourselves to studying the effect of parameters' combinations on the entire simulation trace. However, by slightly modifying how our methodology works, we can change how predictions are performed, permitting us to perform parameter estimation not just on a complete input trace but rather at a fixed time step in order to implement an MPC approach capable of dynamically adjusting parameters on the fly.

The basis of implementing an MPC approach lies in the usage of a computational model, which is fed with future data to simulate possible behavior, choose the best parameters leading to such outcome, and force them into the model at the current time to ensure actual behavior will match predicted one. To do this, we changed our methodology in order to use two models simultaneously, as shown in figure 8.9: the *Main* model evolves by actual data sampled every five seconds and acts as a twin for crops in the greenhouse. From it, its current state is copied into the *prediction* model, which is fed with an input trace representing forecast data of the immediate future. Using such data, the prediction model computes possible scenarios to search which parameter combination results in the highest yield and communicates them to inform the end user about a possible change in the control strategy. If the user agrees, the parameters are updated automatically in both the controller and the main model.

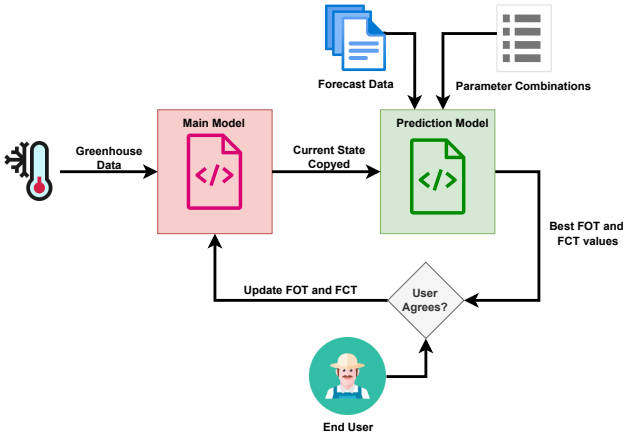


Figure 8.9: General Schema of proposed MPC approach

The presented scheme has been implemented starting from the model's source code used in the previous section by starting two distinct processes, one for the main model and a second for the prediction one, working as shown in schema 8.10. At the start of each new day, the main process communicates to the prediction one its state, represented by states' variables values, and asks it for a prediction. Using such information, the prediction model resets itself to copy the main's state fully and, from there, performs the search process to find which combination of parameters leads to the best result, using as input trace a forecast for the current day. In our case, since actual forecasts for each day were not available, we used as input trace the actual one available for the day but with the addition of some noise. Once the search task is over, the prediction process communicates its results to the main one and halts until a new request is submitted. Meanwhile, the main process receives parameter combinations for windows opening and, in our case, immediately applies them. Finally, it proceeds to simulate the model using actual daily data from the input trace until reaching the beginning of the next day, where the loop restarts.

Using the developed code, simulations were performed over various traces, and results were compared against the best static solution found in the previous section, obtaining results depicted in figure 8.11, which display all state variables for the tomato model for two models, one simulated under our MPC control and another one under the optimal strategy for considered trace found in the previous section. Moreover, the figure also shows temperature data provided as input to both models. For all presented plots, we also overlapped how FCT and FOT parameters evolve. Final results show how, in our case, differences between the MPC control and an a-priori control strategy are minimal, remaining below 0,1%. Still, we must consider that we compared our approach with the best possible obtainable only *after* all climatic data have been gathered, and even in this situation, our approach manages to remain on par with it thanks to the continuous adaptations

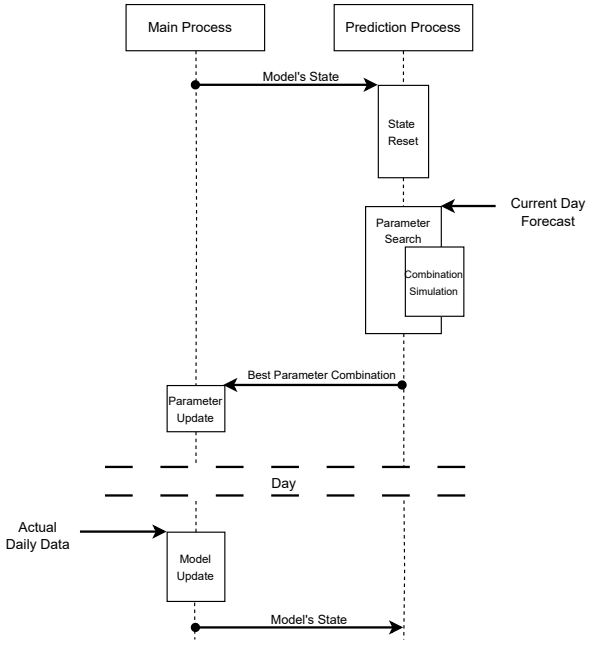


Figure 8.10: Flow diagram showing how main and prediction processes communicate.

the model performs and clearly visible in presented plots. As such, we believe that our methodology can become much more prominent in cases with more significant systems and more complex control strategies made of multiple actuators. Furthermore, we also want to point out how the developed technique separates itself from traditional climate control techniques, as it does not limit itself to using only climatic data to estimate optimal commands, but rather crops' responses to them, bringing the proposed solution much closer to an actually *speaking plant solution* rather than a traditional environmental control technique.

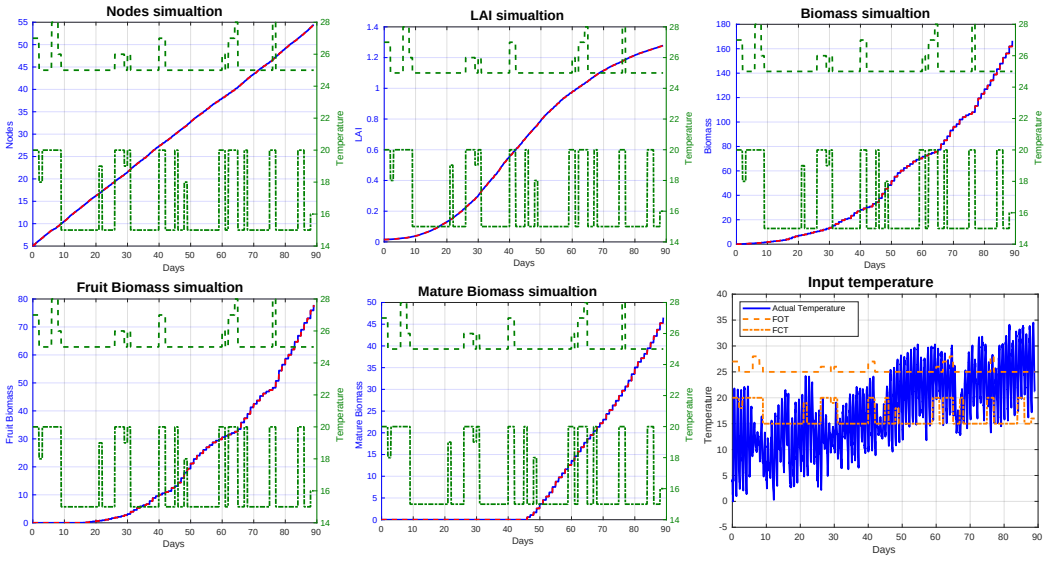


Figure 8.11: MPC-based control against best static control strategy.

8.4 Methods Combination to achieve Spatio-Temporal Simulation

During the entire course of this document, we presented various methodologies aimed at monitoring and controlling different aspects of key components constituting a greenhouse system. However,

each time, we limited ourselves only to verifying the presented solution by itself, writing experiments and simulation setups aimed to showcase the benefits of the discussed methodology. Now, instead, we showcase how presented solutions can be combined by using them all at the same time and by combining them into a pipeline aimed at creating a model representing the entire greenhouse system both in space and time.

The combined system is depicted in figure 8.12, where each module is obtained by applying a methodology described in this document. The initial point of everything lies in data collection from the greenhouse, be it in real-time through sensors or obtained via a model of the greenhouse fed by external data. From there, data are localized by using soft sensors, which act as a sort of prism that splits incoming climatic conditions into their location-specific version to model microclimate in different spatial positions of the greenhouse accurately. In such locations, one or more crops will grow, and therefore climatic data directly feed crop models implementing TOMGRO. However, each crop can also potentially house pathogen spores, meaning each crop may start to become infected and spread the pathogen using the mechanism shown before. Finally, data obtained from crop models are provided both to end users and to a control system, which aggregates them and decides the best policy for the greenhouse.

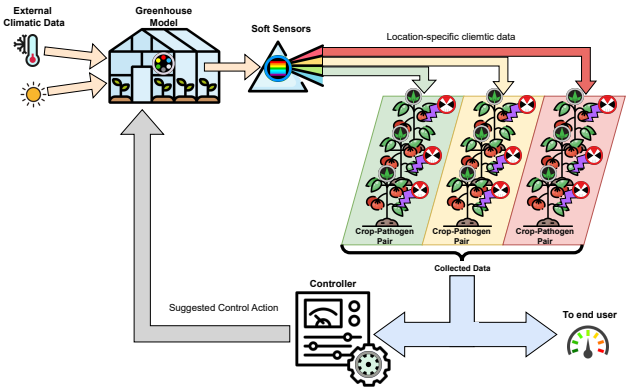


Figure 8.12: Pipeline combining all presented models

Similarly to how presented in section 6.3, we divided our virtual field into nine different areas, each one represented by a different crop/pathogen pair receiving data by a distinct virtual sensor, to model microclimatic heterogeneity, fed itself by our simple greenhouse model. Data produced by crops are then collected and shown to users while also feeding the control system, which plans how to adjust the current strategy to maximize the final outcome. Figure 8.13 shows the evolution over time of crops' and pathogens' main state variables. There, we can clearly see how all crops evolve following different curves dictated by the microclimate they grow inside, leading to different outcomes. Moreover, we can also notice how the pathogen expands, starting with crop 1 (top-left corner) and expanding onto others in the same way as modeled previously. Interestingly, we can also see how microclimatic differences affect pathogens' growth, such as crop 9 being the last one getting infected but having pathogen's growth much quicker with respect to other crops infected earlier, even leading to worse outcomes. Meanwhile, figure 8.14 depicts a spatial representation of the crop's status across the field for those same state variables on the final day of simulation. From it, we can clearly see the variability in crops' production caused by both the difference between the microclimate and pathogen development.

