University of Applied Sciences Hamburg
Faculty of Life Sciences

Master's Degree Course Pharmaceutical Biotechnology
Department of Biotechnology

Development of an interactive *Escherichia coli* fed-batch fermentation simulation

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science
by

Lena Sophia Kaletsch
[redacted]

Supervisors:

Prof. Dr. Gesine Cornelissen (HAW Hamburg)

Prof. Dr. Christian Kaiser (HAW Hamburg)

Hamburg, March 9th 2023

Abstract

Biofermentations are the highly labor-, material-, and time-intensive heart of the manufacturing processes of countless pharmaceutical products. Their setup and operation are exceedingly sophisticated and require the operator to complete thorough training. As this is associated with the aforementioned significant resource investments, simulations of such biofermentations can be developed to make this training more economically viable. New operators can thus be trained by operating a virtual process instead of a physical one. One such simulation is BIOSIM, which was originally conceived by Prof. Dr.-Ing. R. Luttmann at the beginning of the 21st century. For the past years, BIOSIM has been indispensable for training students of the Master's program Pharmaceutical Biotechnology at the University of Applied Sciences Hamburg in the operation of complex batch and fed-batch fermentation processes. Still, as new technology arises and former limitations of hardware subside, improvements to existing systems become feasible. As such, in this Master's thesis, a new simulation was developed to take over the role of BIOSIM. This new simulation application does not require specific hardware and includes additional functionalities, which eliminate the need to plot and evaluate the generated data with supplementary software. It was developed using MATLAB® App Designer and can be run as a standalone program. The application includes the same mathematical model of *Escherichia coli* K12 growth in a BIOSTAT ED 15 l bioreactor as BIOSIM in addition to virtual process control, both of which can be accessed through a graphical user interface. A comparison of the new simulation and BIOSIM shows that it is capable of generating the same results as BIOSIM while making substantial improvements regarding accessibility, required setup time, and handling of generated data. When compared to another available educational biofermentation simulation, the new application showed enhanced customizability regarding simulated experimental setup, process control, and visualization of generated data.

Biofermentationen sind der hochgradig arbeits-, material- und zeitintensive Kern der Herstellungsprozesse zahlloser pharmazeutischer Produkte. Ihr Setup und ihre Durchführung sind äußerst komplex und erfordern eine umfassende Ausbildung der eingesetzten Operatoren. Da dies mit den aufgezeigten Investitionen an Ressourcen verbunden ist, werden zunehmend Simulationen entwickelt, um diese Ausbildung ökonomisch tragbarer zu gestalten. Neue Operatoren können somit ausgebildet werden, indem sie virtuelle anstelle von realen Prozessen betreuen. Eine solche Simulation ist BIOSIM, welche ursprünglich von Prof. Dr.-Ing. R. Luttmann zu Beginn des 21. Jahrhunderts entwickelt wurde. In den vergangenen Jahren war BIOSIM ein essenzieller Bestandteil der Ausbildung der Studenten des Masterprogramms Pharmaceutical Biotechnology der Hochschule für Angewandte Wissenschaften Hamburg in der Durchführung von komplexen Batch und Fed-Batch Fermentationsprozessen. Neue Technologien und das Überkommen alter Hardwarelimitationen machen dennoch Verbesserungen von bestehenden Systemen möglich. Aus diesem Grund wurde in dieser Masterarbeit eine neue Simulation entwickelt,

um die Rolle BIOSIMs zu übernehmen. Diese neue Simulationsapplikation enthält zusätzliche Funktionalitäten für das Zeichnen und Auswerten der generierten Daten, welche die Notwendigkeit ergänzender Software erübrigen. Sie wurde mit dem MATLAB® App Designer konzipiert und kann als eigenständiges Programm ausgeführt werden. Zudem beinhaltet sie dasselbe mathematische Modell des Wachstums von *Escherichia coli* K12 in einem BIOSTAT ED 15 l Bioreaktor wie BIOSIM, zusammen mit virtueller Prozesssteuer und -regelung. Zugriff auf beides erfolgt durch eine graphische Nutzeroberfläche. Eine Gegenüberstellung von BIOSIM und der neuen Simulationsapplikation zeigt, dass diese vergleichbare Ergebnisse generieren, die neue Simulation jedoch maßgebliche Verbesserungen in Bezug auf Zugänglichkeit, erforderliche Setup Zeit und Umgang mit den generierten Daten mit sich bringt. Verglichen mit einer weiteren verfügbaren Trainingssoftware weist die neue Simulationsapplikation eine höhere Anpassbarkeit bezüglich des simulierten experimentellen Setups, der Prozesskontrolle und -steuerung sowie Visualisierung der generierten Daten auf.

Danksagung

Ich möchte mich zuallererst bei Prof. Dr. Gesine Cornelissen für die Gelegenheit bedanken, diese Arbeit im Labor für Bioprozessautomatisierung anfertigen zu dürfen. Auch für das Feedback und die Unterstützung während meiner Arbeit hier bin ich sehr dankbar. Ich danke auch Prof. Dr. Christian Kaiser für die Ideen und Anmerkungen zu meinem Projekt. Des Weiteren gilt mein Dank Dipl.-Ing. Ulrich Scheffler für die wertvolle Anleitung und Unterstützung während des gesamten Entwicklungsprozesses meiner Arbeit. Die Gelegenheiten, mich über mein Projekt hinaus weiterzubilden, weiß ich ebenfalls sehr zu schätzen.

Ich danke auch meinen Kommilitonen Javad Saeidi für seine Unterstützung bei regelungstechnischen Fragen, Phoebe Chan für das Korrekturlesen meiner Arbeit und ihr Feedback bezüglich meines Programms und Yuma Iff für die Idee, LaTeX eine Chance zu geben. Vielen Dank auch an Philipp Schmidt für seine Vorschläge und Anmerkungen zur Erstellung meiner Arbeit.

Ich möchte mich auch bei meinen Eltern, Beate und Stephan, und meiner Schwester Anna bedanken. Ich bin dankbar für eure bedingungslose Liebe und euren immerwährenden Beistand. Meinen Dank möchte ich ebenfalls meiner Oma, Hedwig, aussprechen. Du warst immer mein Vorbild und ich habe alle meine Leistungen auch für dich erbracht. In diesem Zuge auch vielen Dank an meine Freunde für die moralische Unterstützung, und dafür, dass ihr immer ein offenes Ohr für mich habt. I would like to specifically thank Niki and Natasha for always being at my side. If I had not talked to you so often over all these years, writing this thesis in English would never have been as easy.

Contents

Abbreviations

BPA	Bioprocess automation
DCU	Digital control unit
GUI	Graphical user interface
HAW	University of Applied Sciences Hamburg
MATLAB®	Matrix laboratory
ODE	Ordinary differential equation
OTS	Operating training simulator
PHB	Poly-β-hydroxybutyric acid
SISO	Single-input single-output
SPC	Setpoint controller

1 Introduction

The pharmaceutical industry employs complex fermentation processes to produce proteins or small molecules that find use as active ingredients in a multitude of medicinal products. These fermentations are highly labor-, material-, and time-intensive. This makes every fermentation a considerable investment and suboptimal process strategies lead to avoidable loss of the aforementioned resources [1]. Hence, fermentations are sought to be optimized in regard to improving product yields and saving costs.

In a biotechnological context, computational mathematical models have been developed as early as the 1970s to describe and characterize pharmaceutical processes [2]. Mechanistic models, which are based on generally accepted mathematical principles, could now be used to analyze generated process data and explain their inner workings [3]. These models were then employed to identify process bottlenecks and improve process performance regarding the previously mentioned factors. For instance, Mulchandani and colleagues were able to describe the production of poly-β-hydroxybutyric acid (PHB) with the help of a mechanistic model based on the involved inhibition kinetics in 1989 [4]. This model was subsequently developed further and used by Raje and Srivastava to evaluate nutrient feeding profiles and optimize PHB production in 1998 [5]. In 1994, Varma and Palsson were able to create a mechanistic kinetic model for wild-type *Escherichia coli* W3110 and coupled it to a predictive algorithm to describe its growth and by-product secretion in aerobic batch, fed-batch, and anaerobic batch cultures [6].

With new technological and scientific developments, computational limitations faded and models could become more and more complex. This made way for novel application pathways, such as process simulations, which refer to the execution of mathematical models repeatedly over time. With improved processing power, real-time simulations became feasible at the beginning of the 21st century [7]. In the age of big data, specifically in conjunction with artificial intelligence and machine learning, modeling and simulation approaches have become indispensable to understanding biotechnological processes in all their complexity [8].

Artificial intelligence can be used in big data analysis to fill in knowledge gaps and develop so-called data-driven models [9]. These data-driven models find fit functions that are able to represent the recorded input-output behavior of bioprocesses when no equations describing the system exist. Mechanistic and data-driven models can be combined into hybrid models, allowing for the circumvention of restrictions that might apply to either modeling approach. In the literature, mechanistic models are also referred to as white box models and data-driven models as black box models, whereas hybrid models are consequently described as gray box models. A mechanistic or white box model differentiates furthermore between structured and unstructured models, which refer to models that take into account different cellular compartments and those that do not, respectively. Apart from the modeling approach, another distinction should be made regarding the model type. If a model is static, it directly maps an input vector to an output vector using a mapping function. This mapping function is limited to static relations and can be derived from any modeling approach. If a model is dynamical, however, it incorporates ordinary differential equations (ODE), algebraic equations, partial differential equations, or combinations thereof. It starts from an initial state described by a set of initial values and constructs the development of the modeled system over time. In this case, the state of the system for the next point in time is calculated from its current state and current relevant input through a function. The output is calculated from the same state and input data through a different function. Again, either function may be derived from any modeling approach. The description of the system state is thus what differentiates between static and dynamical models [10].

Nowadays, modeling and simulation of bioprocesses are commonly used to design and optimize bioreactors, process control, and fermentation conditions and to predict the most effective ways of up- or downscaling such processes. For instance, Nagy created a data-driven dynamical model by using input-output data generated by a dynamical mechanistic kinetic model to train artificial neural networks in 2007. This data-driven model was consequently used to develop a nonlinear predictive control of a continuous yeast fermentation [11]. Hutmacher and Singh were able to demonstrate that computational fluid dynamics can be used to characterize flow fields in bioreactors for tissue engineering and consequently employed to design and optimize them in 2008 [12]. Goldrick and colleagues were able to simulate the industrial-scale fed-batch fermentation of penicillin in 2015. Their dynamical mechanistic model was able to deliver process data concurrent with industrial fermentation data and can thus be employed for the evaluation and improvement of the respective control strategies [13]. In 2019, Abunde and colleagues improved the performance of an industrial-scale alcoholic fermentation using sorghum by optimizing its control strategy with the help of a dynamical mechanistic model [14]. The same year, Tavasoli and colleagues utilized machine learning to generate a dynamical data-driven model for the production of alpha 1-antitrypsin in *Pichia pastoris*. This model was then used to optimize the feeding control strategy to improve product formation [15]. In 2020, Zhang and colleagues increased the maximum biomass and lipid yield of a *Rhodotorula glutinis* cellulosic ethanol wastewater fermentation by identifying optimal process parameters through a data-driven model [16]. Simultaneously, Culley and colleagues created a hybrid model of *Saccharomyces cerevisiae*, which was subsequently used for growth rate predictions [17]. Two years later, Du and colleagues successfully implemented a dynamical mechanistic model to predict and guide an industrial scale-up for the fermentation of docosahexaenoic acid [18].

As constructing a model that accurately represents a bioprocess requires high degrees of interdisciplinary knowledge, efforts have been made to make this field more accessible. In 2021, Hemmerich and colleagues published pyFOOMB, a Python package that enables users without a programming background to implement dynamic black box bioprocess models. It furthermore allows for the identification of global process parameters, such as maximum growth rate or yield coefficients, and can be used in the application areas mentioned earlier [19].

Simulations of bioprocesses do not only find use in *in silico* process planning and optimization but also at the heart of so-called digital twins. In the biopharmaceutical industry, digital twins are defined as virtual replicas of real-world bioprocesses and used for enhanced process monitoring and control. The virtual and real-world physical processes are perpetually exchanging information bidirectionally. This means that the real-world process information is sent to the virtual one, which then uses that information to quantify the status quo of the physical process and carry out predictive simulations regarding process developments. These predictions are then relayed back to the physical process. As such, potential issues can be identified before they occur and optimization actions can be developed and executed while the process is running. Acting as a closed-loop model-based controller is thus what defines a digital twin [2, 20]. Digital twins have already been implemented on an industrial scale in several asset manufacturing processes. For example, digital twins are used in the automotive industry to forecast failure rates during the design and control stages of new vehicles [21]. GlaxoSmithKline, Siemens, and Atos have recently started the development of a digital twin for a vaccine production process [22]. Though digital twins are not currently employed in industrial-scale pharmaceutical manufacturing processes, they are generally thought to represent the future of such processes, especially in the context of the now emerging industry 4.0, and implementation framework propositions already exist [23].

Furthermore, models and simulations can find use in educational contexts. Operating training simulators (OTS) are frequently employed in industries such as the aerospace or naval industry [24]. Though currently such OTSs are sparsely used in the pharmaceutical industry, they put forth a range of benefits [25]. As previously mentioned, biopharmaceutical manufacturing processes are highly resource-demanding. Since new bioprocess operators naturally make mistakes, extensive training becomes necessary to avoid process failure and subsequently save time and resources. This requires test runs of the bioprocess. To shrink the costs associated with these test runs, simulations of biofermentations can be employed, as no material is used up and no setup is required, which saves time and labor as well [26]. Furthermore, extreme process conditions can be simulated to improve the performance of even senior operators in emergency situations without the need for a real biofermentation [27]. Gerlach and colleagues developed a model-based OTS simulating bioreactor operations for the production of ethanol in *Saccharomyces cerevisiae* and a recombinant protein in *Escherichia coli* [28]. When comparing the performance of bioprocess operators that had been trained with the OTS to that of those who had not been trained with the OTS, the trained group exhibited improved proficiency in operating a real-world biofermentation [25]. A previous version of this OTS is commercially available under the name “BioProzessTrainer” as part of the textbook “Praxis der Bioprozesstechnik mit virtuellem Praktikum” by Hass and Pörtner [29].

OTSs contain dynamical mathematical models that describe all relevant processes that occur during a fermentation. The results generated by these models deliver the information that would usually be generated through off-, at-, and in-line measurements of a physical biofermentation. As the equations making up the mathematical model are able to take input from the process control, such as the input given by a controller, new operators can familiarize themselves with the process response to the process control. Operating a simulated fermentation hence can be used as training for the operation of a real fermentation [25, 24].

Another such training simulation program, BIOSIM, was established by Prof. Dr.-Ing. R. Luttmann at the beginning of the 21st century at the University of Applied Sciences Hamburg (HAW). This simulation was conceived to run on the operating system RTOS-UH and is connected to an external digital control unit (DCU, B. Braun Biotech International/Sartorius Stedim Biotech AG, Goettingen, Germany) through the process control system UBICON (esd electronic system design GmbH, Hannover, Germany). BIOSIM consists of parameters, variables, and equations that describe the physicochemical and biological processes occurring during the fermentation of *E. coli* K12 in the BIOSTAT ED 15 l bioreactor (B. Braun Biotech International/Sartorius Stedim Biotech AG, Goettingen, Germany). The operator interacts only with the DCU to relay process control information to BIOSIM, and thus operates the simulation like a real fermentation. This setup is illustrated in figure 1.

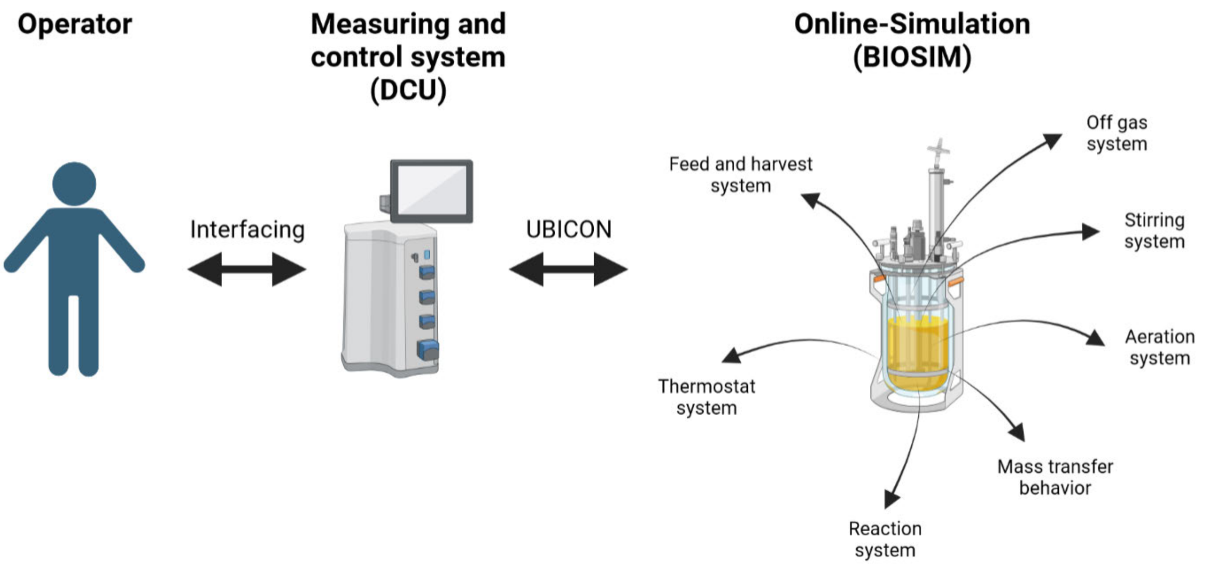


Figure 1: **Setup of the fermentation simulation that is the precursor of the program developed in this work.** BIOSIM, UBICON, and the DCU are to be converted into one virtual standalone application. Created with BioRender based on the BIOSIM model conceived by Prof. Luttmann [30].

In the last years, BIOSIM has been indispensable for the Bioprocess Automation (BPA) special course that is part of the Master's program Pharmaceutical Biotechnology offered by the HAW. However, as the software and hardware building its foundation age, new technology arises capable of improving upon older systems. As such, the aim of this work is to develop a new simulation based on recent technological advancements. An application is established consisting of a graphical user interface (GUI) taking over the responsibilities of the DCU and the visualization of the generated process data, the dynamical mechanistic mathematical model conceived by Prof. Luttmann, and the process control comprising the controllers needed for the operation and automation of the fermentation.

MATLAB® and its built-in App Designer tool were chosen to develop this application. While it is possible to develop such an application in other programming languages such as Python, MATLAB® was preferred here as it offers exceptionally good documentation and is easily accessible to users without a programming background. Moreover, MATLAB® is taught in courses of the Master's program Pharmaceutical Biotechnology. Thus, programming the app in MATLAB® allows the students to easily understand the application's source code and even make alterations if necessary. Furthermore, MATLAB® is employed in industry and research for data analytics, signal and image processing, control design, development of algorithms, and creation of system models [31, 32]. It is widely used in the pharmaceutical and biotech industry specifically and finds application for process engineering purposes, creating models and simulations for drug discovery, and development and hybridization of multiple different data streams [33]. As such, MATLAB® is well-suited to accomplish the task at hand.

2 Theoretical background

2.1 Design of the simulation

For the simulation that is to be established in this work, a modular approach was chosen. This allows for the exchange of specific parts through different ones, such as a different mathematical model or a different process control strategy. The theoretical design is illustrated in figure 2.

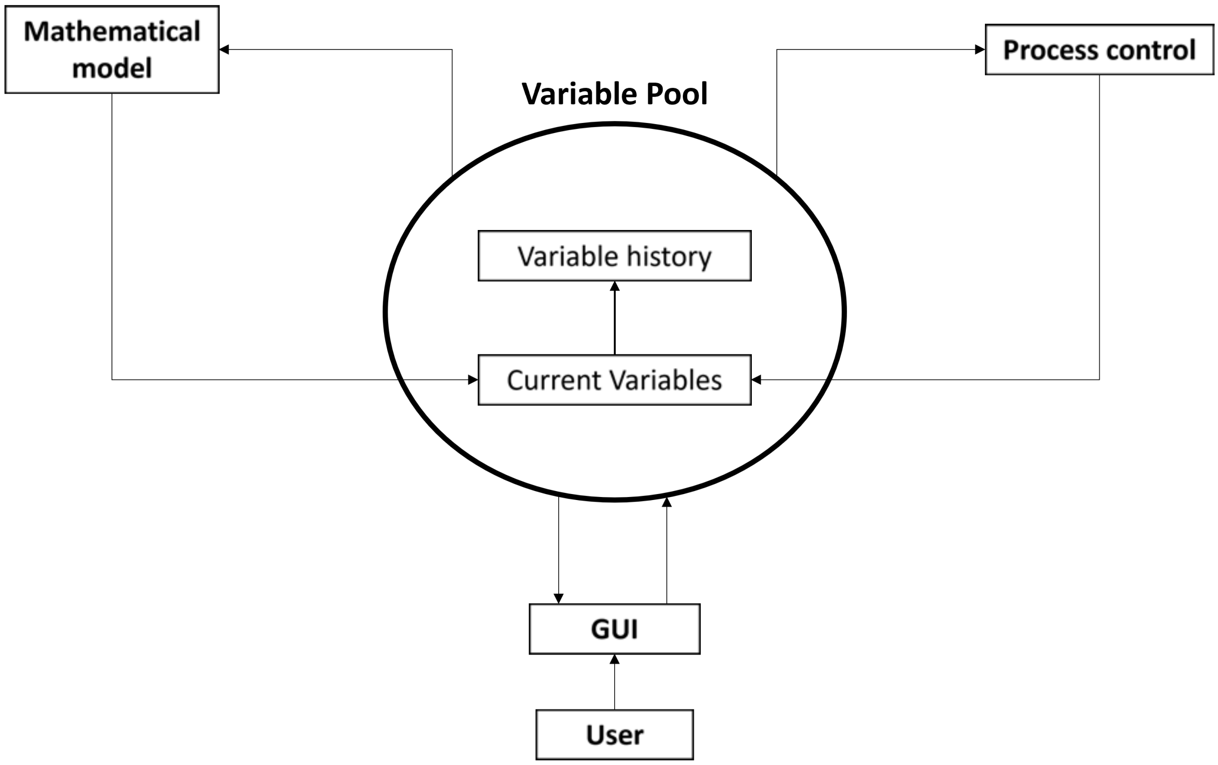


Figure 2: **Schematic overview of the design of the MATLAB®-based simulation.** The central element is the variable pool, which contains the currently valid process variables and archives them in the variable history. The fermentation-specific variables are updated through the mathematical model whereas the process control-specific variables are updated through the process control containing the active controllers. The user interacts with the simulation through the GUI.

The core element of the simulation is the variable pool. This variable pool contains all inputs and outputs of the various operations that take place in the application, namely the process control-related variables and those related to the simulation of the bioprocess itself. The latter includes the bioreactor environment and all its biological and physicochemical processes.

The current variables are the latest inputs and outputs as calculated by the mathematical model and the process control. They are updated when the user changes controller setpoints or manually regulates the process, i.e. by changing the stirrer speed, and when the mathematical model calculates the values for the next point in the simulated process time. Most values are overwritten once they are updated, though a selection, such as cell and substrate concentrations, temperature, or p_{O2}, is archived in the variable history. This allows for evaluation by the user and graphical depiction within the app in the form of a plot. The simulated process thus may be analyzed and monitored comparably to data generated by an actual biofermentation process.

The mathematical model consists of algebraic and differential equations describing the physicochemical and biological processes within the bioreactor, such as cell growth, pH, or temperature courses. A few equations receive their input solely from the variable history or the current variables given by the process control in addition to constant parameters, such as the gas constant R. Examples for this are the molar fraction of oxygen at the reactor inlet x_{OGin} or the iterative calculation of the pH. However, most equations also require the outputs of previous equations of the mathematical model. Thus, they depend on current variable values to calculate other current variable values. For example, the maximum specific growth rates for the different substrates μ_{imax} depend on the current pH, which is calculated earlier in the model. Two of the sub-models that make up the mathematical model are explained in detail in chapter 2.2.

The process control contains setpoint controllers (SPC) that effectuate instrumentation setpoints chosen by the operator and single-input single-output (SISO) controllers responsible for maintaining consistent environmental conditions within the bioreactor. The operator can feed setpoints to the variable pool and switch between manual and automatic process control modes at any time during the course of the simulation. If a manual control mode is selected, instrumentation setpoints, such as the stirrer speed or feed rate, are implemented via SPCs. If an automatic control mode is selected, the bioreactor instrumentation is regulated by SISO controllers. This is, for example, the case when the p_{O2}-agitation controller is active and the stirrer speed is changed automatically to keep the p_{O2} constant. The process control is described in detail in chapter 3.4.

The interface between the variable pool and the user is given by the GUI. The GUI allows the user to make the aforementioned changes to the current variables of the process control in the variable pool. It also allows for the adjustment of parameters of the mathematical model, such as the initial cell concentration, and displays current values for both the process control and the mathematical model. It furthermore visualizes them in the form of a plot.

2.2 Mathematical model and equations

The mathematical model that builds the basis for the simulation that is to be established in this work was conceived by Prof. Luttmann for the BIOSIM system [30]. It comprises a full description of the technical and biological bioreactor system including the physicochemical and biological behaviors as well as the individual in-line, at-line, and on-line measurements.

The engineering aspects are made up of the thermostat, mixing, aeration, gas removal, and mass transport behavior. The biological reaction processes include growth, substrate and oxygen uptake, and carbon dioxide as well as product formation. These systems interact through pH, viscosity, and foam formation. Additionally, feed, harvest, and pH titration systems are implemented. The sub-models are illustrated in figure 3.

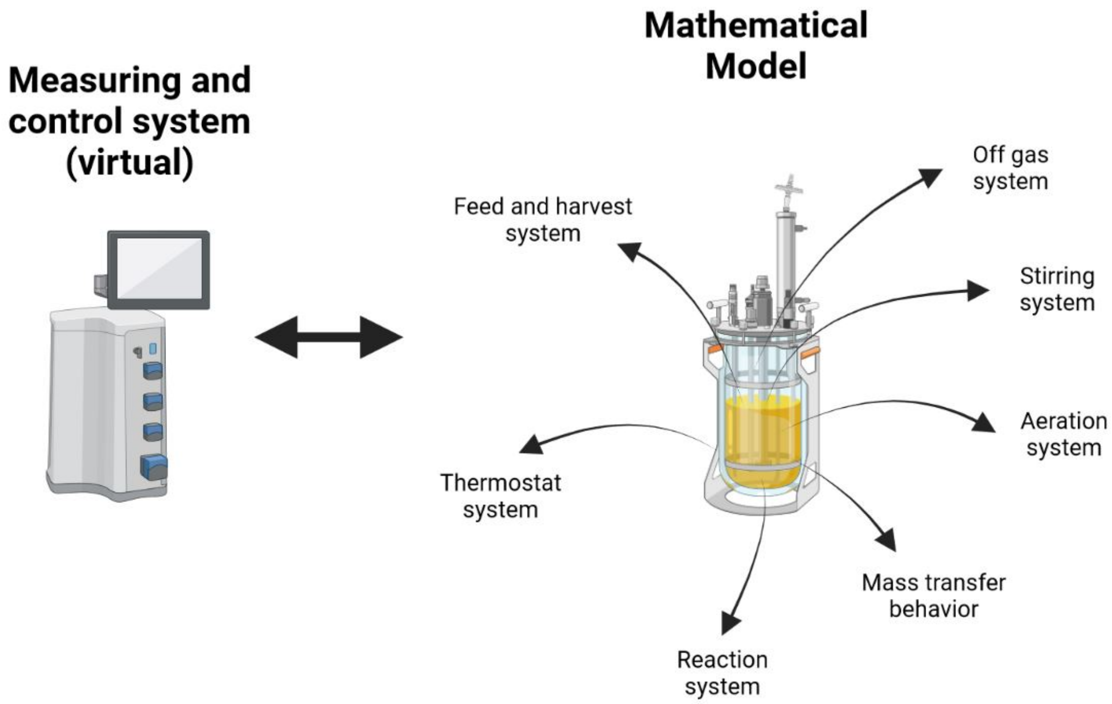


Figure 3: **Overview over the sub-models in the bioreactor system that are described in the mathematical model.** Created with BioRender based on the BIOSIM model conceived by Prof. Luttmann [30].

How the equations of the model are conceived additionally needs to account for real-time control by the operator. This control system was realized via connection to an external DCU for BIOSIM. Thus, no equations for controllers were originally derived for this control system, only the reactions of the different systems to a controller output are described. For example, the BIOSIM mathematical model describes the processes taking place in the temperature control system upon receiving a “heating” or “cooling” (y_H or y_C) signal from a controller. How this signal is calculated by the controller is not described, as it is given by an external control unit.

As the mathematical model is complex and consists of more than one hundred algebraic and about thirty differential equations, only two different models are described in detail in this work. To give a varied impression of how the model works and what it looks like, one physical system, the thermostat system, and one biological system, the reaction system, were chosen. The remaining sub-model equations can be found in the BIOSIM manual or its accompanying publication [30].

2.2.1 The thermostat system

The thermostat system is tasked with regulating the temperature in the liquid phase of the bioreactor by adjusting the temperature of the water circulating through its surrounding double jacket. The heat exchanger for cooling uses water, the heat exchanger for heating uses steam in the case of automatic control and electricity in the case of manual operation.

It receives the signals "HEATING" or "COOLING" if the temperature is controlled manually by the operator. In this case, heating or cooling occurs at their maximum possible outputs (P_{Hmax} in case of heating and $\dot{m}_{Cmax}$ in case of cooling, capitalization is used to differentiate these signals from those given by the controller). In the case of automatic control, the temperature of the liquid phase is regulated by a cascade control operating at split range. Here, the output of the master controller y_{DJ} (the setpoint for the temperature of the double jacket) spans a range of [-1,+1]. As y_{DJ} is operating at split range, this means that for $y_{DJ} < 0$ the reactor liquid will be cooled and for $y_{DJ} > 0$ the reactor liquid will be heated up. In the case of heating, y_{DJ} is given to the heating slave controller, which in turn calculates y_H, a continuous control variable spanning a range of [0,1]. At $y_H = +1$, it is delimited by the maximum possible steam flux rate $\dot{m}_{Hmax}$. In the case of cooling, y_{DJ} is given to the cooling slave controller, which in turn calculates y_C, a continuous control variable spanning a range of [-1,0]. At y_C = -1, it is delimited by the maximum possible cooling water flow rate $\dot{m}_{Cmax}$. To calculate the actual values of the control elements, these maximum outputs are multiplied by their respective control variable. Hence, if $y_H = 0.5$, the heating power will be half of its maximum possible output and vice versa if y_C = -0.5 in case of cooling. The process control is explained further in chapter 3.4.

The bioreactor thermostat system consists of the liquid phase in the reactor, the surrounding double jacket, and the supply unit that is connected to the double jacket by way of piping. As described by Prof. Luttmann, this model assumes two simplifications: all subsystems are ideally mixed and no heat is lost in the piping. Because of these simplifications, ordinary differential equations can be used instead of partial ones, as only the time dependency remains.

Thus, the subsystems thermostat heating (Th), thermostat cooling (Tc), cooling water (C), double jacket (D), and liquid phase reaction (L) are of importance in this submodel. The reactor walls distribute heat capacities proportionately to subsystems D and L. A schematic overview of the components, subsystems, and variables describing the thermostat system is given in figure 4.

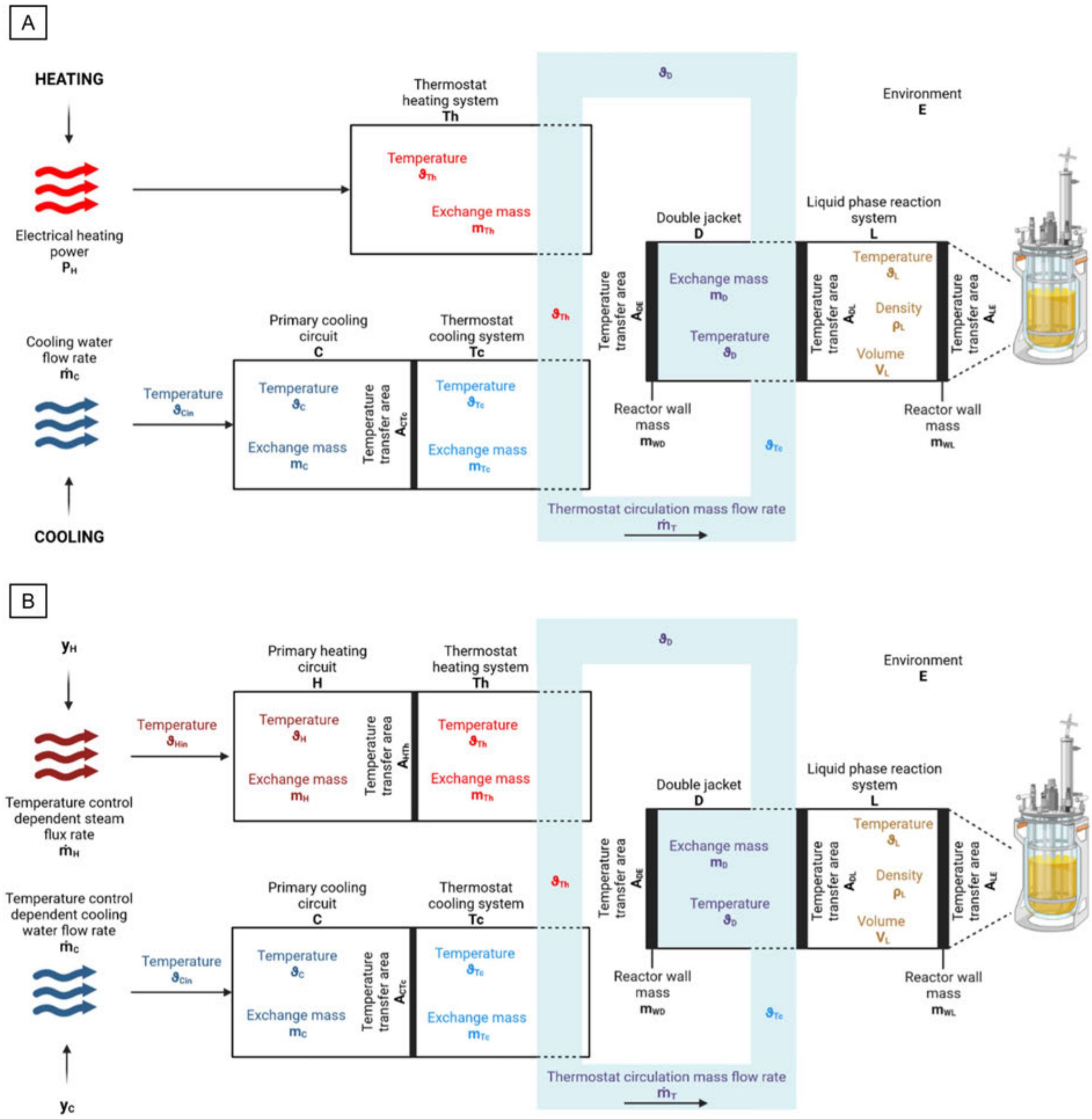


Figure 4: **Schematic overview over the components and variables defining the thermostat system of BIOSIM as described by Prof. Luttmann.** Electrical heating is employed when the heating signal is given manually by the operator (A) whereas steam heating is employed when the temperature is regulated by the temperature cascade control operating at split range (B). In each case, cooling is achieved through cooling water flow. Created with BioRender based on the BIOSIM model conceived by Prof. Luttmann [30].

The temperature is controlled in a pressurized closed circuit. Water is pumped through this circuit into the subsystems K (K = Th,Tc,D) with the mass flow $\dot{m}_T$. The reciprocal mean residence time D_K [h^{-1}] in subsystem K can thus be described as

$$D_K = \frac{\dot{m}_T}{m_K} \tag{1}$$

with

$\dot{m}_T :=$ thermostat circulation mass flow rate [g·h^{-1}]

$m_K :=$ exchange mass in subsystem K [g]

and the equivalent for the time-dependent cooling water system D$_C$ [h^{-1}]

$$D_C\left(t\right) = \frac{\dot{m}_C\left(t\right)}{m_C} \tag{2}$$

with

$\dot{m}_C :=$ time- and temperature control-dependent cooling water flow rate [g·h^{-1}]

$m_C :=$ exchange mass in primary cooling circuit [g]

can be defined.

The heat transmission coefficient k$_{JK}$ [W·m^{-2}·K^{-1}]

$$k_{JK} = \frac{1}{\frac{1}{\alpha_J} + \frac{\delta_{JK}}{\lambda_{JK}} + \frac{1}{\alpha_K}} \tag{3}$$

with

α_I := heat transfer coefficient in subsystem I; I = J,K [W·m^{-2}·K^{-1}]

$\delta_{JK} :=$ wall thickness between J and K; J,K = C,Tc,D,L,E [m]

$\lambda_{JK} :=$ thermal conductivity of the wall between J and K [W·m^{-1}·K^{-1}]

describes classical heat transfer through a plane-shaped wall. This is a simplification assumed for the reactor geometry here.

Consequently, the overall heat transmission rate R$_{JK}$ [K·W^{-1}]

$$R_{JK} = \frac{1}{k_{JK} \cdot A_{JK}} \tag{4}$$

with

$A_{JK} :=$ heat transfer area between J and K; J,K = C,Tc,D,L,E [m^2]

and the thermal transport time constant τ_{JK} [h]

$$\tau_{JK} = R_{JK} \cdot\ C_J \tag{5}$$

with

$C_J :=$ volumetric heat capacity in subsystem J, J = C,Th,Tc,D,L [W·h·K^{-1}]

can be defined.

C_J, the volumetric heat capacity in subsystems C, Th, and Tc, is calculated from

$$C_J = m_J \cdot c_{H2O} \tag{6}$$

with

c_{H2O} := specific heat capacity of water [W·h·kg^{-1}·K^{-1}]

m_J := mass of subsystem J; J = C,Th,Tc [kg].

C_D [W·h·K^{-1}] is given by

$$C_D = m_D \cdot c_{H2O} + m_{WD} \cdot c_w \tag{7}$$

with

m_{WI} := reactor wall fraction in subsystem I; I = D,L [kg]

c_w := specific heat capacity of the wall [W·h·kg^{-1}·K^{-1}]

and determined by the mass of water present in subsystem D.

The heat capacity of the reactor wall is modeled in fractions allotted proportionately to the double jacket D and the liquid phase in the reactor L. The time-dependent heat capacity of the liquid phase C_L [W·h·K^{-1}] can hence be described as

$$C_L(t) = \rho_L(t) \cdot V_L(t) \cdot c_{H2O} + m_{WL} \cdot c_w \tag{8}$$

with

ρ_L := time-dependent density of liquid phase in reactor [g·l^{-1}]

V_L := time-dependent liquid volume in reactor [l].

Therefore, the differential equations of the temperature balances in the thermostat system can be described as follows:

The time-dependent temperature change of the thermostat heating system Th [°C·h^{-1}]

$$\dot{\vartheta}_{Th}(t) = -D_{Th} \cdot \vartheta_{Th}(t) + D_{Th} \cdot \vartheta_D(t) + \frac{P_H(t)}{C_{Th}} \tag{9}$$

with

ϑ_{Th} := time-dependent temperature in thermostat heating system [°C]

P_H := time-dependent required electrical heating power [W]

ϑ_D := time-dependent temperature in double jacket [°C]

includes only the convective heat flux and the heating power P_H as the controlling variable in the case of manual control.