With this setup, we have now shown how all presented methodologies and models can be used together. In our case, we limited ourselves only to simulate using a single input trace sampled from the field. However, we can easily swap out the greenhouse model we used with actual data coming from sensors positioned in the greenhouse. In this way, we can transform our system to mirror in real-time crop's state, similarly to how an actual digital twin actually behaves.

8.5 Scalability analysis for bigger crops' grids

As a follow-up to the test presented in the previous section, we decided to test the scalability of our solution and extend it to bigger grids of crops. As such, experiments with a grid dimension of 5x5, 10x10, 20x20, 30x30, and 40x40 were conducted, and metrics regarding memory usage and timing were gathered. For each experiment, four metrics were registered: the time required by the

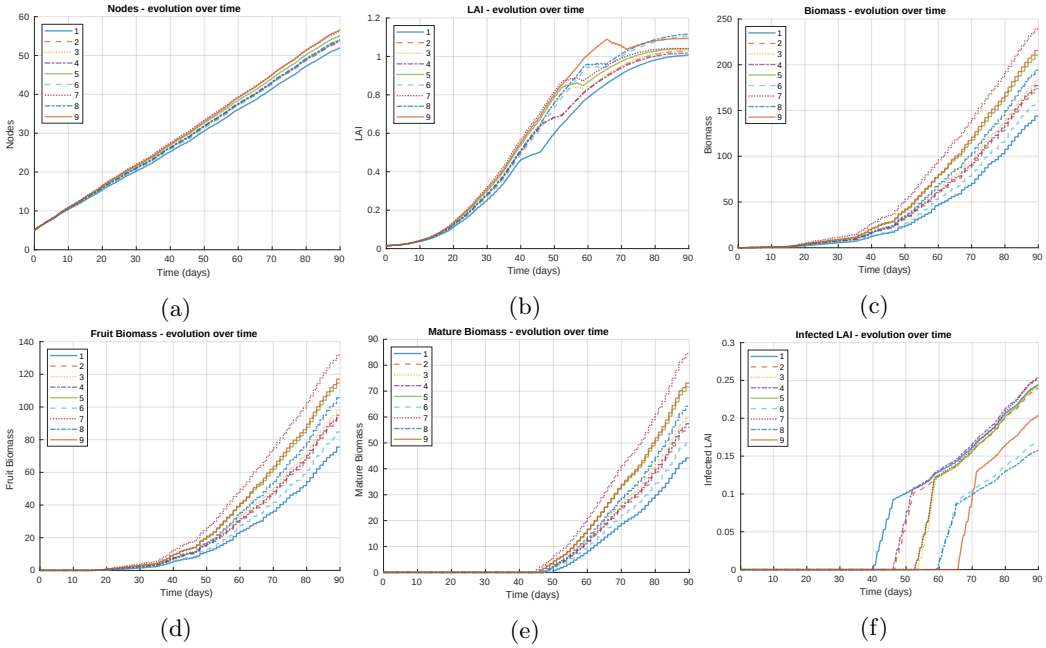


Figure 8.13: Simulation results for our combined model: a) nodes, b) LAI, biomass (c-total, d-fruit, e-mature), and f) infected leaves

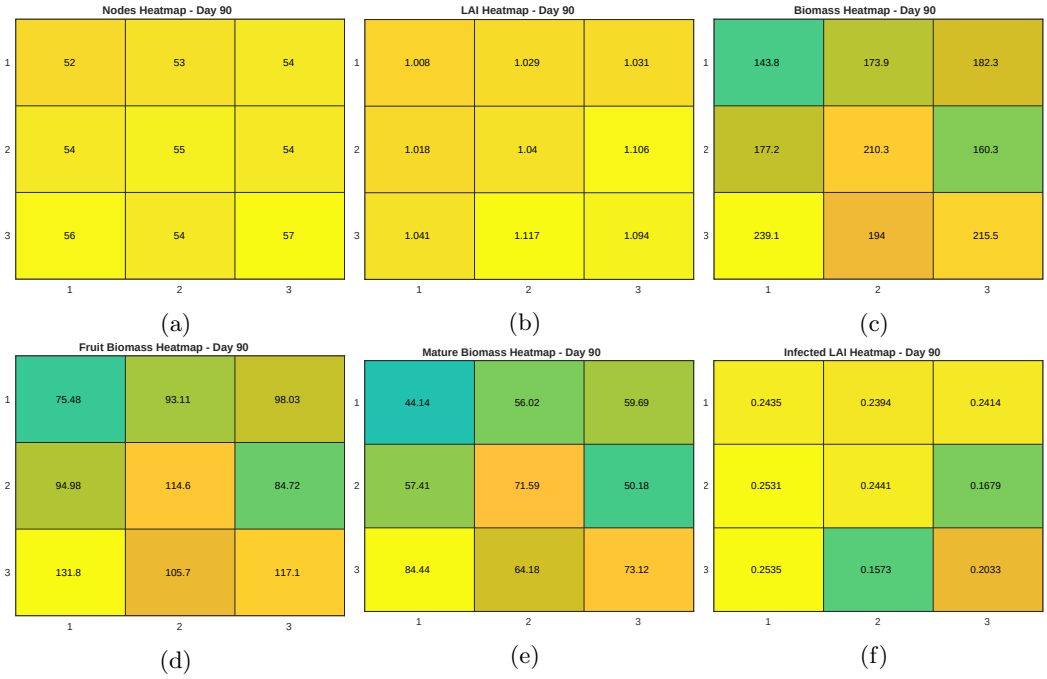


Figure 8.14: Spatial results for our combined model: a) nodes, b) LAI, biomass (c-total, d-fruit, e-mature) and f) infected leaves

ESCET tool to generate the C source code, the total size of the generated files, the time required to run a full simulation using a given input file, and the memory used by the program while running. All programs were set up to run using the same main file, which was responsible only for reading the input, and all logs were turned off to prevent tests with a higher number of automata from being penalized just due to having more variables to log. Tests were all conducted on a 13th gen Intel i7 machine with 16 GB RAM.

Table 8.1 presents described metrics for considered test configurations. Regarding code generation, we can clearly see that, as the problem's size increases, generation time grows exponentially with it, as we can clearly see in figure 8.15: for instance, the code generation time increases from six seconds for the 3x3 test to more than three hours (11682 seconds) for the 40x40 test. Similarly, the code size also scales accordingly, growing from 308KB up to 41MB. Both results for code generation were to be expected, with an increment of both time and memory growing exponentially with respect to the number of simulated crops' blocks. Obtained results regarding code generation may seem to point out that the procedure can become rapidly ineffective with the increase of the problem size; however, we must remember that the code generation process is usually once in the software's lifetime, and therefore, the time requirement can be easily overcome by the lifespan of the program. Similarly to code generation, the simulation time, show in Figure 8.17, also rapidly

Table 8.1: Performance metrics for varying test sizes

Test	Codegen Time (s)	Code Size	Sim Time (s)	Memory Usage
3x3	6.00	308KB	1.060	25MB
5x5	9.00	704KB	1.940	26MB
10x10	19.0	2.60MB	9.630	26MB
20x20	235.0	11.0MB	82.48	30MB
30x30	2170	23.0MB	391.3	35MB
40x40	$11.682 \cdot 10^3$	41.0MB	1180	43MB

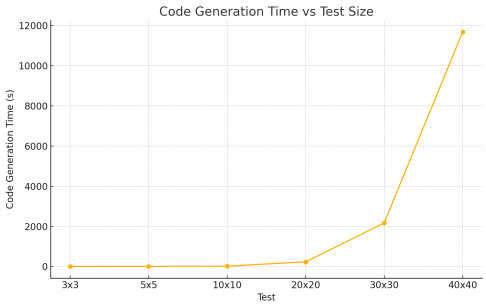


Figure 8.15: Code generation time vs. Grid size

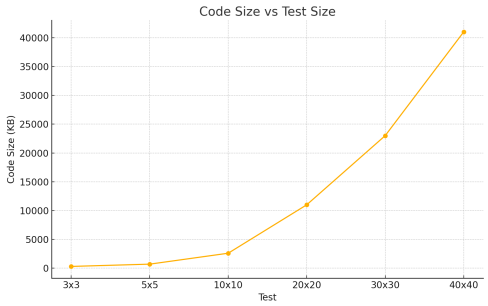


Figure 8.16: Code size vs. Grid size

increase with the problem scale, as consequence of the increased number of automata simulated. Interestingly, memory usage (figure 8.18) increase at a much smaller rate with respect to other metrics.

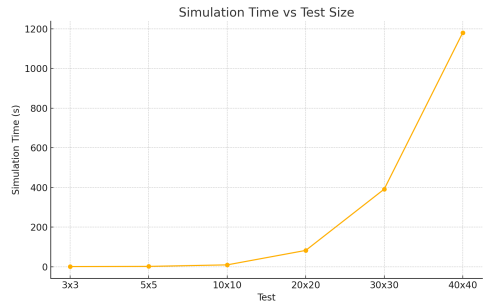


Figure 8.17: Simulation time vs. Grid size

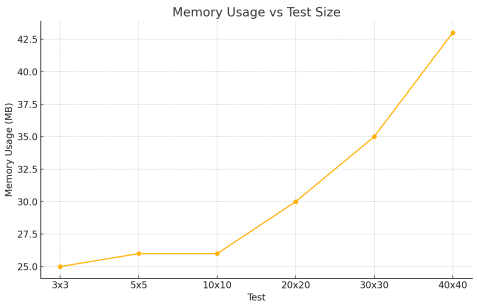


Figure 8.18: Memory usage vs. Grid size

Reported experiments show that the presented methodology can be easily scaled up to an

arbitrary grid of crops, although they also underline some limitations caused by the tools used. In particular, the algorithms used by the ESCET tool to perform code generation seem to be a possible bottleneck for our procedure, meaning future investigations should focus on searching for different code synthesis techniques capable of outperforming presented ones and producing more optimized results.

Chapter 9

Conclusions

The research conducted during this doctoral thesis aims to lay down the foundations needed to take a significant step toward the integration of computational techniques and modern agriculture. By taking as a starting point a simple model of a greenhouse system, we broke it down into its core components and mapped them into a digital twin architecture, taking advantage of the vast historical knowledge in a matter of modeling, control, and computational techniques developed throughout the years. For each identified system component, we addressed its issues and limitations and proceeded to overcome them by applying formal techniques typical of system engineering.

First, we addressed the critical importance of how climatic variables, temperature above all, are monitored by showing that there is a contradictory behavior between simulation tools and reality, the formers assuming complete homogeneity for climatic variables, whereas the truth is that the differences between various microclimate zones of a greenhouse can reach important values. Therefore, we developed a cost-effective methodology to monitor greenhouse climatic variables at various locations, which could be easily used and replicated to extend covered positions. Moreover, we also aimed to keep computational costs low to improve usability on most infrastructure by applying a data-driven machine-learning approach based on linear regression that also allowed for better explainability of the model. Furthermore, thanks to its simplicity, the proposed approach was also reused for anomaly detection in the physical sensors, an essential functionality in an actual agricultural setup. Obtained results showed how our methodology reaches a prediction accuracy of 99% for temperature and 94% for relative humidity; furthermore, false sensor readings were promptly detected.

The second examined component in the greenhouse system was crops. Here, we took the well-known tomato model TOMGRO as a case study, mainly due to its popularity, which granted us access to a good amount of bibliography. The TOMGRO model was deeply studied and adapted following the formal specifications of *Discrete Time Hybrid Automata* (DTHA), which we identified to be the best formalism capable of representing the considered model. By using automata's ability to be composed together, we broke down the TOMGRO model into three sub-components, each one representing a key growth process: node development, leaf area index (LAI), and biomass accumulation. Each automaton was tuned using actual growth data collected from the field, and, in the end, they were composed and synchronized to re-obtain the entire model. Finally, the adapted TOMGRO model was simulated, and results showed that the obtained model could produce a good estimation for tomato growth when compared to actual data, reaching an index of agreement value in the validation phase higher than 0.9.

Subsequently, we extended our work to include plant-pathogen interactions by modeling the Oidium Neolycopersici disease using a model taken from a completely different study as a reference. The pathogen was also modeled as a DTHA capable of composing and synchronizing with our crop model, thanks to the introduction of a third model acting both as an LAI automaton replacement for the crop model and an estimator of damages dealt by the disease. In this way, communication between the crop and pathogen was established, thus permitting the two automata to evolve together. Once the models were simulated, simulation results depicted a scenario close to an actual one, providing a plausible estimation of damage dealt to the crop's total production. In addition, by taking as reference the usage of cellular automata to model epidemics, we also updated our methodology in order to allow the modeling of multiple pairs of crop-pathogen planted at different locations of a field. This was achieved by using synchronization between automata, which allows a pathogen model to communicate with its spatial neighbors and activate them, thus simulating the infection's spread.

Following the modeling of elements growing within the greenhouse, we shifted our focus to the

description of the greenhouse control systems. To do so, we developed a very simplistic greenhouse model to generate the greenhouse's internal conditions starting from the external one and to simulate its response to control actions, in our case limited only to a change in windows' openings. Using the aforementioned model, we applied supervisory control theory to showcase how to automatically synthesize a supervisor capable of forcing wished requirements into the greenhouse. To do so, we first abstracted our models of greenhouse and crop conditions into *Finite State Automata* in order to make them compatible with the ESCET framework used to perform synthesis. On those models, we laid our requirements to restrict the window's opening behavior, and a supervisor was automatically synthesized and integrated into the closed loop with others DTHA to simulate the fully controlled system. The result was a supervisor enforcing a policy that opened and closed greenhouse windows according to the tendency of the internal temperature, similar to how the actual controller behaves.

On a similar note, we also showcase how our developed model could be also used to perform exploration and assessment of already-implemented control strategies. To do so, we first expanded our model system by adding a control strategy relying on user-defined parameters and, subsequently, we used the ESCET framework to perform code synthesis and obtain an executable source code correct by construction of our entire model. Next, we implemented a brute-force search algorithm using the obtained code to find out the best set of parameters for the controller to maximize the outcome. Using our algorithm, we simulated our model onto sixteen different simulation traces for forty-eight possible parameter combinations and ranked them to search for the best ones. Results showed us that no parameter combination was actually the best one across all traces, but each trace had a different combination, leading to the best results and providing end users with helpful information on which combination to adopt depending on the considered time period. Furthermore, we also adapt our search algorithm in order to perform an online prediction by using a model fed by actual data and a second one fed by forecasts, thus implementing a hybrid MPC-Speaking Plant approach where the forecast model searched for the best parameters in the provided horizon to maximize crop's outcome, and communicated it to the main model, which subsequently proceeded to use such combination while evolving with actual data.

Finally, we showcased how presented methodologies were not only standalone solutions but actually parts of the entire system. As such, we composed them all by connecting the greenhouse model to soft sensors to specialize climatic data for different locations inside a 9x9 grid where multiple crops were positioned. To each crop, a pathogen model was associated, with only being allowed to burst out, whereas other ones were coded to remain silent. As crops grew, the pathogen began to grow and spread, infecting nearby and damaging the entire field. Data from all nine locations and crops were also retrieved and provided both to end users to give them a view of the field's status and to the control system, which computed the best action possible to maximize crops' yield. Using this setup, we showcased how easily all presented solutions can be used together, simply thanks to choosing the usage of formal methods to represent the various system's components, while also successfully building a setup that, despite only in simulation, closely behaves as an actual digital world. Following this experiment, we repeated our procedure for growing grid sizes, up to reaching a 40 by 40 grid of crops. Such experiments showcased that our methodology can be effectively scaled up, with the main limiting factor becoming the tool used to generate the code.

Future Work

While the presented work showcased multiple techniques to use computer science knowledge and agricultural research together, it only presented a small subset of all possible junction points between the two research fields. Therefore, searching for more connections between the two subjects is certainly the first and most important point to explore in the future.

Besides this primary goal, there are also topic-specific goals related to perfecting presented techniques by addressing and solving limitations encountered while developing the described solutions. For microclimatic mapping, this translates into investigating smarter techniques to model soft sensors, such as the usage of neural networks, to produce more robust models resistant to nosy readings and missing data. Furthermore, exploring other methods of data acquisition, such as the usage of drones and mobile robots, may lead to a less cumbersome method of obtaining data needed to create models.

Regarding the topic of crops and pathogen modeling, obviously, investigating how our methodology applies to different crops and diseases is the primary task here. Investigating how other formalisms perform when used to model such systems is also a possible road.

Regarding simulation and control, the main limitations we encountered here came from the chosen tool, ESCET. While at the beginning of our work ESCET appeared promising due to how automata can be easily described using the CIF language, it provided us only limited tools while performing supervisor synthesis. As such, investigating other strategies to achieve such a process, such as performing modeling using different tools or applying different synthesis strategies that use more defined abstraction strategies, is certainly an open path we can follow to improve obtained results. Furthermore, the code generation feature provided by the tool, albeit functional, suffered heavy limitations caused by the time needed to synthesize the code and the overall quality of the source code itself, with simulation time quickly degrading as the problem size grew. In this regard, besides investigating other possible tools to generate code, another plausible path to follow is to skip the synthesis process and direct the development of ad-hoc source code by hand. In this case, however, the focus would be on boosting the code's performance by applying parallelization techniques to take advantage of the fact that evolutions of multiple crops and pathogens seem to be easily parallelizable due to the small interaction models have between them, meaning each crop could be mapped into a GPU/CPU core to boost simulation performance. Alternatively to searching for different tools, another path to follow is to improve the ESCET tool and add missing features. Throughout the development of this thesis, on multiple occasions, we entered into contact with curators of the ESCET tool, which is constantly evolving and updating. During such occasions, developers proved to be very open to feedback on updating the tool to add missing features or improve existing ones.