In the case of automatic control, the reactor is steam heated. Steam heating involves complex thermodynamical processes; in addition to time- and place-dependency, mass transitions in vaporous media and condensation processes occur. The energy balance in the heating space [W] can be described as

$$\dot{Q}_H(t) = \frac{d}{dt}(c_H(t)\cdot\ m_H(t)\cdot\vartheta_H(t)) = \dot{m}_H(t)\cdot(c_{Hin}(t)\cdot\vartheta_{Hin}(t) - c_H(t)\cdot\vartheta_H(t)) \tag{10}$$

with

Q_H := time-dependent heat change in heating system [J]

m_H := time-dependent mass of heating medium in heating system [kg]

$\dot{m}_H$:= time-dependent mass flow of heat operated by cascade control [kg·h^{-1}]

ϑ_H := time-dependent temperature in heating system [°C]

C_H := time-dependent specific heat capacity in heating system [W·h·kg^{-1}·K^{-1}]

ϑ_{Hin} := time-dependent temperature of steam at influx [°C]

C_{Hin} := time-dependent specific heat capacity in steam at influx [W·h·kg^{-1}·K^{-1}].

The incoming steam and the mixture of steam and condensate likewise possess temperature- and thereby time-dependent heat capacities and densities. In the following equations, it is assumed that the incoming steam will be completely removed as condensate by a steam trap at ϑ_H < 100 °C. Hence, the evaporation energy remains in the system.

In the transfer heat flux of the double jacket heating up system $\dot{Q}_{HTh}$ [W]

$$\begin{aligned}\dot{Q}_{HTh}(t) &= \dot{m}_H(t)\cdot(c_{Hin}(t)\cdot\vartheta_{Hin}(t) - c_H(t)\cdot\vartheta_H(t)) - \frac{d}{dt}(Q_H(t))\\ &= \dot{m}_H(t)\cdot[c_{H2O}\cdot(\vartheta_{Hin}(t) - \vartheta_H(t)) + \Delta\mathrm{h}_v]\end{aligned} \tag{11}$$

this heat source is taken into account within the completely condensed steam through the specific evaporation enthalpy $\Delta\mathrm{h}_v$ [kJ·kg^{-1}].

The heat radiating from the heating system is taken up through

$$\dot{Q}_{HTh}(t) = k_{HTh}(t) \cdot A_{HTh} \cdot (\vartheta_H(t) - \vartheta_{Th}(t)) \tag{12}$$

by the heating up system.

The heat transfer coefficient k_{HTh} [kJ·m^{-2}] is temperature-dependent as well through α_H [W·m^{-2}·K^{-1}]. In the following, it is considered constant.

After the introduction of a normalization parameter, the maximum possible mass of the heating system (when completely filled with water) m_{Hmax} [kg]

$$m_{Hmax} = V_H \cdot \rho_{H2O} \tag{13}$$

four system parameters can be derived normed to it:

The volumetric heat capacity of the heating system C_{Hmax} [W·h·K^{-1}]

$$C_{Hmax} = m_{Hmax} \cdot c_{H2O} \tag{14}$$

the time constant of the primary heating circuit τ_{HTh} [h]

$$\tau_{HTh} = \frac{m_{Hmax} \cdot c_{H2O}}{k_{HTh} \cdot A_{HTh}} \tag{15}$$

with

A_{HTh} := Transfer area between heating and heating-up circuit [m^2],

the maximum evaporation heat Q_v [kJ]

$$Q_v = \Delta \mathrm{h}_v \cdot m_{Hmax} \tag{16}$$

and the heating flux rate operated by the temperature controller D_H [h^{-1}]

$$D_H(t) = \frac{\dot{m}_H(t)}{m_{Hmax}}. \tag{17}$$

$\dot{m}_H$ is ruled by the control variable y_H, as it is derived by multiplying y_H with the maximum possible steam flux $\dot{m}_{Hmax}$. y_{DJ} and y_H are proportionally related through the gain of the heating slave P controller.

After eliminating the unknown temperature ϑ_H from equations 11 and 12, the wanted transfer heat flux $\dot{Q}_{HTh}$ is given by

$$\dot{Q}_{HTh}(t) = \frac{D_H(t) \cdot (C_{Hmax} \cdot (\vartheta_{Hin}(t) - \vartheta_{Th}(t)) + Q_v)}{1 + T_{HTh} \cdot D_H(t)} \tag{18}$$

as a function of the control parameters D_H and ϑ_{Hin} as well as the temperature of the heating up system ϑ_{Th}.

Finally, the temperature of the heating-up system can be calculated with

$$\dot{\vartheta}_{Th}(t) = -D_{Th} \cdot \vartheta_{Th}(t) + D_{Th} \cdot \vartheta_D(t) + \frac{\dot{Q}_{HTh}(t)}{C_{Th}}. \tag{19}$$

The time-dependent change in temperature of the cooling water system C

$$\dot{\vartheta}_c(t) = -\left(D_C(t) + \frac{1}{T_{CTc}}\right) \cdot \vartheta_C(t) + D_C(t) \cdot \vartheta_{Cin}(t) + \frac{\vartheta_{Tc}(t)}{T_{CTc}} \tag{20}$$

with

ϑ_C := time-dependent temperature of cooling water system [°C]

ϑ_{Cin} := time-dependent temperature of cooling water feed [°C]

is the primary side of the cooling heat exchanger. The secondary side is the thermostat cooling system T_C, whose time-dependent change in temperature [°C·h^{-1}] is described as

$$\dot{\vartheta}_{Tc}(t) = -\left(D_{Tc} + \frac{1}{T_{TcC}}\right) \cdot \vartheta_{Tc}(t) + D_{Tc} \cdot \vartheta_{Th}(t) + \frac{\vartheta_C(t)}{T_{TcC}} \tag{21}$$

with

ϑ_{Tc} := time-dependent temperature of thermostat cooling system [°C],

and connected to the heating system by means of convection.

The time-dependent temperature change of the double jacket system D [°C·h^{-1}] is described as

$$\dot{\vartheta}_D(t) = -\left(D_D + \frac{1}{T_{DL}} + \frac{1}{T_{DE}}\right) \cdot \vartheta_D(t) + D_D \cdot \vartheta_{Tc}(t) + \frac{\vartheta_L(t)}{T_{DL}} + \frac{\vartheta_E(t)}{T_{DE}} \tag{22}$$

with

ϑ_L := time-dependent temperature of liquid phase in reactor [°C]

ϑ_E := time-dependent temperature of reactor environment [°C].

The double jacket system is connected to the cooling system by means of convection and is involved in heat exchange with the liquid phase L and the reactor environment E.

The time dependent temperature in the liquid phase of the reactor L is given by

$$\dot{\vartheta}_L(t) = -\left(\frac{1}{T_{LD}(t)} + \frac{1}{T_{LE}(t)}\right) \cdot \vartheta_L(t) + \frac{\vartheta_L(t)}{T_{LD}(t)} + \frac{\vartheta_E(t)}{T_{LE}(t)} + \frac{\dot{Q}_{St}(t) + \dot{Q}_M(t)}{C_L(t)} \tag{23}$$

with

$\dot{Q}_{St}$:= time-dependent thermal power of stirrer [W]

$\dot{Q}_M$:= time-dependent thermal power of microorganisms [W].

The liquid phase is involved in heat exchange with the double jacket D and environment E through the reactor.

The liquid phase receives heat produced by the stirrer $\dot{Q}_{St}$

$$\dot{Q}_{St}(t) = K_{HSt} \cdot\ V_L(t) \cdot\ {N_{St}}^3(t) \tag{24}$$

with

K_{HSt} := proportional gain of stirrer heat generation [W·min^3·l^{-1}]

N_{St} := time-dependent agitation speed [min^{-1}]

and by the microorganisms $\dot{Q}_M$

$$\dot{Q}_M = K_{HM} \cdot\ V_L(t) \cdot\ OUR(t) \tag{25}$$

with

K_{HM} := proportional gain of microbial heat generation [W·h·g^{-1}]

OUR := time-dependent microbial oxygen uptake rate [g·l^{-1}·h^{-1}].

For better illustration, the thermostat system can be represented by an electric circuit as illustrated in figure 5. Figure 5B also shows how the reactor thermostat model and the split range cascade control mentioned at the beginning of this chapter are connected. The jacket entry temperature ϑ_{Tc}, which equals the temperature in the double jacket ϑ_D, and the liquid phase temperature ϑ_L are calculated in the model after every time interval and fed as actual values to the cascade control. The master controller compares ϑ_L to its setpoint. The output of the master controller y_{DJ} is then either fed to the heating or the cooling slave controller, which compare y_{DJ} to ϑ_D. The slave controllers then either give their output ($y_C < 0$) to the cooling water valve where resistance R_C applies or the steam flux ($y_H > 0$). A detailed depiction of these controllers is given in figure 13.

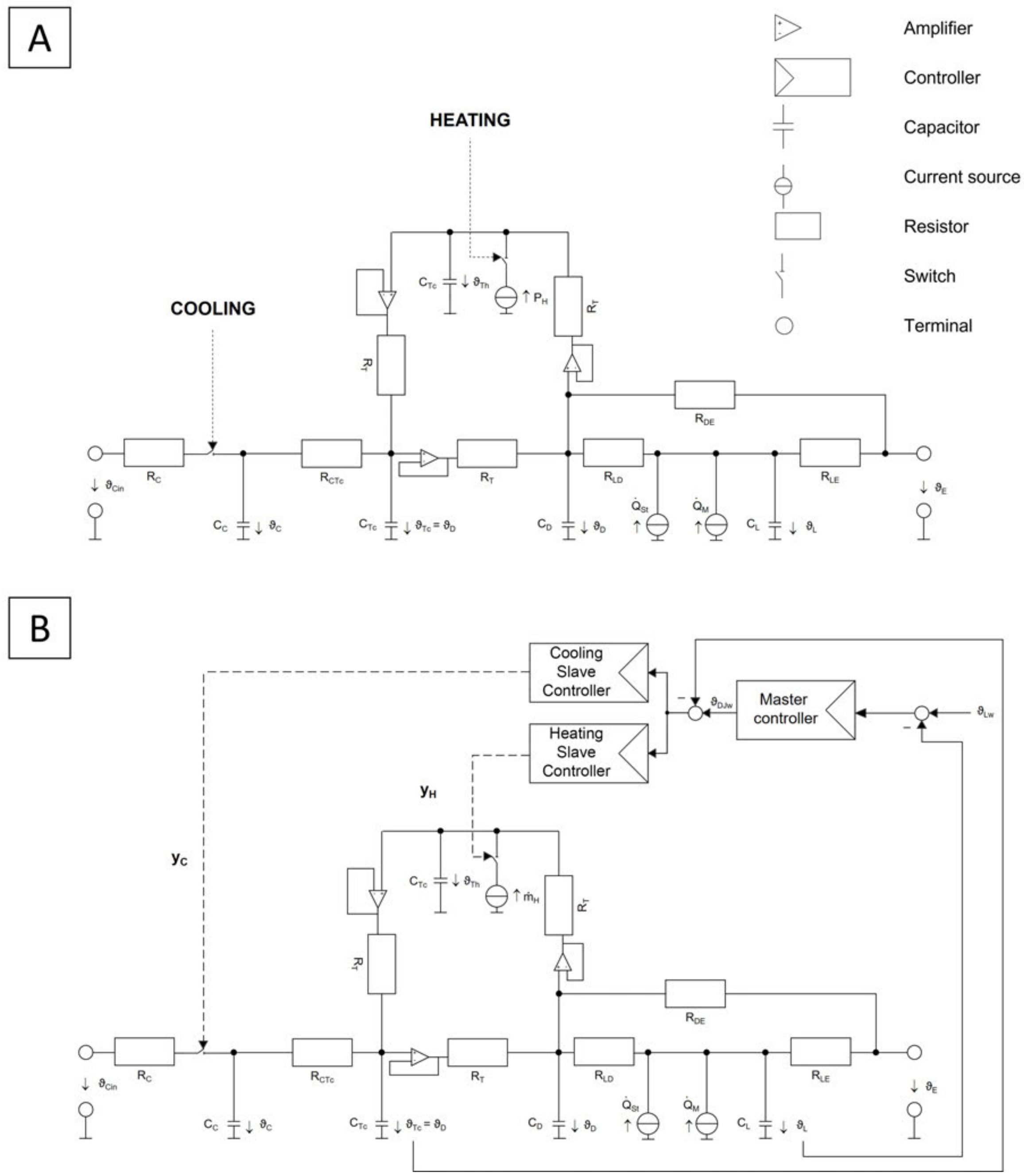


Figure 5: **Electric circuits describing the thermostat system under manual operation (A) or controlled by a cascade control operating at split range (B).** The current sources demonstrate heat influx while the terminals indicate heat loss. Storage of heat is indicated by capacitors. Resistance to temperature change is designated by resistors. Adapted from the BIOSIM manual written by Prof. Luttmann [30].

The time-dependent cooling water entry heat resistance R_C $[K \cdot W^{-1}]$ is given by

$$R_C(t) = \frac{1}{\dot{m}_C(t) \cdot c_{H2O}} \tag{26}$$

and describes the convective primary cooling water flow. In contrast to that, the thermostat circulation heat resistance R_T [K·W^{-1}] is given by

$$R_T = \frac{1}{\dot{m}_T \cdot c_{H2O}} \tag{27}$$

and describes the convective heat flux in the thermostat system. As the convective streams do not give a feedback, they are depicted as trap amplifiers.

In the simulation at hand, the thermostat system described here was implemented by assuming the heat exchange system differential equations 9, 10, 19, 20, and 21 are quasi-stationary in contrast to the exchange processes in the double jacket and liquid phase described in the differential equations 22 and 23.

With the introduction of the dimensionless cooling and heating parameters

$$\varphi_{HT}(t) = \frac{\dot{m}_H(t)}{\dot{m}_T}, \tag{28}$$

$$\varphi_{HTh}(t) = D_H(t) \cdot \ T_{HTh}, \tag{29}$$

$$\varphi_{CTc}(t) = D_C(t) \cdot \ T_{CTc}, \tag{30}$$

and

$$\varphi_{TcC} = D_{Tc} \cdot \ T_{TcC}, \tag{31}$$

the exit temperature of the heating up system ϑ_{Th} can be calculated. In case of an electrically heated bioreactor, ϑ_{Th} is calculated as

$$\vartheta_{Th}(t) = \vartheta_D(t) + R_T \cdot \ P_H(t) \tag{32}$$

and in the case of steam heating as

$$\vartheta_{Th}(t) = \frac{(1+\varphi_{HTh}(t)) \cdot C_{Hmax} \cdot \vartheta_D(t) + \varphi_{HT}(t) \cdot (C_{Hmax} \cdot \vartheta_{Hin}(t) + Q_v)}{C_{Hmax} \cdot (1+\varphi_{HTh}(t) + \varphi_{HT}(t))} \tag{33}$$

in which ϑ_{Th} is controlled via the steam flux $\dot{m}_H$ and the steam entrance temperature ϑ_{Hin}.

The equation of the cooling system exit temperature ϑ_{Tc}

$$\vartheta_{Tc}(t) = \varphi_{CTc}(t) \cdot \vartheta_{Cin}(t) + \frac{(1+\varphi_{CTc}(t)) \cdot \varphi_{TcC} \cdot \vartheta_{Th}(t)}{(1+\varphi_{TcC}) \cdot (1+\varphi_{CTc}(t)) - 1}, \tag{34}$$

which finds use as the double jacket temperature actual value ϑ_D

$$\vartheta_D(t) = \vartheta_{Tc}(t) \tag{35}$$

fed to the slave controllers, and the equation of the cooling water exit temperature ϑ_C

$$\vartheta_C(t) = \frac{\varphi_{CTc}(t) \cdot \vartheta_{Cin}(t) + \vartheta_{Tc}(t)}{1 + \varphi_{CTc}(t)} \tag{36}$$

are valid for both the electrically and the steam-heated bioreactor.

2.2.2 The reaction model

As the temperature control system is a purely technical one, the reaction model of the microorganism *Escherichia coli* K12 is used to illustrate a biological system in the following chapter.

The reaction model is based on the bottleneck principle. Thus, the growth rate μ is governed by a single enzymatic catalysis. As zero-order reactions in the substrate mass balances should be avoided, the substrate uptake for maintenance purposes is substituted by adding $q_{X/Xm}$ in the cell mass balance equation. This means that, in the model at hand, the cell obtains its required maintenance energy from storage substances within the cell and is thus not dependent on the available substrate concentration in the liquid phase. This makes the generation of maintenance energy time-invariant. It will also cause the growth rate to become negative if either no substrate or no oxygen is available, which will in turn cause cell death. Furthermore, the microorganism whose growth is simulated in this work is capable of utilizing two other substrates in addition to glucose (S1), namely glycerol (S2) and acetate (S3), the latter of which is also the product.

The time-dependent cell-specific growth rate $q_{X/X}$ [h^{-1}] can hence be described as

$$q_{X/X}(t) = \mu(t) = min\left\{\mu_{Sgr}(t), \mu_{Ogr}(t)\right\} - q_{X/Xm} \leq \mu_{1max} \tag{37}$$

with

μ := time-dependent observed cell-specific growth rate [h^{-1}]

μ_{1max} := maximum possible cell-specific growth rate on glucose [h^{-1}]

μ_{Sgr} := time-dependent substrate-determined cell-specific growth rate [h^{-1}]

μ_{Ogr} := time-dependent oxygen-determined cell-specific growth rate [h^{-1}]

$q_{X/Xm}$:= cell-specific (death) maintenance rate [h^{-1}].

$q_{X/X}$ is, consequently, controlled by the substrate or oxygen supply, which serves as the rate-determining step, and is limited by the maximum specific growth rate on glucose μ_{1max}.

An overview of the multi-substrate reactions taking place in this mathematical model is given in figure 6.

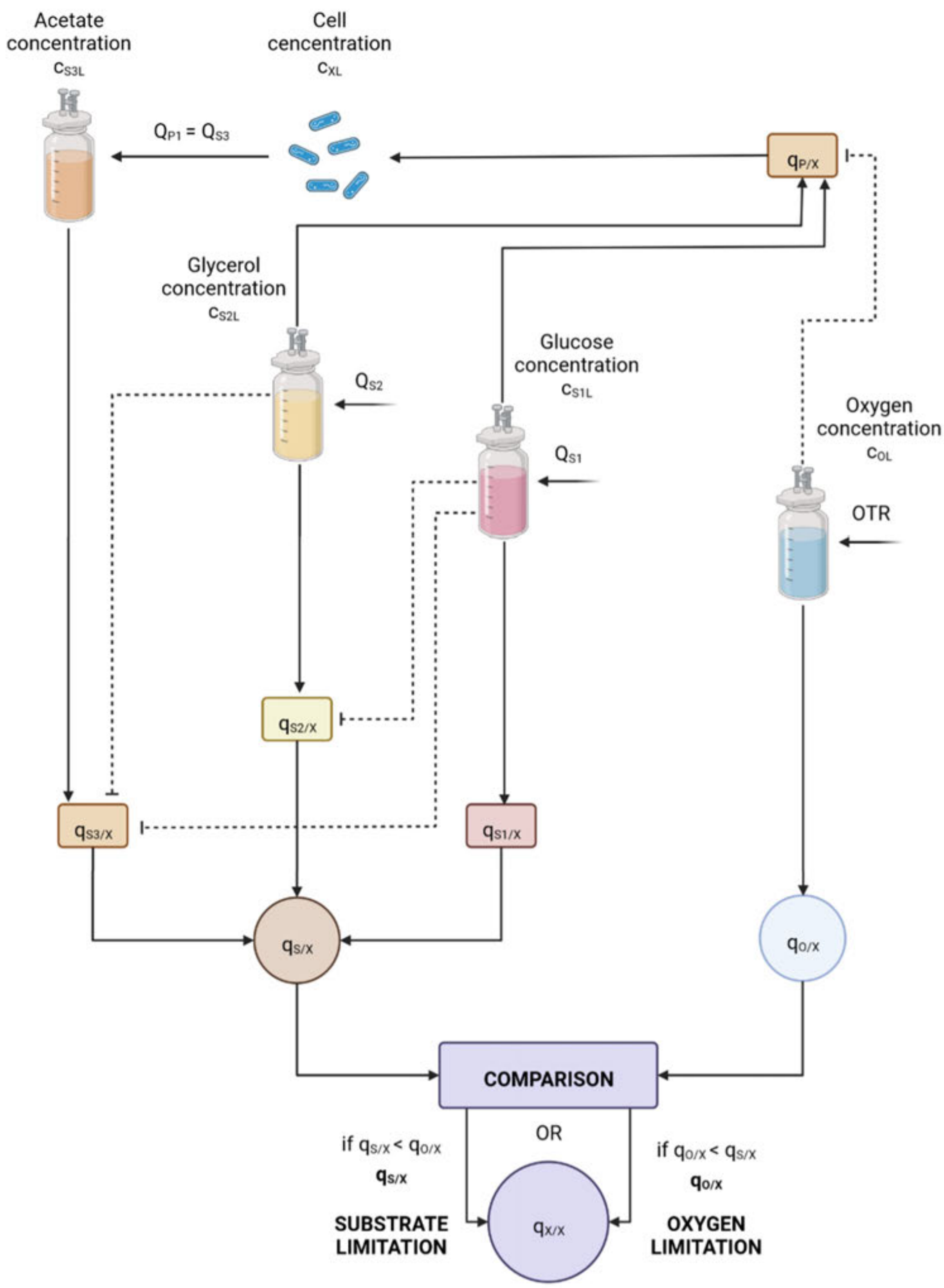


Figure 6: **Visualization of the reactions taking place in the microorganism whose growth is simulated in this work.** The arrows represent an increasing effect, whereas the dotted lines indicate an inhibiting effect. Created with BioRender based on the BIOSIM manual written by Prof. Luttmann [30].