Appendix A

Automaton definitions

Nodes:

$$
\begin{aligned}
\text{Nodes} =& \{\text{Loc}_N, \text{Var}_N, \text{Lab}_N, \text{Edg}_N, \text{Act}_N, \text{Inv}_N\} \\
\text{Loc}_N =& \{N_0\} \\
\text{Var}_N =& \{N, c\} \\
\text{Lab}_N =& \{\tau, \text{hour}\} \\
\text{Edg}_N =& \{\text{upd}_0^N : (N_0, \text{hour}, \mu, N_0), s : (N_0, \tau, Id, N_0)\} \\
\text{Act}_N(N_0) : & f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(N) = N\} \\
\text{Inv}_N(N_0) = & \{v \in V | v(c) < T_p\}
\end{aligned}
$$

where

$$
\mu : (\nu_g, \nu_u) ; \nu_g \in \left\{v \in V | v(c) = T_p\right\}, \nu_u \in \{v \in V | v(c) = 0, v(N) = N + \Delta N\},
$$
$$
\Delta N = N_{mh} f_N(T)
$$

LAI:

$$
\begin{aligned}
\text{LAI} =& \{\text{Loc}_L, \text{Var}_L, \text{Lab}_L, \text{Edg}_L, \text{Act}_L, \text{Inv}_L\} \\
\text{Loc}_L =& \{L_0\} \\
\text{Var}_L =& \{\text{LAI}, \text{LAI}_c, \text{LAI}_{c-1}, c\} \\
\text{Lab}_L =& \{\tau, \text{hour}\} \\
\text{Edg}_L =& \{\text{upd}_0^L : (L_0, \text{hour}, \mu, L_0), s : (L_0, \tau, Id, L_0)\} \\
\text{Act}_L(L_0) : & f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(z) = z \forall z \in \text{Var}_L \backslash \{c\}\} \\
\text{Inv}_L(L_0) = & \{v \in V | v(c) < T_p\}
\end{aligned}
$$

where

$$
\mu : (\nu_g, \nu_u) ; \nu_g \in \left\{v \in V | v(c) = T_p\right\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(\text{LAI}) = \text{LAI} + \Delta\text{LAI} \wedge
$$
$$
v(\text{LAI}_c) = \frac{ab + \text{LAIMAX} \cdot N^d}{b + N^d} \wedge v(\text{LAI}_{c-1}) = \text{LAI}\}
$$

with

$$
\Delta\text{LAI} = \min(\delta\text{LAI}, \max(0, \text{LAI}_c - \text{LAI}_{c-1})), \delta\text{LAI} = \frac{bd(\text{LAIMAX} - a)N^{d-1}}{(b + N^d)^2}
$$

Biomass:

$$
\begin{aligned}
\text{Bio} =& \{\text{Loc}_B, \text{Var}_B, \text{Lab}_B, \text{Edg}_B, \text{Act}_B, \text{Inv}_B\} \\
\text{Loc}_B =& \{B_0, B_1, B_2\} \\
\text{Var}_B =& \{W, W_f, W_m, P_g, R_m, h, T_m, T_{dm}, N_{-1}, c\} \\
\text{Lab}_B =& \{\tau, \text{hour}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Edg}_B =& \{\mathrm{upd}_h^{00} : (B_0, \mathrm{hour}, \mu_h, B_0), \mathrm{upd}_d^{00} : (B_0, \mathrm{hour}, \mu_d^0, B_0), \mathrm{upd}_h^{01} : (B_0, \mathrm{hour}, \mu^{01}, B_1),\\
& \mathrm{upd}_h^{11} : (B_1, \mathrm{hour}, \mu_h, B_1), \mathrm{upd}_d^{11} : (B_1, \mathrm{hour}, \mu_d^1, B_1), \mathrm{upd}_h^{12} : (B_1, \mathrm{hour}, \mu^{12}, B_2),\\
& \mathrm{upd}_h^{22} : (B_2, \mathrm{hour}, \mu_h, B_2), \mathrm{upd}_d^{22} : (B_2, \mathrm{hour}, \mu_d^2, B_2), s_0 : (B_0, \tau, Id, B_0),\\
& s_1 : (B_1, \tau, Id, B_1), s_2 : (B_2, \tau, Id, B_2)\}\\
\mathrm{Act}_B(B_0) = \mathrm{Act}_B(B_1) =& \mathrm{Act}_B(B_2) :\\
& f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(z) = z \forall z \in \mathrm{Var}_B \backslash \{c\}\}\\
\mathrm{Inv}_B(B_0) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge N < N_{\mathrm{ff}})\}\\
\mathrm{Inv}_B(B_1) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge N < N_{\mathrm{ff}} + K_{\mathrm{ff}})\}\\
\mathrm{Inv}_B(B_2) =& \{v \in V | v(c) < T_p\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mu_h :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p \wedge v(h) < 24\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(h) = h + 1 \wedge\\
& v(T_{dm}) = \frac{T_m \cdot (h-1)}{h} + \frac{T}{h} \wedge v(T_{dm}) = \Delta T_{dm}(h), v(P_g) = P_g + \Delta P_g \wedge\\
& v(R_m) = R_m + \Delta R_m\}\\
\mu_d^0 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p \wedge v(h) > 24\}; \nu_u \in \{v \in V | v(c) = 0 \wedge\\
& v(W) = W + \Delta W \wedge v(h) = 0 \wedge v(P_g) = 0 \wedge v(R_m) = 0 \wedge v(T_m) = 0 \wedge\\
& v(T_{dm}) = 0 \wedge v(N_{-1}) = N\}\\
\mu^{01} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge N \geq N_{\mathrm{ff}}\}; \nu_u \in \{v \in V | v(W_f) = 0\}\\
\mu_d^1 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p \wedge v(h) > 24\}; \nu_u \in \{v \in V | v(c) = 0 \wedge\\
& v(W) = W + \Delta W \wedge v(W_f) = W_f + \Delta W_f \wedge v(h) = 0 \wedge v(P_g) = 0 \wedge v(R_m) = 0 \wedge\\
& v(T_m) = 0 \wedge v(T_{dm}) = 0 \wedge v(N_{-1}) = N\}\\
\mu^{12} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge \geq N_{\mathrm{ff}} + K_{\mathrm{ff}}\}; \nu_u \in \{v \in V | v(W_m) = 0\}\\
\mu_d^2 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p \wedge v(h) > 24\}; \nu_u \in \{v \in V | v(c) = 0 \wedge\\
& v(W) = W + \Delta W \wedge v(W_f) = W_f + \Delta W_f \wedge v(W_m) = W_m + \Delta W_m \wedge v(h) = 0 \wedge\\
& v(P_g) = 0 \wedge v(R_m) = 0 \wedge v(T_m) = 0 \wedge v(T_{dm}) = 0 \wedge v(N_{-1}) = N\}
\end{aligned}
$$

with

$$
\begin{aligned}
\Delta T_{dm}(h) =& \begin{cases} \frac{T_{dm} \cdot (h-1)}{h} + \frac{T}{h} & \text{if } 8 \leq h \leq 18 \\ T_{dm} & \text{otherwise} \end{cases}\\
\Delta P_g =& \frac{D \tau CO_2 \mathrm{PgR}(T)}{K} ln\left(\frac{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD}}{(1-m)\tau CO_2 + Q_e \cdot K \cdot \mathrm{PPFD} \cdot e^{-K \cdot \mathrm{LAI}}}\right)\\
\Delta R_m =& Q_{10}^{\frac{T-20}{10}} \mathrm{rm} \cdot (W - W_m)\\
\Delta W =& \min\Big(\mathrm{GR}_{\mathrm{net}} - p_1 \rho (N - N_{-1}), \Delta W_f + (V_{\max} - p1)\rho(N - N_{-1})\Big)\\
\mathrm{GR}_{\mathrm{net}} =& E(P_g - R_m)(1 - f_R(N)\\
\Delta W_f =& \begin{cases} \mathrm{GR}_{\mathrm{net}} \alpha_f F_f(T_m)(1 - e^{-\theta(N - N_{\mathrm{FF}})}) & \text{if } T_{dm} \leq T_{\mathrm{crit}} \\ \mathrm{GR}_{\mathrm{net}} \alpha_f F_f(T_m)(1 - e^{-\theta(N - N_{\mathrm{FF}})})(1 - 0.154(T_{dm} - T_{\mathrm{crit}})) & \text{if } T_{dm} > T_{\mathrm{crit}} \end{cases}\\
\Delta W_m =& D_f(T_m)(W_f - W_m)
\end{aligned}
$$

$$
\mathrm{PgR}(T) = \begin{cases} 0.074T & \text{if } 0 \leq T < 9 \\ 0.11T - 0.32 & \text{if } 9 \leq T < 12 \\ 1 & \text{if } 12 \leq T \leq 28 \\ 5 - \frac{T}{7} & \text{if } 28 < T \leq 35 \\ 0 & \text{otherwise} \end{cases} \quad f_R(N) = \begin{cases} 0 & \text{if } N < 1 \\ 0.205 - 0.0045N & \text{if } 1 \leq N < 12 \\ 0.217 - 0.006N & \text{if } 12 \leq N < 21 \\ 0.17 - 0.003N & \text{if } 21 \leq N \leq 30 \\ 0.07 & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
F_f(T) =& \max\Big(0, \min\big(1, 0.0625\,(T - \mathrm{cfgt})\big)\Big)\\
D_f(T) =& \max\Big(0, \min\big(1, 0.0714 * (T - 9)\big)\Big)
\end{aligned}
$$

Pathogen:

$$
\begin{aligned}
\text{Pathogen} =& \{\text{Loc}_P, \text{Var}_P, \text{Lab}_P, \text{Edg}_P, \text{Act}_P, \text{Inv}_P\}\\
\text{Loc}_P =& \{P_0, P_1, P_1^I, P_2, P_3\}\\
\text{Var}_P =& \{Y, h, c\}\\
\text{Lab}_P =& \{\tau, \text{hour}\}\\
\text{Edg}_P =& \{\text{upd}^0 : (P_0, \text{hour}, \mu^0, P_0), \text{upd}^{01} : (P_0, \text{hour}, \mu^{01}, P_1), \text{upd}^1 : (P_1, \text{hour}, \mu^1, P_1)\\
& \text{upd}^{1I} : (P_1, \text{hour}, \mu^{1I}, P_1^I)\text{upd}^I : (P_1^I, \text{hour}, \mu^I, P_1^I), \text{upd}^{I2} : (P_1^I, \text{hour}, \mu^{I2}, P_2),\\
& \text{upd}^2 : (P_2, \text{hour}, \mu^2, P_2)\text{upd}^{23} : (P_2, \text{hour}, \mu^{23}, P_3), \text{upd}^3 : (P_3, \text{hour}, \mu^3, P_3)\\
& s_0 : (P_0, \tau, Id, P_0), s_1 : (P_1, \tau, Id, P_1), s_I(P_1I, \tau, Id, P_1I), s_2 : (P_2, \tau, Id, P_2),\\
& s_3 : (P_3, \tau, Id, P_3)\}\\
\text{Act}_P(P_0) = \text{Act}_P(P_1) = \text{Act}_P(P_1I) = \text{Act}_P(P_2) = \text{Act}_P(P_3) :& \\
& f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(z) = z \forall z \in \text{Var}_P \backslash \{c\}\}\\
\text{Inv}_P(P_0) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge \text{LAI} < \text{LAI}_t)\}\\
\text{Inv}_P(P_1) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge h < \text{LP})\}\\
\text{Inv}_P(P_1I) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge h < \text{IP})\}\\
\text{Inv}_P(P_2) =& \{v \in V | v(c) < T_p \vee (v(c) = 0 \wedge h < \text{LP} + \text{IP})\}\\
\text{Inv}_P(P_3) =& \{v \in V | v(c) < T_p\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mu^0 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0\}\\
\mu^{01} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge \text{LAI} \geq \text{LAI}_t\}; \nu_u \in \{v \in V | v(h) = 0\}\\
\mu^1 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(h) = h + 1\}\\
\mu^{1I} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge v(h) \geq \text{LP}\}; \nu_u \in \{v \in V | v(x) = x \forall x \in \text{Var}_P\}\\
\mu^I :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(h) = h + 1\}\\
\mu^{I2} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge v(h) \geq \text{IP}\}; \nu_u \in \{v \in V | v(x) = x \forall x \in \text{Var}_P\}\\
\mu^2 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(h) = h + 1 \wedge\\
& v(Y) = Y + r_{\text{LINmax}}\text{DE}(T, \text{RH})\}\\
\mu^{23} :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = 0 \wedge v(h) \geq \text{LP} + \text{IP}\}; \nu_u \in \{v \in V | v(x) = x \forall x \in \text{Var}_P\}\\
\mu^3 :& (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(h) = h + 1 \wedge\\
& v(Y) = Y + r_{\text{Ymax}}\text{DE}(T, \text{RH})Y\left(1 - \frac{Y}{\frac{\text{LAI}}{\rho} + Y}\right)\}
\end{aligned}
$$

with

$$
\text{DE}(T, \text{RH}) = \begin{cases} d(T - T_{\min})^n (T_{\max} - T)^m (1 - e^{-a\text{RH}})^b & \text{if } T_{\min} \leq T \leq T_{\max} \\ 0 & \text{otherwise.} \end{cases}
$$

Infected LAI:

$$
\begin{aligned}
\text{I-LAI} =& \{\text{Loc}_I, \text{Var}_I, \text{Lab}_I, \text{Edg}_I, \text{Act}_I, \text{Inv}_I\}\\
\text{Loc}_I =& \{L_0, L_d\}\\
\text{Var}_I =& \{\text{LAI}, \text{LAI}_c, \text{LAI}_{c-1}, Y_{-1}, c\}\\
\text{Lab}_I =& \{\tau, \text{hour}\}\\
\text{Edg}_I =& \{\text{upd}_0 : (L_0, \text{hour}, \mu, L_0), \text{upd}_1 : (L_d, \text{hour}, \mu', L_d), \text{upd}_{01} : (L_0, \text{hour}, \mu_{01}, L_1),\\
& s_0 : (L_0, \tau, Id, L_0), s_1 : (L_d, \tau, Id, L_d)\}\\
\text{Act}_I(L_0) = \text{Act}_I(L_d) :& f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(z) = z \forall z \in \text{Var}_I \backslash \{c\}\}\\
\text{Inv}_I(L_0) =& \{v \in V | v(c) < T_p \vee (c = 0 \wedge Y \leq 0)\}\\
\text{Inv}_I(L_d) =& \{v \in V | v(c) < T_p\}
\end{aligned}
$$

where

$$
\mu : (\nu_g, \nu_u); \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(\text{LAI}) = \text{LAI} + \Delta\text{LAI} \wedge
$$

$$v(\mathrm{LAI}_c) = \frac{ab + \mathrm{LAIMAX} \cdot N^d}{b + N^d} \wedge v(\mathrm{LAI}_{c-1}) = \mathrm{LAI}\}$$

$$\mu^{01} : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = 0 \wedge v(Y) > 0\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(Y_1) = 0\}$$

$$\mu' : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge$$

$$v(\mathrm{LAI}) = \mathrm{LAI} + \Delta\mathrm{LAI} - \frac{Y - Y_{-1}}{\rho} \wedge v(\mathrm{LAI}_c) = \frac{ab + \mathrm{LAIMAX} \cdot N^d}{b + N^d} \wedge$$

$$v(\mathrm{LAI}_{c-1}) = \mathrm{LAI} \wedge v(Y_{-1}) = Y\}$$

with

$$\Delta\mathrm{LAI} = \min(\delta\mathrm{LAI}, \max(0, \mathrm{LAI}_c - \mathrm{LAI}_{c-1})), \delta\mathrm{LAI} = \frac{bd(\mathrm{LAIMAX} - a)N^{d-1}}{(b + N^d)^2}$$

Greenhouse:

$$\begin{aligned}
\text{Greenhouse} =& \{\mathrm{Loc}_G, \mathrm{Var}_G, \mathrm{Lab}_G, \mathrm{Edg}_G, \mathrm{Act}_G, \mathrm{Inv}_G\} \\
\mathrm{Loc}_G =& \{G_0\} \\
\mathrm{Var}_G =& \{T_{in}, k, c\} \\
\mathrm{Lab}_G =& \{\tau, \mathrm{hour}\} \\
\mathrm{Edg}_G =& \{\mathrm{upd}_0^G : (G_0, \mathrm{hour}, \mu, G_0), s : (G_0, \tau, Id, G_0)\} \\
\mathrm{Act}(G_0) : & f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \nu(z) = z \forall z \in \mathrm{Var}_G \backslash \{c\}\} \\
\mathrm{Inv}(G_0) = & \{v \in V | v(c) < T_p\}
\end{aligned}$$

where

$$\mu : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = T_p\}; \nu_u \in \{v \in V | v(c) = 0 \wedge v(k) = k + 1 \wedge$$

$$v(T_{\mathrm{in}}) = \tau \mathrm{Rad}(r + r_k \cos(\theta)) + T_{\mathrm{out}} + c \cdot e^{\frac{-kS}{r + r_k \cos(\theta)}}\}$$

P-Controller:

$$\begin{aligned}
\text{P-Controller} =& \{\mathrm{Loc}_C, \mathrm{Var}_C, \mathrm{Lab}_C, \mathrm{Edg}_C, \mathrm{Act}_C, \mathrm{Inv}_C\} \\
\mathrm{Loc}_C =& \{C_0\} \\
\mathrm{Var}_C =& \{\theta, c\} \\
\mathrm{Lab}_C =& \{\tau, \mathrm{sample}\} \\
\mathrm{Edg}_C =& \{\mathrm{upd}_0^C : (C_0, \mathrm{sample}, \mu, C_0), s : (C_0, \tau, Id, C_0)\}\} \\
\mathrm{Act}(C_0) : & f(t) = \nu \in \{v \in V | \nu(c) = t + k, k \in \mathbb{R} \wedge \frac{d\theta}{dt} = 0\} \\
\mathrm{Inv}(C_0) = & \{v \in V | v(c) < T_c\}
\end{aligned}$$

where

$$\mu : (\nu_g, \nu_u) ; \nu_g \in \{v \in V | v(c) = T_c\}; \nu_u \in \{v \in V | v(c) = 0 \wedge$$

$$v(\theta) = \max(\min(K \cdot (T - \mathrm{FCT}), 0), 90)\}$$

with

$$K = \frac{90}{\mathrm{FOT} - \mathrm{FCT}}$$

Bibliography

[1] O. B. Umar, L. A. Ranti, A. S. Abdulbaki, A. L. Bola, A. K. Abdulhamid, M. R. Biola, K. O. Victor, O. B. Umar, L. A. Ranti, A. S. Abdulbaki, A. L. Bola, A. K. Abdulhamid, M. R. Biola, and K. O. Victor, "Stresses in Plants: Biotic and Abiotic," in *Current Trends in Wheat Research*. IntechOpen, Oct. 2021.

[2] A. Martínez-Ruiz, I. Lopez-Cruz, A. Ruiz Garcia, J. Pineda, and J. prado hernández, "HortSyst: A dynamic model to predict growth, nitrogen uptake, and transpiration of greenhouse tomatoes," vol. 79, pp. 89–102, Feb. 2019.

[3] R. Drath and A. Horch, "Industrie 4.0: Hit or Hype? [Industry Forum]," *IEEE Industrial Electronics Magazine*, vol. 8, no. 2, pp. 56–58, 2014.

[4] S. Spellini, R. Chirico, M. Lora, and F. Fummi, "Languages and Formalisms to Enable EDA Techniques in the Context of Industry 4.0," in *2019 Forum for Specification and Design Languages (FDL)*, 2019, pp. 1–4.

[5] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.

[6] R. Lanotte, M. Merro, A. Munteanu, and S. Tini, "Formal Impact Metrics for Cyber-physical Attacks," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021, pp. 1–16.

[7] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 209–229.

[8] J. C. Zadoks, "Systems analysis and the dynamics of epidemics," *Phytopathology*, vol. 61, no. 6, pp. 600–610, 1971.