If the oxygen supply is sufficient, the time-dependent cell-specific uptake rate of the preferred substrate 1 (glucose) is only limited by the glucose concentration c_{S1L}. It is given by the equation

$$q_{S1/Xopt}(t) = q_{S1/Xmax} \cdot \frac{c_{S1L}(t)}{k_{S1} + c_{S1L}(t)} \tag{38}$$

with

$q_{S1/Xopt}$:= time-dependent optimum cell-specific glucose uptake rate [h^{-1}]

$q_{S1/Xmax}$:= maximum cell-specific glucose uptake rate [h^{-1}]

k_{S1} := glucose limitation constant [$g \cdot l^{-1}$].

The glycerol uptake is inhibited in the presence of glucose, acetate uptake is inhibited by the presence of both glucose and glycerol. This is taken into account with the limitation constants k_{Iaj} [$g \cdot l^{-1}$] with a = 2,3 and j = 1,2.

Hence, the time-dependent optimum cell-specific substrate uptake rates can be calculated for glycerol

$$q_{S2/Xopt}(t) = q_{S2/Xmax} \cdot \frac{c_{S2L}(t)}{k_{S2} + c_{S2L}(t)} \cdot \frac{k_{I21}}{k_{I21} + c_{S1L}(t)} \tag{39}$$

with

$q_{S2/Xopt}$:= time-dependent optimum cell-specific glycerol uptake rate [h^{-1}]

$q_{S2/Xmax}$:= maximum cell-specific glycerol uptake rate [h^{-1}]

k_{S2} := glycerol limitation constant [$g \cdot l^{-1}$]

k_{I21} := inhibition constant for glycerol (S2) due to glucose (S1) [$g \cdot l^{-1}$]

and for acetate

$$q_{S3/Xopt}(t) = q_{S3/Xmax} \cdot \frac{c_{S3L}(t)}{k_{S3} + c_{S3L}(t)} \cdot \prod_{j=1}^{2} \frac{k_{I3j}}{k_{I3j} + c_{SjL}(t)} \tag{40}$$

with

$q_{S3/Xopt}$:= time-dependent optimum cell-specific acetate uptake rate [h^{-1}]

$q_{S3/Xmax}$:= maximum cell-specific acetate uptake rate [h^{-1}]

k_{S3} := acetate limitation constant [$g \cdot l^{-1}$]

k_{I3j} := inhibition constant for acetate (S3) due to glucose (j = 1) and glycerol (j = 2) [$g \cdot l^{-1}$].

The maximum cell-specific uptake rate for each substrate, $q_{Si/Xmax}$ [h^{-1}], can then be calculated as follows:

$$q_{Si/Xmax} = \frac{\mu_{imax}(t)}{Y_{X/Sigr}} + q_{Si/Xm} \tag{41}$$

with

$Y_{X/Sigr}$:= cell mass growth yield for substrate i; i = 1,2,3 [-]

$q_{Si/Xm}$:= cell-specific maintenance rate for substrate i [g·l^{-1}·h^{-1}].

It contains the time-, pH-, and temperature-dependent maximum cell-specific growth rate on substrate i, μ_{imax}, which is given by

$$\mu_{imax}(t) = \mu_{iopt} \cdot \kappa_{\vartheta_L}(t) \cdot \kappa_{pH}(t) \tag{42}$$

with

μ_{iopt} := optimum specific growth rate on substrate i; i = 1,2,3 [h^{-1}]

κ_I := time-dependent environmental growth control function of parameter I; I = ϑ_L,pH [-].

In the temperature range $\vartheta_L \in [\vartheta_{Lmin}, \vartheta_{Lmax}]$ the growth rate is controlled by the current temperature in the liquid phase. This is accounted for by the control function $\kappa_{\vartheta L}$, which can be calculated as follows:

$$\kappa_{\vartheta_L}(t) = \frac{(\vartheta_L(t) - \vartheta_{Lmin})^2 \cdot (\vartheta_L(t) - \vartheta_{Lmax})}{(\vartheta_{Lopt} - \vartheta_{Lmin}) \cdot [(\vartheta_{Lopt} - \vartheta_{Lmin}) \cdot (\vartheta_L(t) - \vartheta_{Lopt}) - (\vartheta_{Lopt} - \vartheta_{Lmax}) \cdot (\vartheta_{Lopt} + \vartheta_{Lmin} - 2\vartheta_L(t))]} \tag{43}$$

with

ϑ_{Lmin} := minimum temperature limit of growth [°C]

ϑ_{Lmax} := maximum temperature limit of growth [°C]

ϑ_{Lopt} := optimal temperature for growth [°C].

Likewise, the growth behavior is influenced by the pH in a range of pH ∈ [pH$_{min}$,pH$_{max}$] as demonstrated in the following equation for κ_{pH} [-]:

$$\kappa_{pH}(t) = \frac{(pH(t) - pH_{min}) \cdot (pH(t) - pH_{max})}{(pH(t) - pH_{min}) \cdot (pH(t) - pH_{max}) \cdot (pH(t) - pH_{opt})^2} \tag{44}$$

with

pH_{min} := minimum pH limit of growth [-]

pH_{max} := maximum pH limit of growth [-]

pH_{opt} := optimal pH for growth [-].

Outside of their defined ranges both the temperature control function $\kappa_{\vartheta L}$ and the pH control function κ_{pH} are valued at zero. This means that outside of the minimum and maximum limits, no growth can occur.

To keep the model consistent, the cell-specific maintenance rate $q_{X/Xm}$ must be of equal magnitude for all substrates (glucose, glycerol, and acetate). This means that

$$q_{X/Xm} = Y_{X/Sgr} \cdot \ q_{Si/Xm} \tag{45}$$

with i = 1,2,3. Mathematically, the cell will take up all three substrates simultaneously if the oxygen supply is sufficient:

$$\mu_{Sgr}(t) = \sum_{i=1}^{3} Y_{X/Sgr} \cdot q_{Si/Xopt} \leq \mu_{1max} + q_{X/Xm}. \tag{46}$$

Real organisms, in contrast, take up substrates sequentially as the presence of one substrate inhibits the uptake of another due to catabolite repression. This is compensated by the inhibition mechanics described above.

As previously mentioned, the oxygen supply is another limiting factor and the second bottleneck in the model at hand. Comparably to the substrate uptake rate, the time-dependent oxygen uptake rate $q_{O/X}$ [h^{-1}] also contains a growth and a maintenance part:

$$q_{O/X}(t) = q_{O/Xgr}(t) + q_{O/Xm} \tag{47}$$

The growth part $q_{O/Xgr}$ [h^{-1}] refers to, for instance, the energy required for oxygen uptake, while the maintenance part $q_{O/Xm}$ [h^{-1}] refers to the conversion of maintenance storage substances.

If oxygen is the limiting factor, the time-dependent growth fraction μ_{Ogr} [h^{-1}] can be calculated as

$$\mu_{Ogr}(t) = Y_{X/Ogr} \cdot \ q_{O/Xgr}(t) = \left(\mu_{1max} + q_{X/Xm}\right) \cdot \frac{c_{OL}(t)}{k_O + c_{OL}(t)} \tag{48}$$

with

$Y_{X/Ogr}$:= oxygen growth yield coefficient [-]

k_O := oxygen limitation constant [g·l^{-1}].

μ_{Ogr} is controlled by the dissolved oxygen concentration c_{OL} [g·l^{-1}].

As illustrated in equation 37, the rate-determining step is to be assessed by evaluating if the growth fraction is smaller for the oxygen or the substrate uptake:

$$q_{X/Xgr}(t) = min\left\{\mu_{Sgr}(t), \mu_{Ogr}(t)\right\} \tag{49}$$

with

$q_{X/Xgr}$:= time-dependent resulting cell-specific growth fraction [h^{-1}].

Subsequently, the associated cell-specific reaction rates can be calculated.

The time-dependent cell-specific growth rate $q_{X/X}$ is given by

$$q_{X/X}(t) = q_{X/Xgr}(t) - q_{X/Xm} \tag{50}$$

and the time-dependent cell-specific oxygen uptake rate $q_{O/X}$ is given by

$$q_{O/X}(t) = \frac{q_{X/Xgr}(t)}{Y_{X/Ogr}} + q_{O/Xm}, \tag{51}$$

while the time-dependent cell-specific glucose rate $q_{S1/X}$ [h^{-1}] can be calculated as

$$q_{S1/X}(t) = \frac{q_{X/Xgr}(t)}{Y_{X/S1gr}} \tag{52}$$

and the time-dependent cell-specific ammonia uptake rate $q_{Al/X}$ [h^{-1}] as

$$q_{Al/X}(t) = \frac{q_{X/Xgr}(t)}{Y_{X/Algr}} \tag{53}$$

with

$Y_{X/Algr}$:= cell mass yield for ammonia [-].

The time-dependent cell-specific glycerol uptake rate $q_{S2/X}$ [h^{-1}] can then be assessed with

$$q_{S2/X}(t) = \frac{q_{X/Xgr}(t) - Y_{X/S1gr} \cdot q_{S1/X}(t)}{Y_{X/S2gr}} \tag{54}$$

as well as the time-dependent cell-specific acetate uptake rate $q_{S3/X}$ [h^{-1}] with

$$q_{S3/X}(t) = \frac{q_{X/Xgr}(t) - \sum_{i=1}^{2} Y_{X/Sigr} \cdot q_{Si/X}(t)}{Y_{X/S3gr}} \tag{55}$$

With $q_{S1/X}$, $q_{S2/X}$ and $q_{S3/X}$ known, the time-dependent cell-specific acetate production rate $q_{P/X}$ [h^{-1}] can be calculated:

$$q_{P/X}(t) = \frac{k_{IPO}}{k_{IPO} + c_{OL}(t)} \cdot \sum_{i=1}^{2} Y_{P/Si} \cdot q_{Si/X}(t) \tag{56}$$

with

k_{IPO} := inhibition constant for acetate production due to oxygen [g·l^{-1}]

$Y_{P/Si}$:= acetate yield from substrate i, i = 1,2 [-].

Acetate is produced from either glucose or glycerol, although its production is inhibited by oxygen.

As mentioned earlier, the model designed by Prof. Luttmann and implemented in the simulation at hand has additional sub-models, which are illustrated in full in the BIOSIM manual [30].

3 Methods

3.1 MATLAB® R2022b

MATLAB®, derived from "MATrix LABoratory", is a programming language capable of directly expressing matrix and array mathematics sold by the MathWorks® corporation. This proprietary software includes a numeric computing environment designed to operate on whole matrices and arrays, contrasting most other programming languages, which mostly work on one number at a time [31, 34]. The MATLAB® version that was used to program the simulation at hand is R2022b. Additionally, MATLAB® contains a variety of toolboxes for a multitude of applications, such as data science or optimization. It furthermore contains a built-in tool for designing applications, the MATLAB® App Designer. With this tool, the user may program and compile apps that can be run independently from the MATLAB® environment.

3.2 MATLAB® App Designer

Introduced in 2016 with version R2016a, the MATLAB® App Designer is a tool for creating apps with the core mechanic of dragging and dropping premade app components to lay out the design of a GUI. These premade components can range from simple edit fields or buttons to check box trees or context menus [35, 36, 37]. While the user selects components and arranges them in the app space, App Designer subsequently generates the corresponding core code automatically, which makes the creation of an app accessible and straightforward. This core code may not be edited by the user but can be overwritten by redefining the characteristics of the objects in the app elsewhere in the app code. Once the needed components are present, the app behavior may be coded.

3.2.1 Use of callback functions in App Designer

In essence, MATLAB® functions themselves are a task or a set of tasks that accept an input and return an output. They are generally used when these tasks need to be performed repeatedly to avoid coding the same expressions multiple times. Once a function is defined, it can be called upon indefinitely [38]. In MATLAB® App Designer, functions are used as callback functions to program the behavior of specific app components and as methods that can be accessed by any app component.

Callback functions are specific to their respective component and triggered once the event associated with that specific component is executed. For instance, the indicator on a slider is moved to point at a different value on the slider. The event "indicator was moved" then triggers the slider callback function, which uses the new input, i.e., the new numerical value corresponding to the position the slider is at now, to calculate a new output. Events can range from the push of a button to checking a box or even resizing a window.

General functions can be defined as a method in MATLAB® App Designer and allow for the reuse of code. These functions can be accessed anywhere in the app and are not associated with any specific app component or event. Private functions may only be accessed by their app of origin, whereas public functions may be accessed by other apps as well. This is useful if a specific function should be triggered by multiple different events or if inputs need to be relayed from one app to another.

In the context of functions and their execution, the timer object is of importance. The timer is an object capable of triggering events at specific points in time and is central to the application established in this work. A timer can be started upon launching the app or in response to a specific event. The timer object fires in certain time intervals as specified by the user. The event "timer has fired" in turn triggers a timer-associated function until the timer has either fired a specific number of times or until it is stopped manually.

3.2.2 Use of properties in App Designer

MATLAB® App Designer, furthermore, shows some differences in its syntax and command usage compared to standard MATLAB® scripts. The function of global variables is taken over by so-called properties, which are defined in the app code. These properties are variables that can be referred to anywhere in the app code by any function. If a variable is not defined as a property, it can only be accessed by the function it was defined in. By default, properties are classified as private and can only be referred to and accessed in the app they are defined in. If a property is, however, defined as public, it can be accessed by other apps that communicate with the app the public property is defined in. Properties are used to pass information between different functions and different apps or store it for later use.

3.2.3 Use of dot-indexing in App Designer

The app properties, functions, and components of the GUI need to be referred to using dot-indexing. Dot-indexing here works similarly as in usual MATLAB scripts, in which certain features of an object may be changed by accessing the specific feature and setting it to the desired value in the form of a name-value argument. This is done, for example, to change the color of a plotted line. If the object is accessed in the same app that it originates from, it is referred to using the "app.[property/function/component]" notation. If an object in another app is to be accessed, that object is referred to as "app.[name of the app as defined in the code of the current app].[property/function/component]".

3.2.4 Solving ordinary differential equations in MATLAB®

MATLAB® is, moreover, very adept at solving first-order differential equations. There are a variety of ODE-solving commands available with varying degrees of accuracy and suitability for stiff and nonstiff problems. The solutions to ODE problems are attained iteratively with solutions of higher accuracy requiring more computing power. Which ODE command to use is determined by the problem type (stiff or nonstiff), the required accuracy, and the available computing power [39, 40].

The stiffness of an ODE problem refers to the degree of scaling difference in the problem, such as two solution components varying on considerably different time scales [40]. This does not apply to the problem at hand, as the equations describing the fermentation model are conceived to vary on a comparable time scale. Hence, an ODE solver for a nonstiff ODE system is needed.

Naturally, the accuracy of the ODE system solution and thus of the model should be as high as possible. However, as there are around 30 different differential equations in the model at hand that need to be solved for every point in (simulated) time in intervals of milliseconds, the computing power is a limiting factor. Thus, an ODE solver of medium accuracy should be sufficient.

The ode45 ODE solver is suited for nonstiff problem types, operates at medium accuracy, and does not require too much computing power. Thus, it was the ODE solver chosen for solving the ODE systems that constitute the fermentation model at hand [40]. How ODE systems are solved in the simulation established in this work is illustrated in the following abstract of its source code:

```
function [dyV] = ODE_Volume(FR,FH,FT1,FT2,t,yV)

% Liquid volume balance [l/h]
dyV(1) = FR+FT1+FT2-FH;

% Substrate reservoir balance [l/h]
dyV(2) = -FR;

% pH-base reservoir balance [l/h]
dyV(3) = -FT2;

% pH-acid reservoir balance [l/h]
dyV(4) = -FT1;

dyV = dyV';
end
```

Figure 7: **ODE system concerning the liquid volume balance in the bioreactor.** The system consists of an equation for the overall volume balance (dyV(1)), the substrate reservoir balance (dyV(2)), the pH-base reservoir balance (dyV(3)), and the pH-acid balance (dyV(4)).

```
% Define ODE function for volume calculation
ODE_Vol       = @(t,yV) ODE_Volume(app.FR(end),app.FH(end),FT1,FT2,t,yV);

% Liquid volume balance
[~,yV] = ode45(ODE_Vol,[0 app.CallingApp.deltat],[app.VL(end) app.VR(end) app.VT2(end) app.VT1(end)]);
app.VL(end+1)  = yV(end,1);
app.CallingApp.mLkgEditField.Value = app.VL(end)*app.CallingApp.rhoL;
app.VR(end+1)  = yV(end,2);
app.VT2(end+1) = yV(end,3);
app.VT1(end+1) = yV(end,4);
```

Figure 8: **Abstract of the simulation source code corresponding to the online solving of the previously shown liquid volume ODE system.** The ODE system and its required variables are first defined and associated with a handle. Then, the ode45 solver is called upon to solve the ODE system. The entries for the next point in time for all liquid balances are then made in the form of an additional entry into a preexisting property.

As illustrated by figures 7 and 8, the ODE system itself is defined in an external .m file as a MATLAB® function. This function is called by the simulation by providing a function handle and defining the variables in the code that should be passed to the function (current feed rate, current harvest rate, and the current titration rates for acid and base in this case). The ode45 solver is then called with the input arguments time interval and initial values. The generated data can in turn be used to create the next entries in the simulated liquid weight development.

3.3 Getting started with MATLAB® and its App Designer

Upon launching the MATLAB® software, the user is directed to the startup page consisting of an overview of the currently chosen folder containing all saved MATLAB® files, the workspace containing all currently defined variables, and the command window. The command window is used for a limited number of commands at once, as its history is usually not saved. For larger programs, a script can be created. The command window can be used to launch the built-in MATLAB® App Designer. Alternatively, the App Designer can be launched by selecting "Design App" in the "Apps" tab. Once the App Designer is opened, the user may switch between two different views: the design view, which is where the different GUI components can be dragged and dropped to compose the app layout, and the code view, where the automatically generated code parts are displayed and the content of functions callback or other may be added. When an app is finished, it can be launched by pressing the green arrow at the top where the save and export/compilation features are located as well.

In the "Design View" window, the different components are listed in the component library on the left side. From here, any component can be dragged and dropped into the app space in the middle of the window. On the right side, the component browser gives an overview of the components that constitute the app at hand. This component browser also gives modification options for each component, such as color or text changes. The "Design View" window is shown in figure 9.

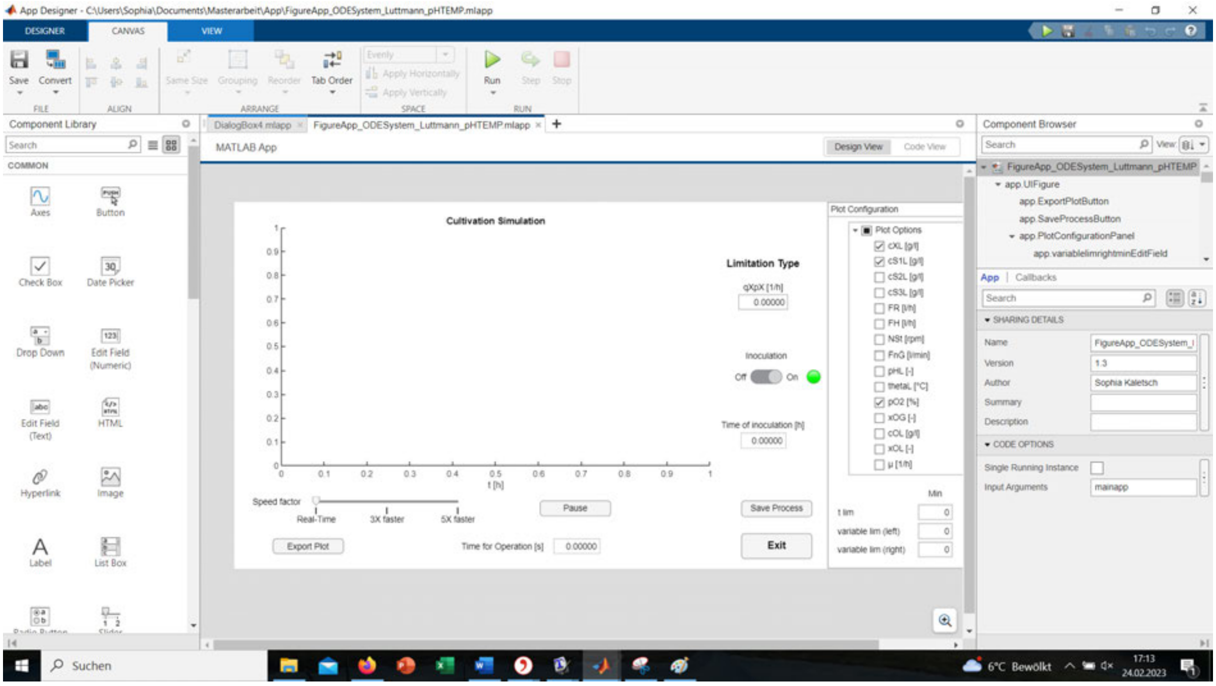


Figure 9: **Design view of the MATLAB® App Designer.** This window is used to assemble the app components, launch the app, save it, or compile it.

In the “Code View” window, the automatically generated code can be viewed and the app behavior can be programmed. Automatically generated code is grayed out while editable code is not. Callback functions may be added by selecting the “Callback” option at the top or by clicking right on an app component and selecting “add callback”. Other public or private functions can be added likewise by selecting the “Function” option at the top. Public or private properties are added the same way. The left side offers an overview table of callbacks and functions at the top and displays the app layout at the bottom. The right side consists of the same component browser that is visible in the “Design View” window. The “Code View” window is shown in figure 10.

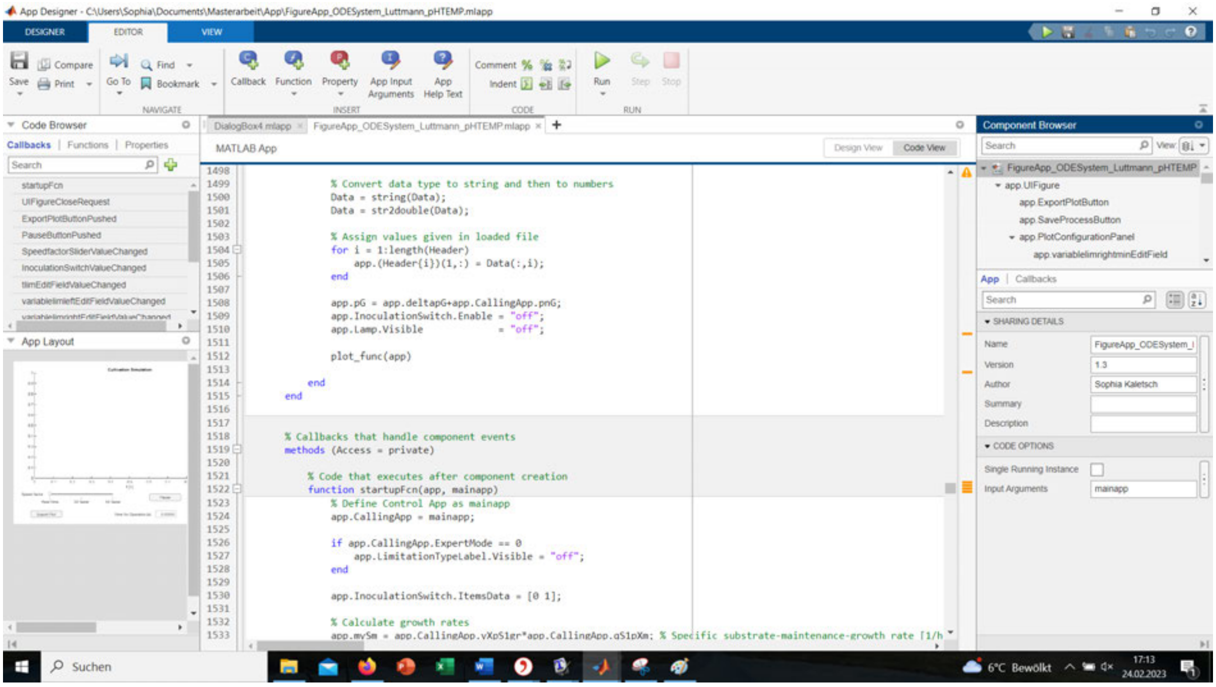


Figure 10: **Code view of the MATLAB® App Designer.** This window is used to program the app behavior.

When an app is launched from the App Designer, it is necessary that all needed files, such as function files, files containing data, or other apps communicating with the app at hand, are located in the currently selected folder or subfolders thereof. This is illustrated in figure 11.

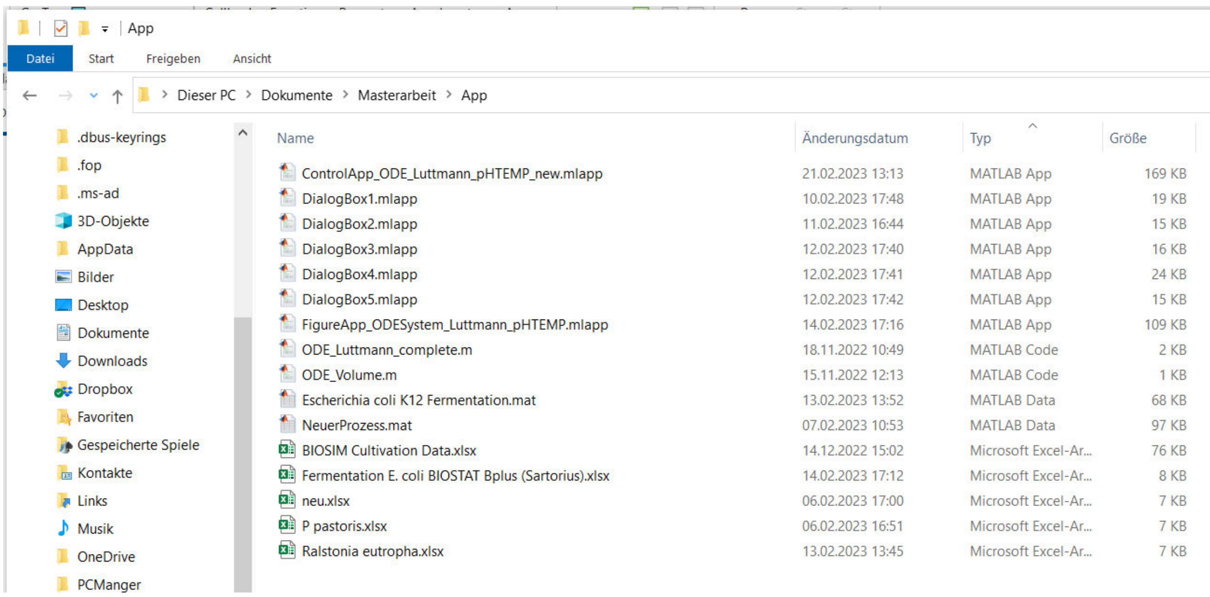


Figure 11: **All files communicating with an app need to be located in the same folder or subfolders thereof.** If this is not the case, files cannot be accessed and the app gives an error message.

3.4 Process control

In contrast to the BIOSIM model, which is connected to an external DCU, this simulation is conceived as a standalone. Thus, the process control that regulates the fermentation is also embedded in the app itself. This also has the advantage that the simulation can be sped up without a change in controller behavior and without the need to adjust any controller parameters. The external DCU unit always processes the signals it receives the same way, regardless of the simulation speed. In this chapter, the setup of the controllers and the derivation of their underlying mathematical principles for the MATLAB®-based simulation are explained.

The controllers utilized in this simulation are SPCs and SISO P, PI, and PID controllers. SPCs realize setpoints transmitted to them by a SISO controller via cascading or manually entered by the operator. For example, the setpoint for the stirrer speed may be given by the p_{O2}-agitation controller or by the operator by entering it into the corresponding edit field. In the MATLAB®-based situation, this translates as the actual value of a control variable being set to its setpoint.

The P, PI, and PID controllers compensate for errors of a controlled variable by proportional, integral, and derivative means or combinations thereof. The input is given by the error between the setpoint and the actual value of the controlled variable, the output is the control variable for the control element. In this simulation, the measured error ε is filtered through a PT1 delay. A PT1 delay is characterized by its output signal exponentially approaching the input signal given to the PT1 delay. This gradual approach is determined by the time constant T. In this case, the PT1 delay is used to even out the measured signal and avoid fluctuations. The equations for the P-, I-, and D-parts of a controller as well as the PT1 delay were based on those described by Unbehauen [41].

The measured error is calculated as follows:

$$\varepsilon(t) = w(t) - x(t) \tag{57}$$

with

$w :=$ time-dependent setpoint for the controlled variable

$x :=$ time-dependent actual value of the controlled variable.

The equation for the filtered error e through a PT1 delay is given by:

$$e(t) = T \cdot \frac{de(t)}{dt} + \varepsilon(t) \tag{58}$$

with

T := time constant of PT1 delay [h].

In the simulation, $\frac{de(t)}{dt}$ was calculated by dividing the difference between the last two measured errors by the time difference Δt. The equation was thus realized as:

$$e(t) = T \cdot \frac{e(t-\Delta t) - e(t)}{\Delta t} + \varepsilon(t) \tag{59}$$

As the filtered error for the last point in time e(t-Δt) is known, the equation can be rearranged to e:

$$e(t) = \frac{\varepsilon(t) + \frac{T}{\Delta t} \cdot e(t-\Delta t)}{\frac{T}{\Delta t} + 1} \tag{60}$$

To derive the normalized filtered error e_n [-], the filtered error e is divided by the difference between the minimum and maximum setpoint for the respective controlled variable:

$$e_n(t) = \frac{e(t)}{w_{max}(t) - w_{min}(t)} \tag{61}$$

with

w_{max} := maximum value of setpoint for the controlled variable

w_{min} := minimum value of setpoint for the controlled variable.

Once the filtered and normalized error e_n is known, the controller output y [-] can be calculated through variations of the following mathematical principle:

$$y(t) = K_P \cdot \ e_n(t) + K_I \cdot \int_0^t e_n(\tau)\, d\tau + K_D \cdot \frac{de_n(t)}{dt} \tag{62}$$

with

K_a := gain of controller of type a, a = P,I,D [-]

y refers to the control variable, such as the speed of a stirrer or the temperature of the fermenter double jacket. y is normalized, thus a value of 1 would refer to the maximum possible output, and 0 to no output of the control variable. The controller aims to adjust the control variable constantly in a way that allows a measured process variable or controlled variable to be kept constant. This could mean, for instance, that the speed of the stirrer is increased when cells multiply to compensate for their increased oxygen uptake so that the p_{O2} will be kept constant.

The proportional part of a controller is given by the term $K_P \cdot e_n(t)$. This is the proportional section of the controller, as the control variable and the error value are proportionally related. The gain of the P-part K_P dictates how large the influence of the controller P-part is on the control variable. This term can be calculated easily in MATLAB®, as the gain and the filtered error are known.

The integral part of a controller is given by the term $K_I \cdot \int_0^t e_n(\tau)\,d\tau$. It accounts for the error value history by integrating all errors that occurred in the span of 0 to t. For example, if the measured process variable p_{O2} has been below the setpoint for a while, this will cause the integrated error to become larger and thereby increase the output represented by the control variable. The stirrer controlled by a PI controller would, in this case, stir faster than the stirrer controlled by a P controller. The integral term is calculated in MATLAB® for the last two calculated filtered errors and then added to the previously calculated integral. Thus, the equation used in the simulation is:

$$y_I(t) = K_I \cdot \frac{e_n(t-\Delta \mathrm{t}) + e_n(t)}{2} \cdot \Delta \mathrm{t} + y_I(t-\Delta \mathrm{t})\,. \tag{63}$$

Again, the gain of the I-part K_I dictates how large the influence of the controller I-part is on the control variable.

The derivative part of a controller is given by the term $K_D \cdot \frac{e_n(t)}{dt}$ and accounts for the pace at which the error decreases or increases at time t. For example, if there is still a difference between the p_{O2} setpoint and the actual value of the p_{O2}, but the error is becoming rapidly smaller, meaning the slope of the error is negative, the D-part will dampen the output of the control variable to prevent an overshoot. Vice versa, if the error becomes rapidly larger, meaning the slope of the error is positive, the D-part will increase the response of the control variable accordingly. In the simulation at hand, the derivative of the filtered error is calculated simply by assessing the slope between the last two variable entries:

$$y_D(t) = K_D \cdot \frac{e_n(t) - e_n(t-\Delta \mathrm{t})}{\Delta \mathrm{t}}\,. \tag{64}$$

Similarly to the gains of the P- and I-parts, the gain of the D-part K_D dictates how large the influence of the controller D-part is on the control variable.

Once the controller output is calculated, it should be assessed in regard to the maximum possible values for the controlled variable, i.e. the calculated output for the stirrer speed cannot be higher than the maximum possible stirrer speed or lower than the minimum possible stirrer speed. Hence, the control variable is kept steady in the range [0,1] for regular controllers or [-1,1] if the controller is operating at split range. In the case of the agitation or aeration controllers, the control variable should not become zero as to keep agitation and aeration to a minimum even if the oxygen in the liquid phase is sufficient. An example of how a controller was implemented in the MATLAB® simulation for the p_{O2}-feed control is given in figure 12.

```
elseif app.CallingApp.pO2_feed == 1

    % Calculate deviation from setpoint
    cEfeed            = app.CallingApp.pO2w-app.pO2(end); % Error/Controller difference between measured pO2 and Setpoint e = (w-x)
    app.cE_feed       = (cEfeed+0.001/app.CallingApp.deltat*app.cE_feed)/(0.001/app.CallingApp.deltat+1); % Filtered error through a PT1 delay with T = 0.001 and deltat
    app.ce_feed(end+1) = app.cE_feed/(100-0); % Normalized controller difference e = (w-x)/(w_max - w_min); [-1,+1]

    % Calculate controller outputs
    app.cP_feed = app.ce_feed(end)*app.CallingApp.KP_feed; % P-part of controller with KP = -2.0
    app.cI_feed = app.cI_feed+(app.ce_feed(end)+app.ce_feed(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_feed; % I-part of controller with KI = -15.0 and deltat
    app.cD_feed = (app.ce_feed(end)-app.ce_feed(end-1))/app.CallingApp.deltat*app.CallingApp.KD_feed; % D-part of controller with KD = -0.009

    % Calculation of new relative setpoint for feed pump
    % PID sum in percent
    app.yfeed  = ((app.cP_feed+app.cI_feed+app.cD_feed)*100+app.yfeed)/2;

    % Define limits for new setpoint
    if app.yfeed < 0
        app.yfeed = 0;
    elseif app.yfeed > 100
        app.yfeed = 100;
    end

    % Calculation of new agitation speed setpoint [l/h]
    app.FR(end+1) = app.yfeed/100*app.CallingApp.FRmax;
```

Figure 12: **MATLAB® code for the p_{O2}-feed controller.** The measured error is calculated, filtered and normalized. Afterward, the proportional, integral, and derivative outputs of the controller are calculated. Their sum is then passed through a limit assessment (0 % to 100 % in this case) and used to calculate the new feed rate.

The proportional, integral, and derivative controller parts can be mixed and matched at will, though a proportional part is always present. The SISO controllers employed in the simulation at hand are mostly P or PID controllers and an overview of their parameters is given in table 1.

Table 1: **Controller parameters of the SISO controllers implemented in the simulation.** The gains for P-, I-, and D-parts are given – if applicable – and the minimum/maximum values of the respective control variables are indicated as well as any dead bands.

Control loop	K_P	K_I	K_D	Maximum value	Minimum value	Dead band
p_{O2}-agitation	10.0	1000.0	0.015	1500 rpm	450 rpm	-
p_{O2}-aeration	20.0	0.003	0.0188	AIR: 10.0 l/h; O2: 5.0 l/h	AIR: 3.0 l/h; O2: 0 l/h	-
p_{O2}-gasmix	0.4	0.003	0.0005	1	0.2094	-
p_{O2}-feed	-2.0	-15.0	-0.009	0.5 l/h	0 l/h	-
Temperature	8.0	0.1	-	1	-1	-
Heating	0.01	-	-	1500 kg/h	0 kg/h	-
Cooling	-10.0	-	-	1500 kg/h	0 kg/h	-
pH	10000.0	-	-	1	-1	0.1
Acid	1.0	-	-	1.0 l/h	0 l/h	-
Alkali	-1.0	-	-	1.0 l/h	0 l/h	-
Liquid weight-harvest	10.0	0.3	1.0	5 l/h	0 l/h	-
Foam	-	-	-	-	-	-

3.4.1 Special controller types

The p_{O2}, pH, liquid weight, and temperature controllers operate in a cascade control arrangement. This means that two or more controllers are interlinked in such a way that the output of the first controller, the master controller, becomes the input of one or more following controllers, the slave controllers. This allows for a better dynamic response of the control loop to disturbances [42]. As defined by Prof. Luttmann, all slave controllers employed by the p_{O2} and liquid weight master controllers are SPCs. In the simulation at hand, this means that the actual value of the controlled element, i.e. the stirrer speed, is simply set to the output given by the master controller. For example, in the case of the p_{O2}-agitation controller, the p_{O2} is measured and compared to its setpoint by the master controller. The master controller then calculates the new setpoint for the stirrer speed, which the actual value of the stirrer speed is then set to. In contrast, the slave controllers of both the temperature and the pH master controllers are P controllers. Thus, the actual values of their controlled elements are not derived from the master controller, but from a SISO slave controller. For example, the temperature master controller calculates a new setpoint for the reactor double jacket temperature. The temperature slave controllers then calculate the new setpoints for either the steam flux in case of heating or the cooling water mass flow in case of cooling.

In addition, the pH and temperature controllers operate at split range. This means that the output from one controller serves as a control variable for two or more different control elements. In contrast to multi-channel controllers, such as the p_{O2}-aeration and -gasmix slave controllers, split range controllers operate on an exclusive basis. Either one or the other control element is employed at once. To exemplify this, the temperature master controller relays the new setpoint for the temperature of the double jacket to either, if it is too high, the cooling water slave controller or, if it is too low, the steam heating slave controller. The pH controller, similarly, may either drive the acid pump SPC if the pH is too high or the alkali pump SPC if the pH is too low. This was explained in detail for the temperature controller in chapter 2.2.1. If, in contrast, the p_{O2}-aeration controller is employed, it can drive both the air aeration and the pure oxygen aeration SPC at once should the need arise.

The antifoam controller is a special case. It is not conceptualized as a conventional controller altogether but instead adds an antifoam agent to the liquid phase based on the foam build-up rate and the cell concentration. It is only mentioned in the table to satisfy the requirement of comprehensiveness.

The controllers utilized in this work are illustrated in figure 13.