[9] R. Kopecká, M. Kameniarová, M. Černý, B. Brzobohatý, and J. Novák, "Abiotic Stress in Crop Production," *International Journal of Molecular Sciences*, vol. 24, no. 7, p. 6603, Apr. 2023.

[10] M. Hasanuzzaman, M. Fujita, M. C. M. T. Filho, T. A. R. Nogueira, and F. S. Galindo, *Sustainable Crop Production*. BoD – Books on Demand, Jun. 2020.

[11] Y. Wu, J. Liu, L. Zhao, H. Wu, Y. Zhu, I. Ahmad, and G. Zhou, "Abiotic stress responses in crop plants: A multi-scale approach," *Journal of Integrative Agriculture*, Sep. 2024.

[12] A. Bhujel, J. K. Basak, F. Khan, E. Arulmozhi, M. Jaihuni, T. Sihalath, D. Lee, J. Park, and H. T. Kim, "Sensor Systems for Greenhouse Microclimate Monitoring and Control: A Review," *Journal of Biosystems Engineering*, vol. 45, no. 4, pp. 341–361, Dec. 2020.

[13] J. W. Forrester, *Industrial Dynamics*. M.I.T. Press, 1961, google-Books-ID: 4CgzAAAAMAAJ.

[14] J. C. Zadoks, "Systems analysis and the dynamics of epidemics," *Phytopathology*, vol. 61, no. 6, pp. 600–610, 1971. [Online]. Available: https://www.apsnet.org/publications/phytopathology/backissues/Documents/1971Articles/Phyto61n06_600_A.pdf

[15] F. W. T. . L. Van Penning De Vries, *Simulation of Plant Growth and Crop Production*. Wageningen: Center Agricultural Pub & Document, Jan. 1982.

[16] R. Rabbinge, Ed., *Simulation and systems management in crop protection*, ser. Simulation monographs. Wageningen: Pudoc, 1989, no. 32.

[17] V. Rossi, S. Giosuè, and T. Caffi, "Modelling Plant Diseases for Decision Making in Crop Protection," in *Precision Crop Protection - the Challenge and Use of Heterogeneity*, E.-C. Oerke, R. Gerhards, G. Menz, and R. A. Sikora, Eds. Dordrecht: Springer Netherlands, 2010, pp. 241–258. [Online]. Available: https://link.springer.com/10.1007/978-90-481-9277-9_15

[18] D. Kasampalis, T. Alexandridis, C. Deva, A. Challinor, D. Moshou, and G. Zalidis, "Contribution of Remote Sensing on Crop Models: A Review," *Journal of Imaging*, vol. 4, p. 52, Mar. 2018.

[19] J. Jones, E. Dayan, L. Allen, Keulen, and H. Challa, "A dynamic tomato growth and yield model (TOMGRO)," *Transaction ASAE 34 (1991) 2. - ISSN 0001-2351*, vol. 34, Mar. 1991.

[20] V. Keulen and E. Dayan, "TOMGRO : a greenhouse - tomato simulation model," CABO-DLO, Tech. Rep., 1993. [Online]. Available: https://scholar.google.com/scholar_lookup?title=TOMGRO+%3A+a+greenhouse+-+tomato+simulation+model&author=Keulen%2C+van%2C+H.&publication_year=1993

[21] J. Jones, A. Kenig, and C. E. Vallejos, "Reduced state-variable tomato growth model," *Transactions of the ASAE*, vol. 42, pp. 255–265, Jan. 1999.

[22] L. Bacci, P. Battista, and B. Rapi, "Evaluation and adaptation of TOMGRO model to Italian tomato protected crops," *New Zealand Journal of Crop and Horticultural Science*, vol. 40, no. 2, pp. 115–126, Jun. 2012, publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01140671.2011.623706. [Online]. Available: https://doi.org/10.1080/01140671.2011.623706

[23] G. L. Schumann, *Plant diseases: their biology and social impact*. American Phytopathological Society Press, 1991.

[24] C. P. W. Barbora Mieslerová, Roger T. A. Cook and A. Lebeda, "Ecology of powdery mildews – influence of abiotic factors on their development and epidemiology," *Critical Reviews in Plant Sciences*, vol. 41, no. 6, pp. 365–390, 2022. [Online]. Available: https://doi.org/10.1080/07352689.2022.2138044

[25] R. A. Guzman-Plazola, R. Davis, and J. J. Marois, "Effects of relative humidity and high temperature on spore germination and development of tomato powdery mildew (leveillula taurica)," *Crop Protection*, vol. 22, no. 10, pp. 1157–1168, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0261219403001571

[26] R. Guzmán-Plazola, M. Fajardo-Franco, and M. Coffey, "Control of tomato powdery mildew (leveillula taurica) in the comarca lagunera, coahuila state, mexico, supported by the spray forecast model tomato.pm," *Crop Protection*, vol. 30, no. 8, pp. 1006–1014, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0261219411000937

[27] D. Jacob, D. R. David, A. Sztjenberg, and Y. Elad, "Conditions for development of powdery mildew of tomato caused by oidium neolycopersici," *Phytopathology®*, vol. 98, no. 3, pp. 270–281, 2008, pMID: 18944077. [Online]. Available: https://doi.org/10.1094/PHYTO-98-3-0270

[28] J. M. Whipps and S. P. Budge, "Effect of humidity on development of tomato powdery mildew (oidium lycopersici) in the glasshouse," *European Journal of Plant Pathology*, vol. 106, no. 4, pp. 395–397, 2000. [Online]. Available: https://doi.org/10.1023/A:1008745630393

[29] H. Jones, J. Whipps, and S. Gurr, "The tomato powdery mildew fungus oidium neolycopersici," *Molecular plant pathology*, vol. 2, pp. 303–9, 11 2001.

[30] J. Chelal and B. Hau, "Modelling the interaction between powdery mildew epidemics and host dynamics of tomato," *European Journal of Plant Pathology*, vol. 142, no. 3, pp. 461–479, 2015. [Online]. Available: https://doi.org/10.1007/s10658-015-0626-7

[31] ——, "Temporal dynamics of powdery mildew (oidium neolycopersici) and its effects on the host growth dynamics of tomato," *Journal of Phytopathology*, vol. 163, no. 9, pp. 711–722, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/jph.12368

[32] S. Chen, A. Liu, F. Tang, P. Hou, Y. Lu, and P. Yuan, "A Review of Environmental Control Strategies and Models for Modern Agricultural Greenhouses," *Sensors*, vol. 25, p. 1388, Jan. 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/5/1388

[33] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, Feb. 1995.

[34] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Linearizing Discrete-Time Hybrid Systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5357–5364, Oct. 2017.

[35] T. Stauner, "Discrete-Time Refinement of Hybrid Automata," in *Hybrid Systems: Computation and Control*, C. J. Tomlin and M. R. Greenstreet, Eds. Berlin, Heidelberg: Springer, 2002, pp. 407–420.

[36] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[37] W. J. Fokkink, M. A. Goorden, D. Hendriks, D. A. van Beek, A. T. Hofkamp, F. F. H. Reijnen, L. F. P. Etman, L. Moormann, J. M. van de Mortel-Fronczak, M. A. Reniers, J. E. Rooda, L. J. van der Sanden, R. R. H. Schiffelers, S. B. Thuijsman, J. J. Verbakel, and J. A. Vogel, "Eclipse ESCET™: The Eclipse Supervisory Control Engineering Toolkit," in *Tools and Algorithms for the Construction and Analysis of Systems*, S. Sankaranarayanan and N. Sharygina, Eds. Cham: Springer Nature Switzerland, 2023, pp. 44–52.

[38] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers, "CIF 3: Model-Based Engineering of Supervisory Controllers," in *Tools and Algorithms for the Construction and Analysis of Systems*, E. Ábrahám and K. Havelund, Eds. Berlin, Heidelberg: Springer, 2014, pp. 575–580.

[39] S. Revathi, N. Sivakumaran, and T. Radhakrishnan, "Design of solar-powered forced ventilation system and energy-efficient thermal comfort operation of greenhouse," *Materials Today: Proceedings*, vol. 46, pp. 9893–9900, 2021, international Mechanical Engineering Congress 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321005009

[40] B. Bailey, "Microclimate, physical processes and greenhouse technology," *Acta Hortic*, vol. 174, pp. 35–42, 1985.

[41] C. Bersani, A. Ouammi, R. Sacile, and E. Zero, "Model Predictive Control of Smart Greenhouses as the Path towards Near Zero Energy Consumption," *Energies*, vol. 13, no. 14, 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/14/3647

[42] R. Liao, S. Zhang, X. Zhang, M. Wang, H. Wu, and L. Zhangzhong, "Development of smart irrigation systems based on real-time soil moisture data in a greenhouse: Proof of concept," *Agricultural Water Management*, vol. 245, p. 106632, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037837742032179X

[43] K. Saranya, P. Uva Dharini, P. Uva Darshni, and S. Monisha, "IoT Based Pest Controlling System for Smart Agriculture," in *2019 International Conference on Communication and Electronics Systems (ICCES)*, 2019, pp. 1548–1552.

[44] M. C. Singh, J. Singh, and K. Singh, "Development of a microclimate model for prediction of temperatures inside a naturally ventilated greenhouse under cucumber crop in soilless media," *Computers and Electronics in Agriculture*, vol. 154, pp. 227–238, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016816991830022X

[45] X. Feng Li, L. L. Qin, G. Q. Ma, and G. Wu, "Modeling greenhouse temperature by means of PLSR and BPNN," in *2016 35th Chinese Control Conference (CCC)*, 2016, pp. 2196–2200.