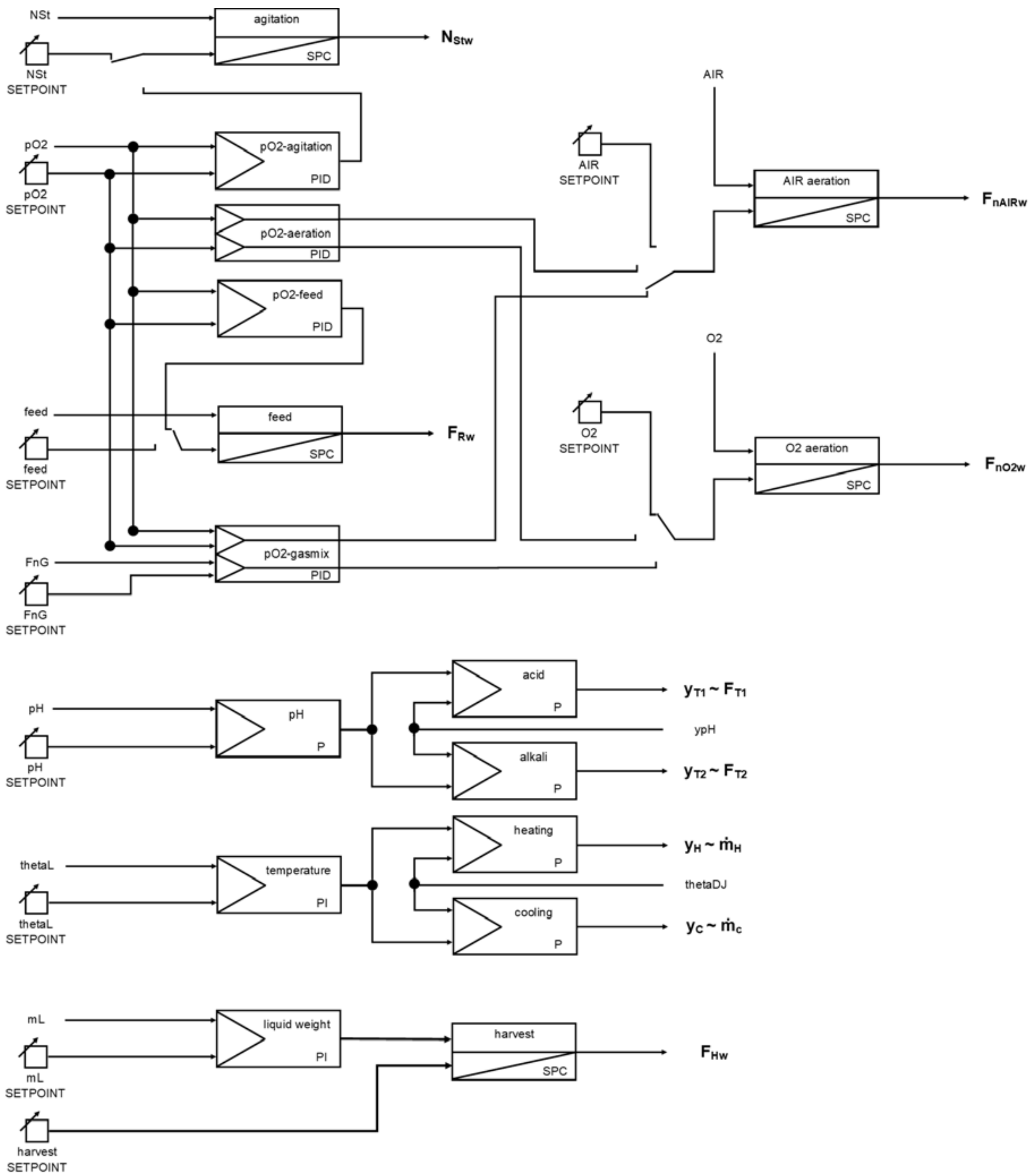


Figure 13: **Schematic overview over the controllers employed in the MATLAB®-based simulation.** The antifoam controller is not depicted as it is not a conventional controller. Adapted from the BIOSIM manual written by Prof. Luttmann [30].

4 Results and discussion

In this chapter, the graphical user interface and the different functionalities of the MATLAB®-based simulation are explained. The theoretical design of the MATLAB®-based simulation was explained in chapter 2.1. Figure 14 illustrates how this design was translated into the finished program.

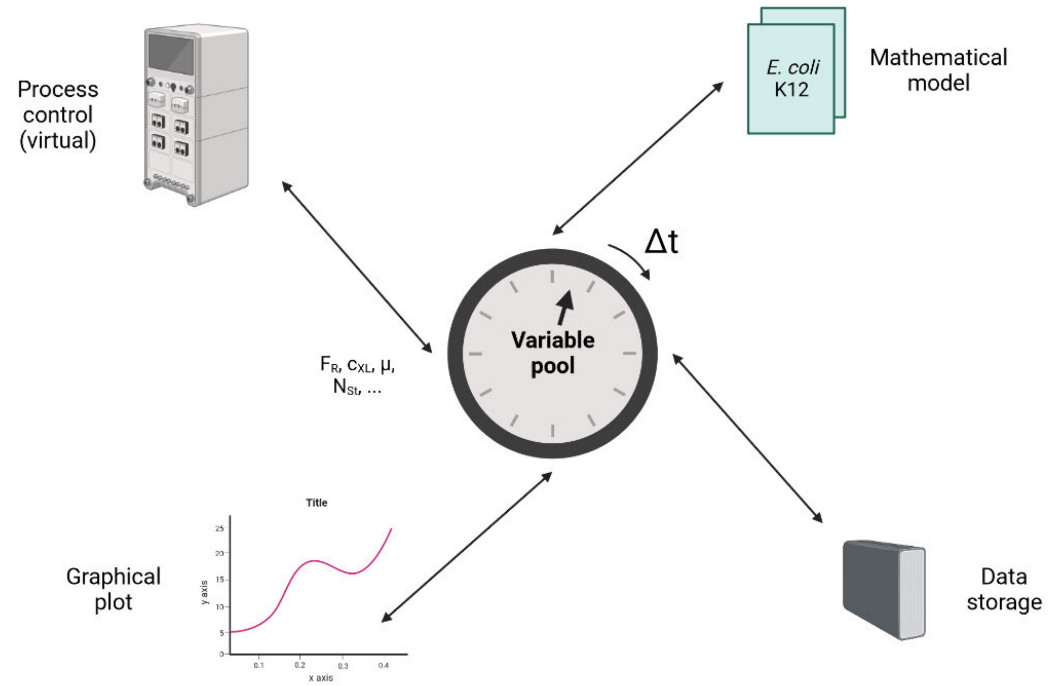


Figure 14: **Visualization of the mode of operation of the MATLAB®-based simulation.** This figure represents how the theoretical design of the simulation was implemented in the finished program. Created with BioRender.

The central element of the MATLAB®-based simulation is the variable pool. It contains all current and previous values of the variables of both the process control and the mathematical model. The variables are updated after the amount of (simulated) time specified by Δt has passed (0.0005 h or 1.8 s per default). For every update of the variable pool, the virtual process control assesses which controllers are selected and which setpoints for the controlled variables or SPCs were chosen by accessing the variable pool. The controller outputs and values for the control variables are then calculated and all generated variables are again saved to the variable pool. In the case of PI and PID controllers, the variable histories of the controlled variables are accessed to calculate the controller I- and D-parts.

Once the operations related to the process control are finished, the variable pool is accessed by the mathematical model, which then calculates the new values for its variables based on the variables related to the process control. For example, in the case of p_{O2}-agitation control, the process control calculates the new setpoint for the stirrer speed. This setpoint is then saved to the variable pool and the mathematical model incorporates it into, among many others, the calculation of heat generated by the stirrer as seen in equation 24. After the mathematical model has completed its calculations, the generated values for its associated variables are saved to the variable pool. Optionally, the data can then be saved by storing them in an external .txt file.

When the process control and mathematical model have concluded, the updated variables are visualized in a graphical plot and the cycle can start anew. Per default, the simulation progresses in real-time. This means that for every 1.8 s that pass in the simulation, 1.8 s pass in reality. However, the simulation may be sped up to a fifth of Δt, meaning that the simulation will progress five times as fast as real-time. In the simulation, Δt will remain the same, meaning that the time increments at which new data is calculated will always correspond to Δt, regardless of how fast the variable entries were updated. The speed at which the variable pool can be updated is thus only limited by the available computing power.

The user interacts with the process control and mathematical model by making changes to the variable pool, for example by changing the setpoint for the stirrer speed or the initial cell concentration. The GUI represents the interface between the user and the simulation by allowing the user to interact with the variable pool through a variety of ways illustrated in the following.

The simulation established in this work consists of two apps, the control app and the simulation app. The control app contains the process control options and general app operation functionalities, whereas the simulation app contains the mathematical model and the plot of its generated data. This allows for the figure to be resizable without affecting the layout of the control options as well as improving the convenience of the inbuilt features.

4.1 The control app

The control app needs to be launched before any other component of the simulation, as it serves as their point of reference. There are two distinct modes in which it can be used: expert and student mode. Both modes function largely similarly, though the expert mode allows the user to access and change controller parameters and get information on the currently active bottleneck when the simulation is running. Upon launching the control app, a dialogue window will open prompting the user to enter a password (figure 15A). This password is set to “BPA1” and hardcoded into the control app. If the password is to be changed, the value of the property “Password” in the control app can be set to any other string of words or numbers. If the password was entered correctly, another window will open, notifying the user that expert mode is now accessible (figure 15B). If the dialogue window was closed or the password entered incorrectly, another dialogue window will open, informing the user that expert mode is not accessible (figure 15C).

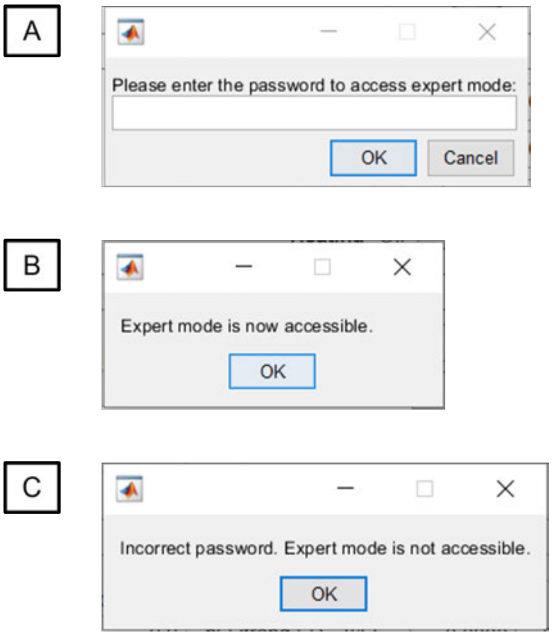


Figure 15: **Dialogue windows related to accessing the expert mode of the control app.** A: Dialogue window prompting the user to enter the expert mode password. B: Dialogue window informing the user that the password was correct and expert mode is now accessible. B: Dialogue window informing the user that the password was incorrect and expert mode is not accessible.

If the control app was launched in student mode, it is possible to access expert mode at any time by pressing the “Expert mode” button located in the overarching bottom panel of the control app and reopening the password dialogue window.

The control app consists of four distinct tabs and an overarching panel indicating the simulation status. The different tabs, “Control Options”, “Variable Pool”, “App Operation” and “Author Information” may be selected by clicking on the tab names at the top of the app window. The “Control Options” tab of the control app in expert mode is shown in figure 16. If the app is launched in student mode, the “Parameters” buttons located on the right side of the “Control Options” tab are not visible. This is the only change made to the control app in expert mode.

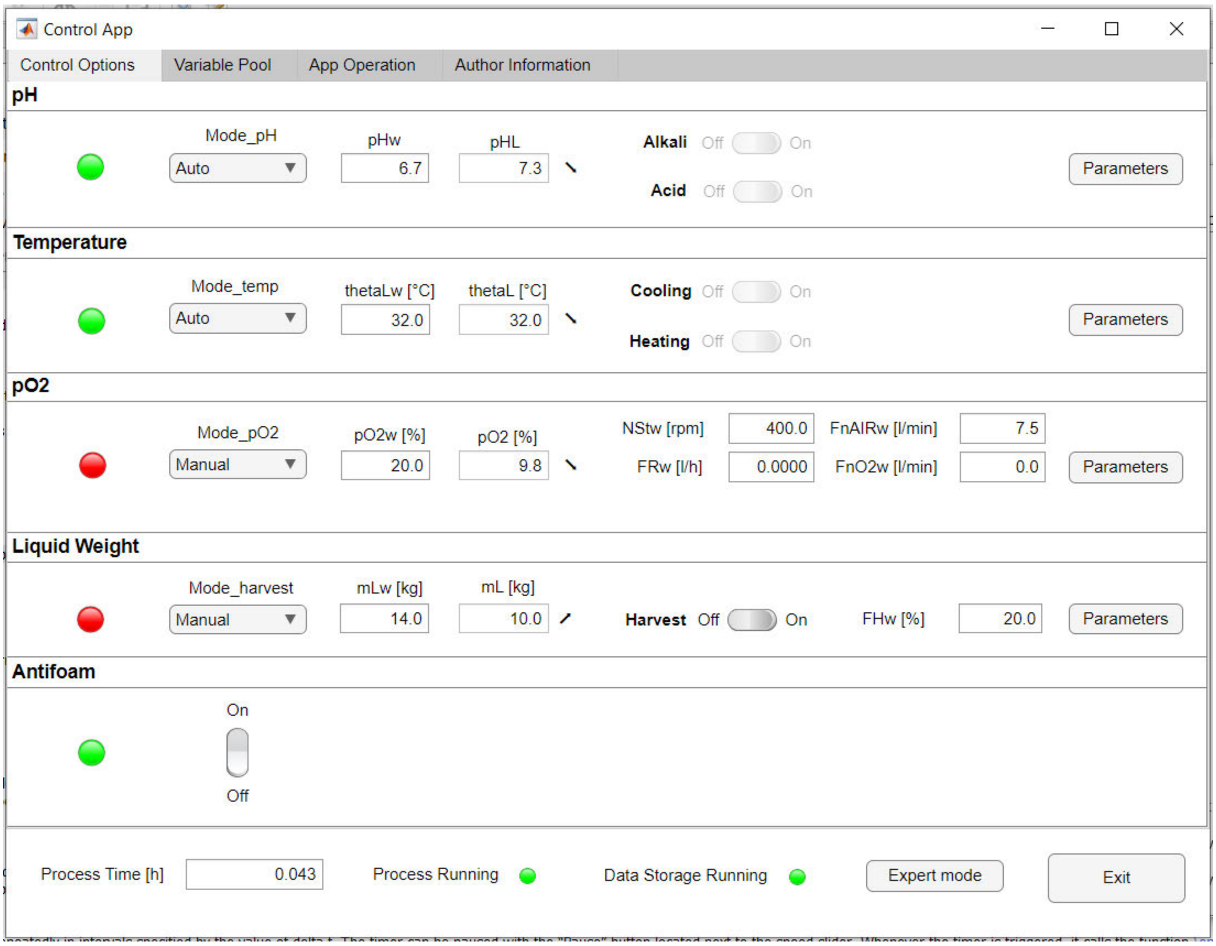


Figure 16: **"Control Options" tab of the control app in expert mode.** This tab is the first tab shown once the app is launched. Here the operator can choose which controllers to activate and, in case of p_{O2} control, which control variable to employ. If manual control is selected, the control variable setpoints can be set via SPCs.

The panel at the bottom of the app window is always visible, independently of the selected tab. Here the process time is indicated along with a light signifying whether the simulation is running (green: simulation in progress, red: simulation is not in progress) and a light signifying whether the data generated by the simulation are being saved (green: data are being saved, red: data are not being saved). To the right of the data storage lamp, the "Expert mode" button is located. At the bottom right corner, the exit button is located. This button is linked to the same *UIFigureCloseRequest* function as the built-in "x" button at the top right corner of the app window. If either button is pressed, a window is opened asking the operator to confirm the close request. The app only closes if the close request is confirmed by pressing "OK". Closing the control app will also prompt the simulation app to close, should it be running. Pressing "Cancel" will close the close request window and resume regular app operations. The close request window is shown in figure 17.

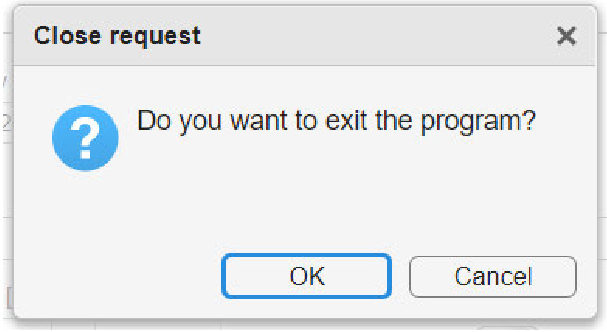


Figure 17: **Close request window.** This window opens when the operator triggers the *UIFigureCloseRequest* function by pressing the close button in the top right or the exit button in the bottom right corner of the control app. Selecting "OK" will then close the app and end the program, while selecting "Cancel" will close the close request window and return the operator to the control app.

4.1.1 Control Options

When starting the app, the user is first directed to the "Control Options" tab. This tab contains all controllers implemented in this simulation organized by controlled variable. Implemented are controllers for pH, temperature, p_{O2}, liquid weight, and antifoam. The p_{O2}-control is unique, as it comprises four different control modes: p_{O2}-feed, p_{O2}-agitation, p_{O2}-aeration and p_{O2}-gasmix. pH, temperature, and liquid weight all contain a single control mode. The user may select an automatic control mode or choose to operate the process manually from the drop-down menus located in the panels labeled with the respective controlled variables. The antifoam controller may be switched on and off, but it cannot be operated manually due to the reasons explained in chapter 3.4.1.

If a setpoint for a controlled variable is required, it can be changed at any time by entering the new value into the corresponding edit field to the right of the control mode selection. These edit fields are labeled with the name of the controlled variable and the index w. Next to the setpoint, the current actual value of the controlled variable is indicated. If any of these controlled variables are selected in the "Variable Pool" tab, the trend of their development will be assessed and displayed to the right of the current actual value. This is explained in detail in chapter 4.1.2.

Active controllers are indicated by a green light, manual modes are indicated by a red light left to the control mode selection. If a control mode is selected, the corresponding interface for the manual operation is grayed out. For example, if the pH control is turned on, acid and alkali pumps cannot be operated manually and the matching switches are disabled. The pH controller will then automatically turn acid or alkali addition on or off to reach the setpoint entered in the pHw edit field.

As the p_{O2}-control has multiple different modes, the values of the control variables that are not currently in use may be changed via SPCs. For instance, if the p_{O2} is controlled by the stirrer, the setpoints for the air and oxygen aeration rates can be set manually by the operator. The respective SPC will then effectuate the changes made. In the simulation at hand, this means that the actual value will be the same as the setpoint.

If expert mode is accessible, the gains of the different controllers can be changed by pressing the "Parameters" buttons located to the right in each panel. This will allow the user to change the values of the gains K_P, K_I, and K_D for each distinct controller if applicable. Once a "Parameters" button is pressed, a dialogue box opens displaying the currently valid values for each gain in accordingly labeled edit fields. Changes can be made by entering new values into the corresponding edit fields and pressing the "Confirm" button at the bottom of the dialogue box. The four distinct dialogue boxes for the pH, temperature, p_{O2}, and liquid weight controllers are illustrated in figure 18.

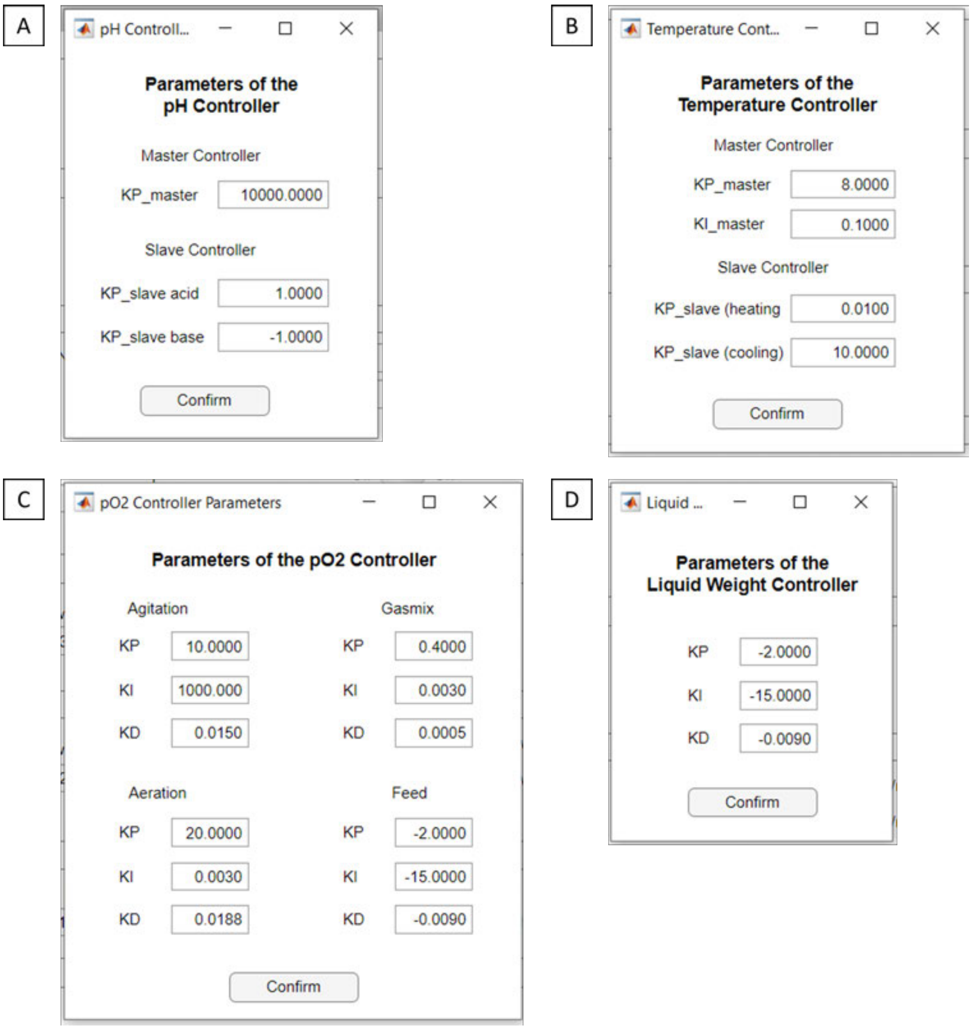


Figure 18: **Dialogue boxes related to changing the parameters of the pH, temperature, p_{O2} and liquid weight controllers.** A: Parameter dialogue box of the pH controllers. B: Parameter dialogue box of the temperature controllers. C: Parameter dialogue box of the p_{O2} controllers. D: Parameter dialogue box of the liquid weight controller.

4.1.2 Variable Pool

The second tab of the control app is the "Variable Pool" tab shown in figure 19. This tab gives an overview of the current values for a selection of the most interesting variables, such as the liquid volume V_L or cell concentration c_{XL}, in a trend table located in the middle of the window. It furthermore gives detailed information on the current flow rates affecting the liquid volume (acid (F_{T1}) and base (F_{T2}) titration, feed (F_R), and harvest (F_H)) and indicates how much acid, base, antifoam, and feed has already been added to the liquid volume. It also presents data related to the p_{O2}-control, namely current aeration rate (F_{nG}), stirrer speed (N_{St}), and oxygen mole fraction at the reactor inlet (x_{OGin}).

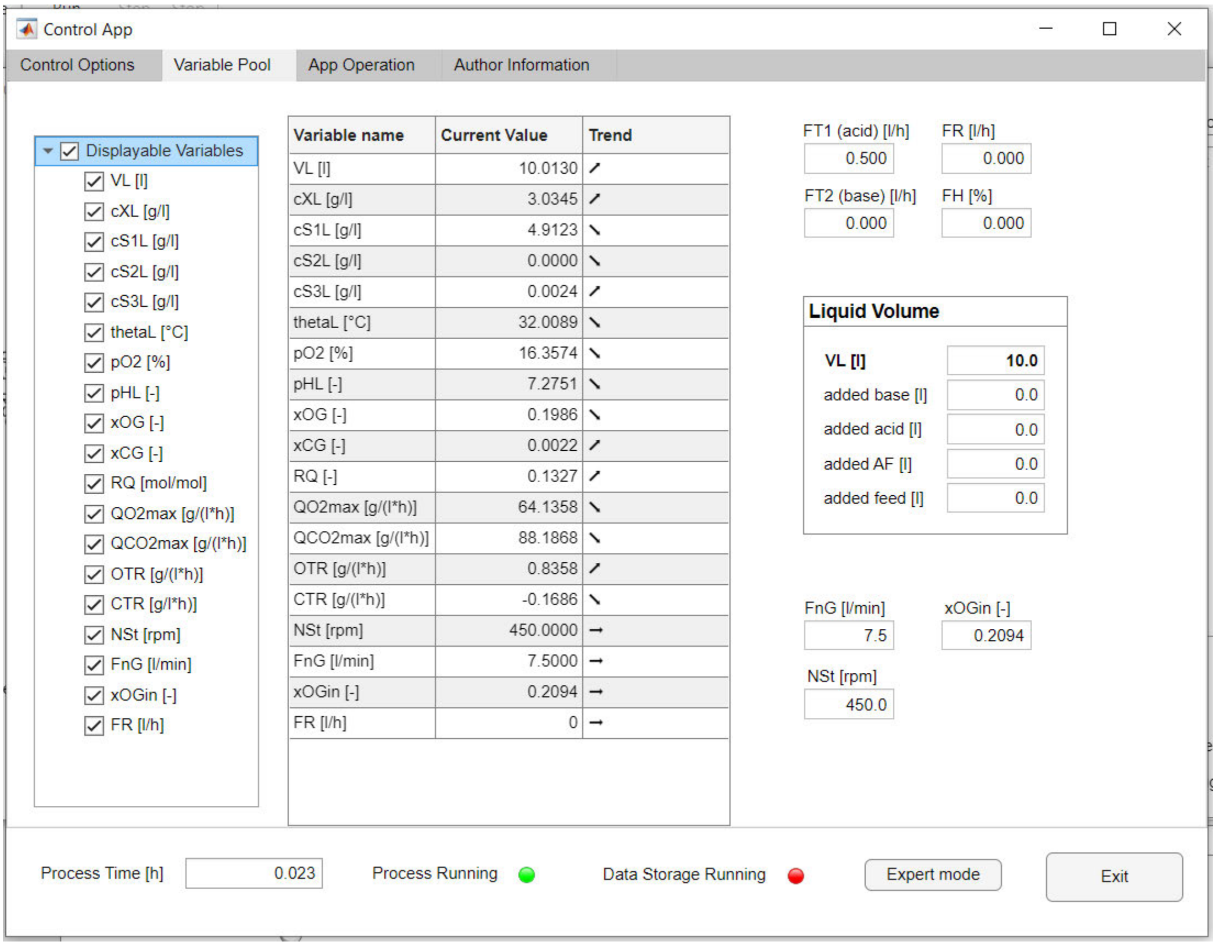


Figure 19: **"Variable Pool" tab of the control app.** This tab displays the most relevant variables in the variable pool that inform the operator about the status of the simulated process. The current values are displayed and for a selection of them, a trend table can be generated illustrating the general development of each specific variable.

The variables for which a trend table is to be generated may be selected by the operator from the list on the left side of the app window. Variables can be selected and unselected at will during the entire process. In the trend table, the variable name will be displayed as well as its current value and the trend at which the values are developing. If the values are increasing over time, the trend column will show an arrow pointing upwards. Likewise, if the values are decreasing over time, the trend column will display an arrow pointing downwards. If the value is not changing, the arrow will simply point to the right in a horizontal line. The trend table is refreshed each time the simulation has finished calculating the next set of variable entries or when the selection of variables has changed.

The trend is assessed by interpolating the last eleven entries in the variable history through a spline function. If there are not enough entries yet, all available entries will be interpolated instead. This spline will break down this last part of the variable history into a sequence of piecewise polynomials. The piecewise polynomial itself is then disassembled and the information about each polynomial is extracted. These data are in turn used to calculate the derivative of the spline fit. The value of the derivative at the latest position is then assessed. If the value is positive, the values of the variable are increasing, if it is negative, the values are decreasing and if the value is zero, no change is occurring. This method is used instead of simply calculating the difference between the last two variable entries to account for fluctuations and focus on larger trends.

4.1.3 App Operation

The third tab is the “App Operation” tab. This tab includes general functionalities, such as data storage start and stop, operations relating to starting variables, time increments after which new data are generated, and the start of the actual simulation either from the beginning or from a previously saved process. Additionally, this tab offers a plot functionality that lets the user plot previously generated data. It also displays the name of the file containing the starting variables if one was loaded. If a process was loaded or if previously generated data were plotted from an external file, the corresponding file name will also be displayed. The options given here are grouped by similar functionality.

The “App Operation” tab is shown in figure 20.

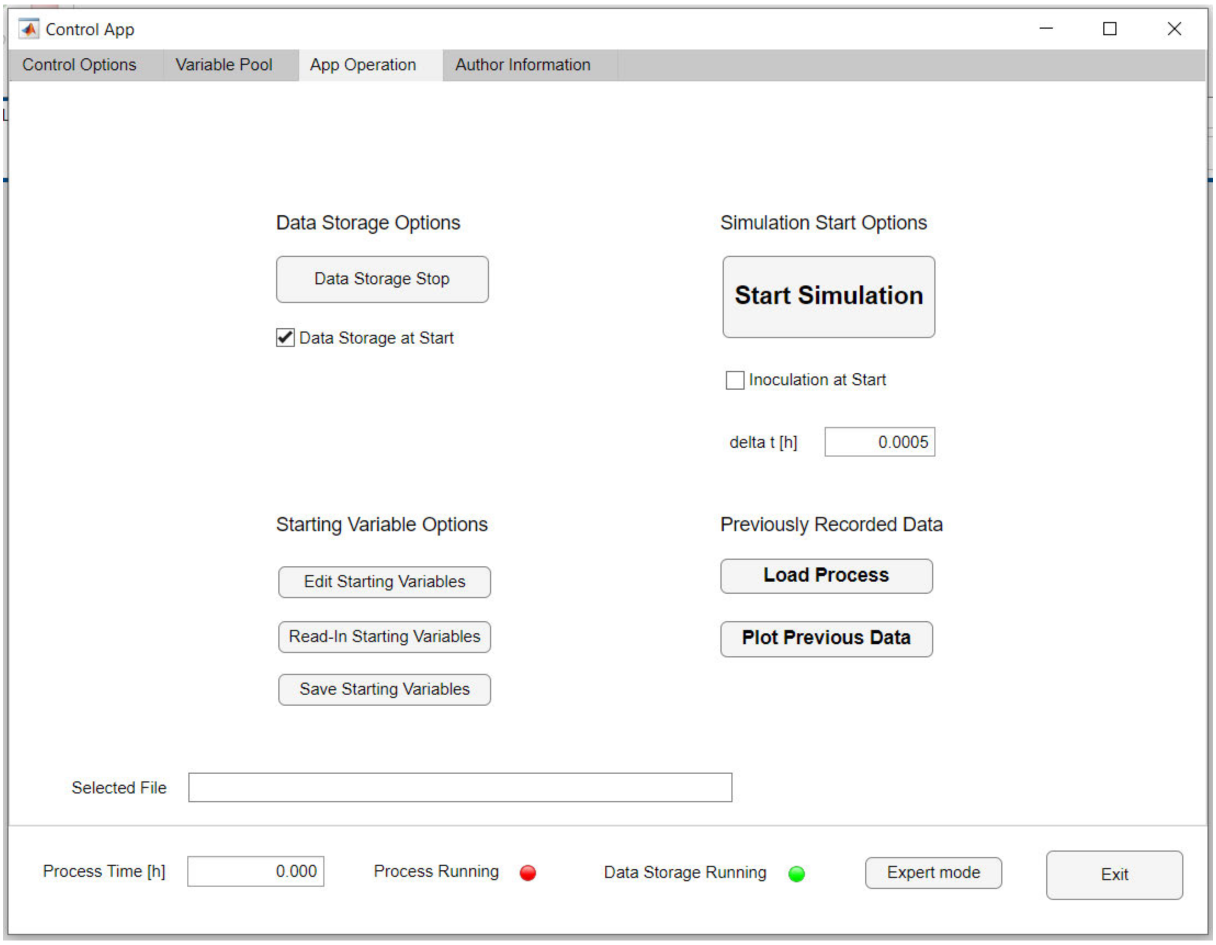


Figure 20: **"App Operation" tab of the control app.** This tab contains general functionalities related to the simulation, such as data storage, manipulation of starting variables and time increments as well as starting the actual simulation itself. It also displays the name of the loaded preset containing starting variables if one was selected or of a loaded process. The "Inoculation at Start" button gives the operator the option to inoculate the fermenter right at the start of the process. Processes can be loaded, for example, to continue a process that was started previously, and generated data can be plotted.

The data storage options include the manual start and stop of data storage as well as a checkbox related to starting data storage with the start of the simulation. Once data are being recorded, either when the“”Data Storage Start" button is pressed or when data are recorded from the start of the simulation, two files are generated automatically. The first file is a parameter file containing important parameters related to the simulation (initial cell and substrate concentrations (c_{XL0} and c_{S1L0}), chosen initial feed rate (F_{R0}), chosen initial stirrer speed (N_{St0}), chosen initial air aeration rate (F_{nAIR0}), maximum liquid volume (V_{Lmax}), substrate concentration in the reservoir (c_{S1R}), initial liquid volume (V_{L0}), and the time for which the fermentation is supposed to run (t_{max})). The other file is a data file that contains all generated data for the system time of the computer that runs the simulation (time), the simulated time (t), the cell concentration (c_{XL}), the substrate concentrations (c_{S1L}, c_{S2L}, and c_{S3L}), the p_{O2}, the temperature of the liquid volume (ϑ_L), the pH, the oxygen and carbon mole fractions in the gas phase (x_{OG} and x_{CG}), the liquid volume (V_L), the pressure (p_G), the overall aeration rate (F_{nG}), the feed rate (F_R), the stirrer speed (N_{St}), the growth rate (μ), and the molar concentration of ammonia (C_{AlLtot}).

Data storage ends with pressing the data storage button. While data storage is indicated by the light at the bottom of the app window turning green, it is also indicated by the text on the button changing from “Data Storage Start” to “Data Storage Stop”. The generated files start with “Data” or “Parameter” respectively and the date and time at which data storage was started.

The starting variable options contain all functionalities related to changing the values of the starting variables for the mathematical model within the simulation. The “Edit Starting Variables” button lets the user change the values for the initial cell concentration in the reactor at the time of inoculation (c_{XL0}), the initial glucose (substrate 1) concentration (c_{S1L0}), the maximum volume in the fermenter (V_{Lmax}), the concentration of glucose in the feed reservoir (c_{S1R}), the initial volume of the feed reservoir (V_{R0}) and the time for which the simulation is supposed to run (t_{max}). This is illustrated in figure 21.

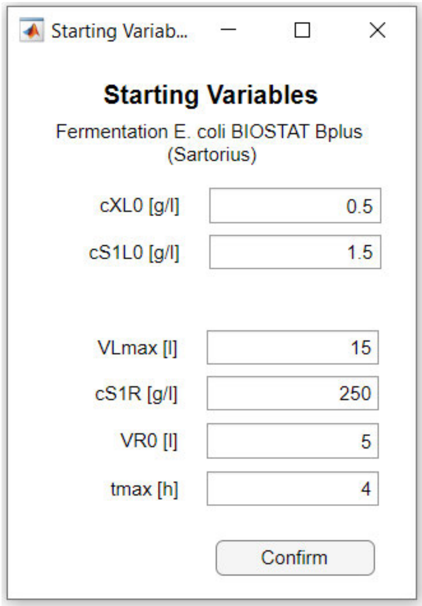


Figure 21: **Edit window enabling the simulation operator to change the starting variables.** Once the window is opened, it displays the currently chosen starting variables. Changing the values in the corresponding edit fields and pressing “Confirm” updates them. If starting variables are loaded from an external file, the name of the file will be displayed below the window title.

The values displayed upon opening the edit window are the values currently in use by the simulation. The user may change these values to any other value within the defined limits (c_{XL0}: 0.05-10 g/l, c_{S1L0}: 0-10 g/l, V_{Lmax}: 1-50 l, c_{S1R}: 20-700 g/l, V_{R0}: 0.1-10 l, t_{max}: 0.1-50 h). If new values were entered, they are only accepted by the simulation if the user presses the “Confirm” button. If the edit window is closed by pressing the “x” button in the top right corner, no changes will be made.

The starting variables may also be read in from an .xlsx or .txt file. If the “Read-In Starting Variables” button is pressed, a dialogue window will open prompting the user to select the file the variables are stored in (figure 22).

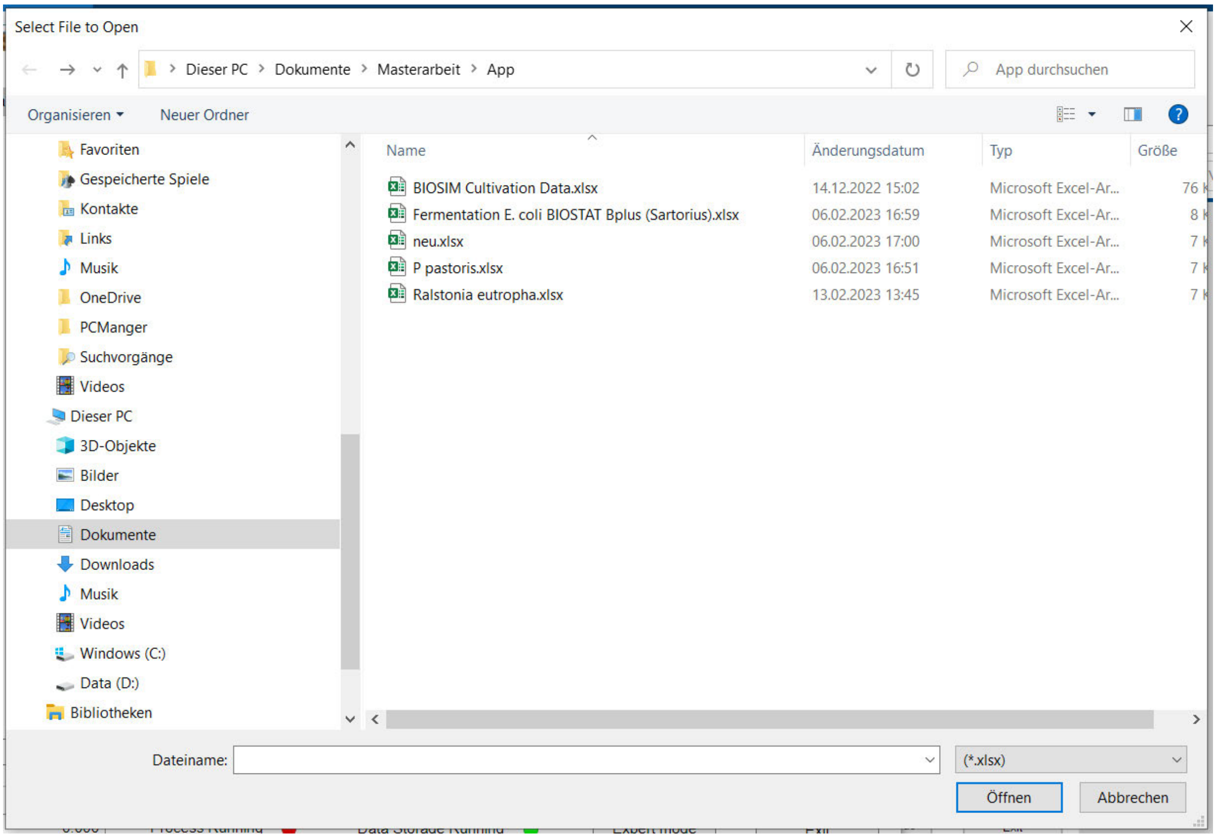


Figure 22: **Dialogue window for starting variable file selection.** This window opens when the user presses the "Read-In Starting Variables" button in the "App Operation" tab. The file types are pre-filtered to allow for .xlsx and .txt files.