[46] W. Hongkang, L. Li, W. Yong, M. Fanjia, W. Haihua, and N. Sigrimis, "Recurrent Neural Network Model for Prediction of Microclimate in Solar Greenhouse," *IFAC-PapersOnLine*, vol. 51, no. 17, pp. 790–795, 2018, 6th IFAC Conference on Bio-Robotics BIOROBOTICS 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318312151

[47] C. R. Bojacá, R. Gil, and A. Cooman, "Use of geostatistical and crop growth modelling to assess the variability of greenhouse tomato yield caused by spatial temperature variations," *Computers and Electronics in Agriculture*, vol. 65, no. 2, pp. 219–227, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169908002202

[48] J. López-Martínez, J. L. Blanco-Claraco, J. Pérez-Alonso, and Ángel J. Callejón-Ferre, "Distributed network for measuring climatic parameters in heterogeneous environments: Application in a greenhouse," *Computers and Electronics in Agriculture*, vol. 145, pp. 105–121, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917308189

[49] C.-J. Zhao, M. Li, X.-T. Yang, C.-H. Sun, J.-P. Qian, and Z.-T. Ji, "A data-driven model simulating primary infection probabilities of cucumber downy mildew for use in early warning systems in solar greenhouses," *Computers and Electronics in Agriculture*, vol. 76, no. 2, pp. 306–315, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169911000597

[50] N. Katsoulas, K. P. Ferentinos, A. Tzounis, T. Bartzanas, and C. Kittas, "Spatially distributed greenhouse climate control based on wireless sensor network measurements," in *ActaHortic.*, no. 1154. International Society for Horticultural Science (ISHS), Leuven, Belgium, Mar 2017, pp. 111–120. [Online]. Available: https://doi.org/10.17660/ActaHortic.2017.1154.15

[51] A. Saberian and S. M. Sajadiye, "The effect of dynamic solar heat load on the greenhouse microclimate using CFD simulation," *Renewable Energy*, vol. 138, pp. 722–737, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148119301223

[52] K. Xu, X. Guo, J. He, B. Yu, J. Tan, and Y. Guo, "A study on temperature spatial distribution of a greenhouse under solar load with considering crop transpiration and optical effects," *Energy Conversion and Management*, vol. 254, p. 115277, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890422000735

[53] X. Cheng, D. Li, L. Shao, and Z. Ren, "A virtual sensor simulation system of a flower greenhouse coupled with a new temperature microclimate model using three-dimensional CFD," *Computers and Electronics in Agriculture*, vol. 181, p. 105934, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920331392

[54] A. Morris, G. Montague, and M. Tham, "Soft-sensors in industrial process control," in *IEE Colloquium on Applied Developments in Process Control*, 1989, pp. 1/1–1/3.

[55] L. Fortuna, S. Graziani, A. Rizzo, and M. G. Xibilia, *Soft Sensors for Monitoring and Control of Industrial Processes*. Springer Science & Business Media, 05 2007.

[56] S. Joe Qin, "Statistical process monitoring: basics and beyond," *Journal of Chemometrics*, vol. 17, no. 8, pp. 480–502, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cem.800

[57] Y. Jiang, S. Yin, J. Dong, and O. Kaynak, "A review on soft sensors for monitoring, control, and optimization of industrial processes," *IEEE Sensors Journal*, vol. 21, no. 11, pp. 12 868–12 881, 2021.

[58] J. A. Sánchez, F. Rodríguez, J. L. Guzmán, and M. R. Arahal, "Virtual Sensors for Designing Irrigation Controllers in Greenhouses," *Sensors*, vol. 12, no. 11, pp. 15 244–15 266, 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/11/15244

[59] C. H. Guzmán, J. L. Carrera, H. A. Durán, J. Berumen, A. A. Ortiz, O. A. Guirette, A. Arroyo, J. A. Brizuela, F. Gómez, A. Blanco, H. R. Azcaray, and M. Hernández, "Implementation of Virtual Sensors for Monitoring Temperature in Greenhouses Using CFD and Control," *Sensors*, vol. 19, no. 1, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/1/60

[60] B. Basso, L. Liu, and J. T. Ritchie, "A Comprehensive Review of the CERES-Wheat, -Maize and -Rice Models' Performances," in *Advances in Agronomy*, ser. Advances in Agronomy, D. L. Sparks, Ed. Academic Press, Jan. 2016, vol. 136, pp. 27–132. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065211315001480

[61] H. H. v. Laar, *SUCROS97: Simulation of Crop Growth for Potential and Water-limited Production Situations: As Applied to Spring Wheat.* DLO Research Institute for Agrobiology and Soil Fertility, 1997.

[62] K. J. Boote, R. Seepaul, M. J. Mulvaney, A. K. Hagan, M. Bashyal, S. George, I. Small, and D. L. Wright, "Adapting the CROPGRO model to simulate growth and production of Brassica carinata, a bio-fuel crop," *GCB Bioenergy*, vol. 13, no. 7, pp. 1134–1148, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/gcbb.12838

[63] E. Heuvelink, "Evaluation of a Dynamic Simulation Model for Tomato Crop Growth and Development," *Annals of Botany*, vol. 83, no. 4, pp. 413–422, Apr. 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305736498908320

[64] A. Kenig and J. W. Jones, "TOMGRO V3.0: A dynamic model of tomato growth and yield," in *Optimal Environmental Control for Indeterminate Greenhouse Crops: Final Report, Report No. IS-1995-91*, 1997, pp. Chap. II–5.

[65] A. Cooman and E. Schrevens, "A Monte Carlo Approach for estimating the Uncertainty of Predictions with the Tomato Plant Growth Model, Tomgro," *Biosystems Engineering*, vol. 94, no. 4, pp. 517–524, Aug. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511006001784

[66] ——, "Sensitivity of the Tomgro Model to Solar Radiation Intensity, Air Temperature and Carbon Dioxide Concentration," *Biosystems Engineering*, vol. 96, no. 2, pp. 249–255, Feb. 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511006003394

[67] M. A. Vazquez-Cruz, R. Guzman-Cruz, I. L. Lopez-Cruz, O. Cornejo-Perez, I. Torres-Pacheco, and R. G. Guevara-Gonzalez, "Global sensitivity analysis by means of EFAST and Sobol' methods and calibration of reduced state-variable TOMGRO model using genetic algorithms," *Computers and Electronics in Agriculture*, vol. 100, pp. 1–12, Jan. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169913002482

[68] R. Shamshiri, D. Ahmad, A. Zakaria, W. I. Wan Ismail, H. Che Man, and M. Yamin, "Evaluation of the Reduced State-Variable TOMGRO Model using Boundary Data," Jul. 2016.

[69] L. Gong, M. Yu, S. Jiang, V. Cutsuridis, S. Kollias, and S. Pearson, "Studies of evolutionary algorithms for the reduced Tomgro model calibration for modelling tomato yields," *Smart Agricultural Technology*, vol. 1, p. 100011, Dec. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772375521000113

[70] E. J. Van Henten, "Sensitivity Analysis of an Optimal Control Problem in Greenhouse Climate Management," *Biosystems Engineering*, vol. 85, no. 3, pp. 355–364, Jul. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511003000680

[71] B. H. E. Vanthoor, C. Stanghellini, E. J. van Henten, and P. H. B. de Visser, "A methodology for model-based greenhouse design: Part 1, a greenhouse climate model for a broad range of designs and climates," *Biosystems Engineering*, vol. 110, no. 4, pp. 363–377, Dec. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511011000948

[72] B. H. E. Vanthoor, P. H. B. de Visser, C. Stanghellini, and E. J. van Henten, "A methodology for model-based greenhouse design: Part 2, description and validation of a tomato yield model," *Biosystems Engineering*, vol. 110, no. 4, pp. 378–395, Dec. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511011001413

[73] A. Sokolidi, R. Webster, A. Milne, M. Bielik, P. Morley, J. Clarkson, and J. West, "Spatial characteristics of the fungus powdery mildew (erysiphe neolycopersici) on tomatoes and its spread in industrial greenhouses," *Biostatistics Research*, pp. 18–30, 01 2023.

[74] A. Bhatia, A. Chug, and A. P. Singh, "Statistical analysis of machine learning techniques for predicting powdery mildew disease in tomato plants," *International Journal of Intelligent Engineering Informatics*, vol. 9, no. 1, pp. 24–58, 2021. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJIEI.2021.116087

[75] A. Bhatia, A. Chug, A. P. Singh, R. P. Singh, and D. Singh, "A machine learning-based spray prediction model for tomato powdery mildew disease," *Indian Phytopathology*, vol. 75, no. 1, pp. 225–230, 2022. [Online]. Available: https://doi.org/10.1007/s42360-021-00430-3

[76] S. Wolfram, *A New Kind of Science*, first edition ed. Wolfram Media Inc, 2002.

[77] S. H. White, A. M. del Rey, and G. R. Sánchez, "Modeling epidemics using cellular automata," *Applied Mathematics and Computation*, vol. 186, pp. 193–202, Mar. 2007.

[78] V. Rossi, F. Salinari, S. Poni, T. Caffi, and T. Bettati, "Addressing the implementation problem in agricultural decision support systems: the example of vite.net®," *Computers and Electronics in Agriculture*, vol. 100, pp. 88–99, Jan. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169913002536

[79] L. Ahmad and S. S. Mahdi, "Decision Support System for Precision Farming," in *Satellite Farming: An Information and Technology Based Agriculture*, L. Ahmad and S. S. Mahdi, Eds. Cham: Springer International Publishing, 2018, pp. 167–180. [Online]. Available: https://doi.org/10.1007/978-3-030-03448-1_13

[80] J. W. Jones, G. Hoogenboom, C. H. Porter, K. J. Boote, W. D. Batchelor, L. A. Hunt, P. W. Wilkens, U. Singh, A. J. Gijsman, and J. T. Ritchie, "The DSSAT cropping system model," *European Journal of Agronomy*, vol. 18, no. 3, pp. 235–265, Jan. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1161030102001077

[81] A. de Wit, H. Boogaard, D. Fumagalli, S. Janssen, R. Knapen, D. van Kraalingen, I. Supit, R. van der Wijngaart, and K. van Diepen, "25 years of the WOFOST cropping systems model," *Agricultural Systems*, vol. 168, pp. 154–167, Jan. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308521X17310107

[82] P. Aggarwal, N. Kalra, S. Chander, and H. Pathak, "InfoCrop: A dynamic simulation model for the assessment of crop yields, losses due to pests, and environmental impact of agro-ecosystems in tropical environments. I. Model description," *Agricultural Systems*, vol. 89, no. 1, pp. 1–25, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308521X05001459

[83] T. Foster, N. Brozović, A. P. Butler, C. M. U. Neale, D. Raes, P. Steduto, E. Fereres, and T. C. Hsiao, "AquaCrop-OS: An open source version of FAO's crop water productivity model," *Agricultural Water Management*, vol. 181, pp. 18–22, Feb. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378377416304589

[84] R. L. McCown, G. L. Hammer, J. N. G. Hargreaves, D. P. Holzworth, and D. M. Freebairn, "APSIM: a novel software system for model development, model testing and simulation in agricultural systems research," *Agricultural Systems*, vol. 50, pp. 255–271, Jan. 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0308521X9400055V

[85] R. L. McCown, "Locating agricultural decision support systems in the troubled past and socio-technical complexity of 'models for management'," *Agricultural Systems*, vol. 74, pp. 11–25, Oct. 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0308521X02000203

[86] S. Chen, A. Liu, F. Tang, P. Hou, Y. Lu, and P. Yuan, "A Review of Environmental Control Strategies and Models for Modern Agricultural Greenhouses," *Sensors*, vol. 25, p. 1388, Jan. 2025, number: 5 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/25/5/1388

[87] S. Zhang, Y. Guo, H. Zhao, Y. Wang, D. Chow, and Y. Fang, "Methodologies of control strategies for improving energy efficiency in agricultural greenhouses," *Journal of Cleaner Production*, vol. 274, p. 122695, Nov. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652620327426

[88] E. González-Domínguez, T. Caffi, A. Bodini, L. Galbusera, and V. Rossi, "A fuzzy control system for decision-making about fungicide applications against grape downy mildew," *European Journal of Plant Pathology*, vol. 144, 10 2015.

[89] S. Kouro, P. Cortes, R. Vargas, U. Ammann, and J. Rodriguez, "Model predictive control—a simple and powerful method to control power converters," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1826–1838, 2009.

[90] A. Castañeda-Miranda and V. M. Castaño, "Smart frost control in greenhouses by neural networks models," *Computers and Electronics in Agriculture*, vol. 137, pp. 102–114, May 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169916305798

[91] H. Nishina, "Development of Speaking Plant Approach Technique for Intelligent Greenhouse," *Agriculture and Agricultural Science Procedia*, vol. 3, pp. 9–13, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210784315000054

[92] Y. Hashimoto, "RECENT STRATEGIES OF OPTIMAL GROWTH REGULATION BY THE SPEAKING PLANT CONCEPT," *Acta Horticulturae*, no. 260, pp. 115–122, Sep. 1989. [Online]. Available: https://www.actahort.org/books/260/260_5.htm

[93] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular Specification of Hybrid Systems in Charon," in *Hybrid Systems: Computation and Control*, N. Lynch and B. H. Krogh, Eds. Berlin, Heidelberg: Springer, 2000, pp. 6–19.

[94] R. Sanfelice, D. Copp, and P. Nanez, "A toolbox for simulation of hybrid systems in matlab/simulink: hybrid equations (HyEQ) toolbox," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, ser. HSCC '13. New York, NY, USA: Association for Computing Machinery, Apr. 2013, pp. 101–106. [Online]. Available: https://dl.acm.org/doi/10.1145/2461328.2461346

[95] C. Georgiou, P. P. Mavrommatis, and J. A. Tauber, "Implementing Asynchronous Distributed Systems Using the IOA Toolkit," Oct. 2004. [Online]. Available: https://dspace.mit.edu/handle/1721.1/30412

[96] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Information and Computation*, vol. 185, pp. 105–157, Aug. 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540103000671

[97] A. Hartmanns and H. Hermanns, "The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, vol. 8413, pp. 593–598. [Online]. Available: http://link.springer.com/10.1007/978-3-642-54862-8_51

[98] "Ariadne : a framework for reachability analysis of hybrid automata," in *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/215697417_Ariadne_a_framework_for_reachability_analysis_of_hybrid_automata

[99] S. Bak and M. Caccamo, "Computing reachability for nonlinear systems with hycreate," ACM, 2013.

[100] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "HYTECH: the next generation," in *Proceedings 16th IEEE Real-Time Systems Symposium*, Dec. 1995, pp. 56–65, iSSN: 1052-8725. [Online]. Available: https://ieeexplore.ieee.org/document/495196

[101] "Uppaal SMC tutorial," *ResearchGate*, Dec. 2024. [Online]. Available: https://www.researchgate.net/publication/270596354_Uppaal_SMC_tutorial

[102] A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani, "AMYTISS: Parallelized Automated Controller Synthesis for Large-Scale Stochastic Systems," May 2020. [Online]. Available: http://arxiv.org/abs/2005.06191

[103] S. Mouelhi, A. Girard, and G. Gössler, "CoSyMA: a tool for controller synthesis using multi-scale abstractions," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, ser. HSCC '13. New York, NY, USA: Association for Computing Machinery, Apr. 2013, pp. 83–88. [Online]. Available: https://dl.acm.org/doi/10.1145/2461328.2461343

[104] B. Aminof, G. D. Giacomo, A. Lomuscio, A. Murano, and S. Rubin, "Synthesizing strategies under expected and exceptional environment behaviors," vol. 2, Jul. 2020, pp. 1674–1680, iSSN: 1045-0823. [Online]. Available: https://www.ijcai.org/proceedings/2020/232

[105] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, ser. HSCC '11. New York, NY, USA: Association for Computing Machinery, Apr. 2011, pp. 313–314. [Online]. Available: https://doi.org/10.1145/1967701.1967747

[106] P. Nilsson, N. Ozay, and J. Liu, "Augmented finite transition systems as abstractions for control synthesis," *Discrete Event Dynamic Systems*, vol. 27, 2017. [Online]. Available: https://link.springer.com/epdf/10.1007/s10626-017-0243-z

[107] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Automatica*, vol. 48, no. 5, pp. 947–953, May 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000510981200088X

[108] J. Liu and N. Ozay, "Finite abstractions with robustness margins for temporal logic-based control synthesis," *Nonlinear Analysis: Hybrid Systems*, vol. 22, pp. 1–15, Nov. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1751570X16000169

[109] M. A. Reniers and J. M. van de Mortel-Fronczak, "An Engineering Perspective on Model-Based Design of Supervisors," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 257–264, Jan. 2018.

[110] L. Ricker, S. Lafortune, and S. Genc, "DESUMA: A Tool Integrating GIDDES and UMDES," in *2006 8th International Workshop on Discrete Event Systems*, Jul. 2006, pp. 392–393.

[111] R. Meira-Góes, A. Wintenberg, S. Matsui, and S. Lafortune, "MDESops: An Open-Source Software Tool for Discrete Event Systems Modeled by Automata," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 6093–6098, Jan. 2023.

[112] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," in *2006 8th International Workshop on Discrete Event Systems*, Jul. 2006, pp. 384–385.

[113] E. Brentarolli, S. Migliorini, D. Quaglia, and C. Tomazzoli, "Mapping Micro-Climate in a Greenhouse Through a Context-Aware Recurrent Neural Network," 2023, pp. 113–117. [Online]. Available: https://ieeexplore.ieee.org/document/10291595

[114] C. Tomazzoli, E. Brentarolli, D. Quaglia, and S. Migliorini, "Estimating Greenhouse Climate Through Context-Aware Recurrent Neural Networks Over an Embedded System," *IEEE Transactions on AgriFood Electronics*, vol. 2, no. 2, pp. 554–562, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10663269

[115] L. Bacci, P. Battista, B. Rapi, A. Pardossi, L. Incrocci, and G. Carmassi, "SGx: sistema di supporto alla fertirrigazione del pomodoro in serra," *Informatore Fitopatologico*, pp. 32–35, Dec. 2005.

[116] S.-L. Fang, Y.-H. Kuo, L. Kang, C.-C. Chen, C.-Y. Hsieh, M.-H. Yao, and B.-J. Kuo, "Using Sigmoid Growth Models to Simulate Greenhouse Tomato Growth and Development," *Horticulturae*, vol. 8, no. 11, p. 1021, Nov. 2022. [Online]. Available: https://www.mdpi.com/2311-7524/8/11/1021

[117] C. Stöckle, J. Kjelgaard, and G. Bellocchi, "Evaluation of estimated weather data for calculating Penman-Monteith reference crop evapotranspiration," *Irrigation Science*, vol. 23, no. 1, pp. 39–46, 2004.

[118] C. J. Willmott, "Some Comments on the Evaluation of Model Performance," *Bulletin of the American Meteorological Society*, vol. 63, no. 11, pp. 1309–1313, Nov. 1982.

[119] J. E. Nash and J. V. Sutcliffe, "River flow forecasting through conceptual models part I — A discussion of principles," *Journal of Hydrology*, vol. 10, no. 3, pp. 282–290, Apr. 1970.

[120] R. Diekmann and D. Weidemann, "Event enforcement in the context of the supervisory control theory," in *2013 18th International Conference on Methods & Models in Automation & Robotics (MMAR)*, Aug. 2013, pp. 783–788.

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU
Azione IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione”
e Azione IV.5 “Dottorati su tematiche Green”