The variables do not need to be saved in a specific order, as the program is written to sort the variables by name automatically, but they need to be saved in a specific format and with specific names: cXL0, cS1L0, FRw, NStw, FnGw, VLmax, cS1R, VR0, and tmax. If they are titled otherwise, they will not be recognized by the program. Once starting variables are loaded from an external file, the starting variable window will open displaying the updated values. The name of the loaded .xlsx or .txt file will be shown below the window title. This was seen in figure 21.

The variable names should be saved in one row and the corresponding variable values should be saved in the row below at the respective matching column. If an .xlsx file is chosen as the format, the table containing the variables does not need to start at column A or row 1. The table may be positioned anywhere in the file. If a .txt file is chosen as the format, the variable names should be written in the first row with the corresponding variable values in the second row. Each entry should be either tab- or comma-delimited. Examples of such .xlsx and .txt files are given in figures 23 and 24 respectively.

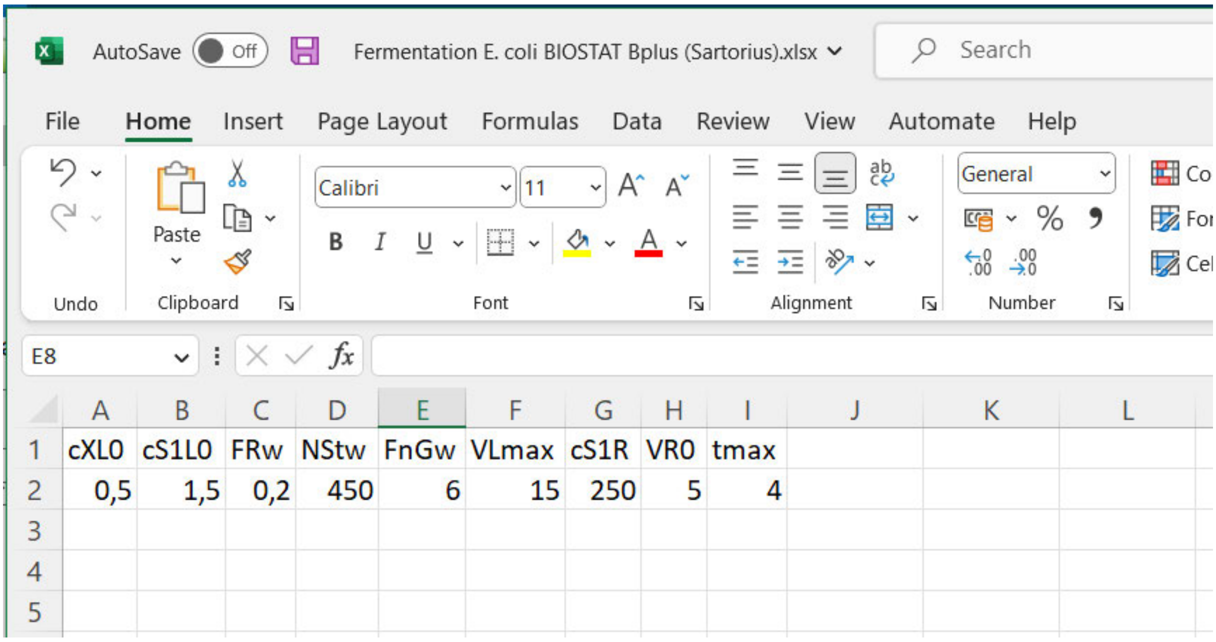


Figure 23: **Example .xlsx file containing the starting variables for the simulation at hand.** The first row should contain the variable names with the corresponding values written in the row below.

Figure 24: **Example .txt file containing the starting variables for the simulation.** The variable names and their corresponding values are tab-delimited.

MATLAB® will automatically recognize differing decimal notations in .xlsx files. This means that an .xlsx file containing the starting variables with commas as decimal separators can be loaded and MATLAB® will change the decimal separators to dots. In contrast, if the starting variables are loaded from a .txt file, MATLAB® will not automatically change the decimal separator and display a loading error instead. Hence, only .txt files containing starting variables written with dots as decimal separators may be loaded.

If the starting variables are to be saved, the operator can press the “Save Starting Variables” button. A dialogue window will open asking the user to choose a file to store the new starting variables in. If no such file exists, a new one can be created by clicking right in the dialogue window and selecting “New”, “Microsoft Excel File” or “Text File”. This is illustrated in figure 25. It should be noted that the table containing the new starting variables will be saved to an .xlsx file starting in column A and row 1. This means that, if an .xlsx file containing other starting variables was chosen and the original table containing the old starting variables was not located in column A and row 1, the new table will only partially override the old one. However, due to how this functionality is set up in the code, when the updated file containing partially overwritten old data is loaded again, the program will only read in the updated data and ignore any remnants.

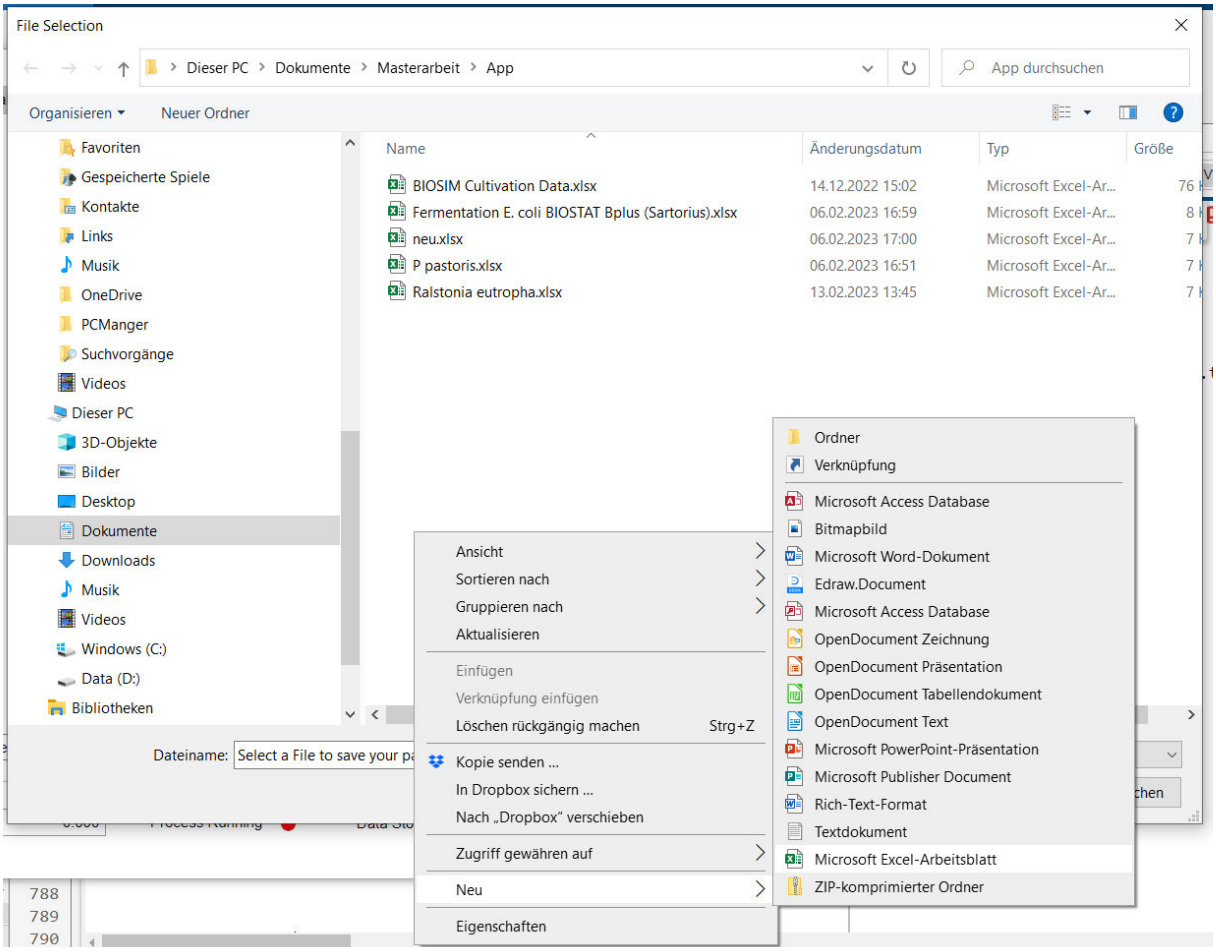


Figure 25: **Creation of a new .xlsx file within the dialogue window.** Similarly, a new .txt file may be created.

When a file is selected by pressing the “Read-In Starting Variables” or the “Save Starting Variables” button, the file name of the selected file is displayed in the edit field titled “Selected File”.

The simulation start options contain functionalities related to the general course of the simulation. The “Start Simulation” button launches the simulation app by calling its public *cycle_func* function. This function creates a timer object and sets the initial values for the variables that are calculated by the mathematical model. Once the simulation is running, the text on the button changes to “Stop Simulation” and pressing it again closes the simulation app. This functionality is similar to closing the simulation app by pressing the “Exit” button on the lower right or the close button at the top right corner of the simulation app. The “Inoculation at Start” checkbox, similarly to the “Data Storage at Start” checkbox, causes inoculation to take place at the start of the simulation with $t = 0$ h. The “delta t” edit field displays the currently valid setpoint for the time increment after which the simulation calculates new values. Per default, Δt is set to 0.0005 h or 1.8 s. This means that every 1.8 s the mathematical model is used to update the values for each variable. The smaller Δt, the higher the resolution of the simulated process and the better the accuracy of the generated data. However, at smaller values of Δt the simulation also becomes slower, as more data need to be generated to cover the same time span. Thus, if runtime is a factor, Δt should be higher for simulations covering a longer time span and smaller for simulations covering a shorter time span.

The previously recorded data options contain additional functionalities related to processes for which data have already been generated. If an ongoing process is saved through the simulation app, it can be loaded again by pressing the “Load Process” button and selecting the .m file it was saved as (figure 26). Saving a process is explained further in chapter 4.2.

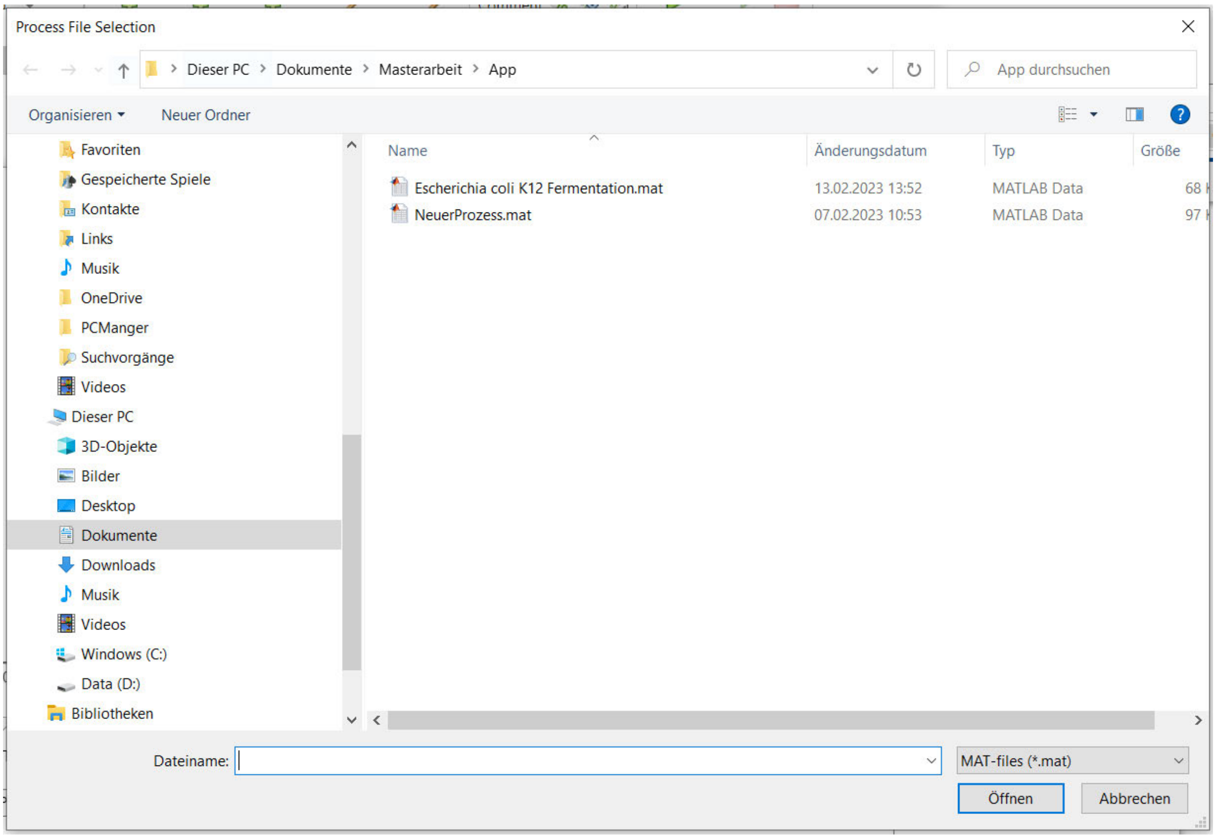
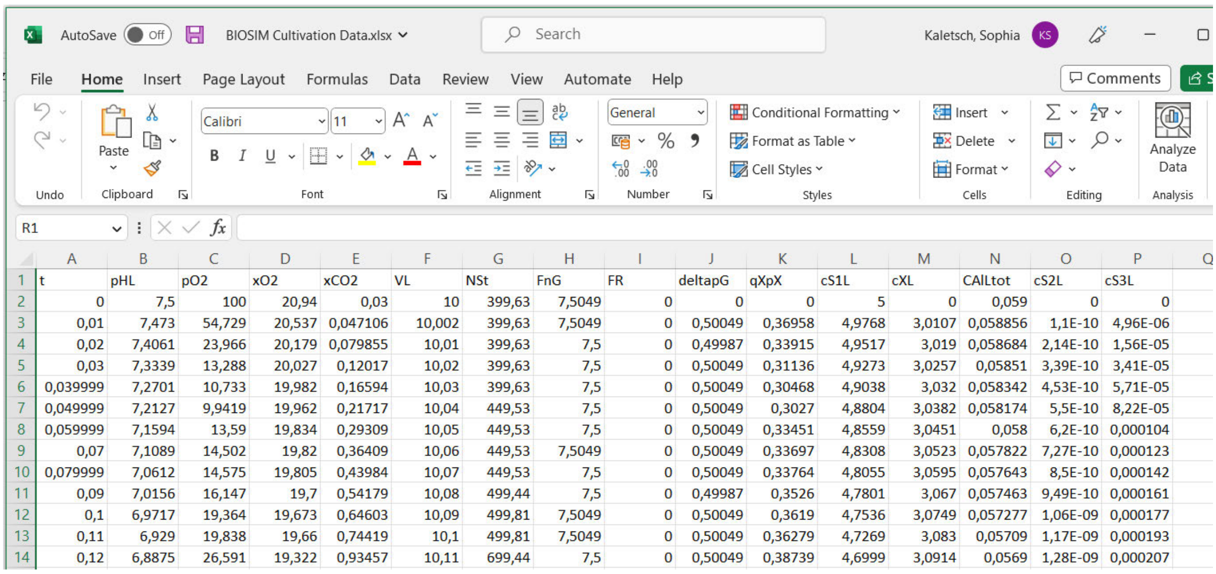


Figure 26: **Process file selection window.** This window lets the operator select a file containing a process to load and is opened upon pressing the “Load Process” button in the control app. The process file is a MATLAB® file (.mat).

This .m file is a MATLAB® file containing information about the generated data and the state of the simulation app, such as values displayed in edit field windows. Pressing the “Load Process” button furthermore opens the simulation app, which remains inactive until a file was selected. Then, all variables, properties, and property status information are assigned. The process is fully loaded once the edit fields in the figure app display their designated values and needs to be resumed by the operator by pressing the "Resume" button. The process does not start running immediately, allowing the user to make any adjustments. Notably, loading a process will not make changes to the selected controllers, but will change the setpoints for the SPCs to their last values in the loaded process. The name of the loaded process will be displayed in the "Selected File" edit field of the control app and the title of the generated plot in the figure app. The speed of the simulation will be real-time upon loading a process, similarly to starting a new process.

Furthermore, previously generated data can be loaded and plotted in the simulation app by pressing the "Plot Previous Data" button in the control app. The origin of the data, i.e. generated by the MATLAB®-based simulation or any other program, is irrelevant. However, the data file (either .xlsx or .txt file) needs to follow a specific structure. The variable names have to be written in the upmost row with the respective values stored in the rows below but in the same column. This parallels how the MATLAB®-based simulation saves data. The order in which the variables are saved is not of importance, but they should be named after their corresponding property in the MATLAB®-based simulation. For example, the property in which the temperature of the liquid phase is stored is named "thetaL". This name must be used in the file that contains the plottable data or the variable will not be recognized. This is illustrated in figures 27 and 28 for an .xlsx and a .txt file respectively.



t	pHL	pO2	xO2	xCO2	VL	NSt	FnG	FR	deltapG	qXpX	cS1L	cXL	CAlLtot	cS2L	cS3L
0	7,5	100	20,94	0,03	10	399,63	7,5049	0	0	0	5	0	0,059	0	0
0,01	7,473	54,729	20,537	0,047106	10,002	399,63	7,5049	0	0,50049	0,36958	4,9768	3,0107	0,058856	1,1E-10	4,96E-06
0,02	7,4061	23,966	20,179	0,079855	10,01	399,63	7,5	0	0,49987	0,33915	4,9517	3,019	0,058684	2,14E-10	1,56E-05
0,03	7,3339	13,288	20,027	0,12017	10,02	399,63	7,5	0	0,50049	0,31136	4,9273	3,0257	0,05851	3,39E-10	3,41E-05
0,039999	7,2701	10,733	19,982	0,16594	10,03	399,63	7,5	0	0,50049	0,30468	4,9038	3,032	0,058342	4,53E-10	5,71E-05
0,049999	7,2127	9,9419	19,962	0,21717	10,04	449,53	7,5	0	0,50049	0,3027	4,8804	3,0382	0,058174	5,5E-10	8,22E-05
0,059999	7,1594	13,59	19,834	0,29309	10,05	449,53	7,5	0	0,50049	0,33451	4,8559	3,0451	0,058	6,2E-10	0,000104
0,07	7,1089	14,502	19,82	0,36409	10,06	449,53	7,5049	0	0,50049	0,33697	4,8308	3,0523	0,057822	7,27E-10	0,000123
0,079999	7,0612	14,575	19,805	0,43984	10,07	449,53	7,5	0	0,50049	0,33764	4,8055	3,0595	0,057643	8,5E-10	0,000142
0,09	7,0156	16,147	19,7	0,54179	10,08	499,44	7,5	0	0,49987	0,3526	4,7801	3,067	0,057463	9,49E-10	0,000161
0,1	6,9717	19,364	19,673	0,64603	10,09	499,81	7,5049	0	0,50049	0,3619	4,7536	3,0749	0,057277	1,06E-09	0,000177
0,11	6,929	19,838	19,66	0,74419	10,1	499,81	7,5049	0	0,50049	0,36279	4,7269	3,083	0,05709	1,17E-09	0,000193
0,12	6,8875	26,591	19,322	0,93457	10,11	699,44	7,5	0	0,50049	0,38739	4,6999	3,0914	0,0569	1,28E-09	0,000207

Figure 27: **Exemplary .xlsx file containing generated bioprocess data that can be plotted using the MATLAB®-based app.** The first row should contain the variable names and the following rows the respective data.

FedBatch_BIOSIMcomparison.txt - Editor

Datei Bearbeiten Format Ansicht Hilfe

t	cXL	cS1L	cS2L	cS3L	pO2	thetaL	pHL	xOG	xCG	RQ	QO2max	QCO2max	OTR	CTR	qXpX	CAlltot
0.00	3.00	5.00	0.00	0.0000100800	98.54	32.00	7.45	0.21	0.00	4.91	64.25	88.34	0.04	-0.03	0.40	0.06
0.00	3.00	5.00	0.00	0.0000205909	96.96	32.00	7.45	0.21	0.00	0.43	64.25	88.34	0.08	-0.03	0.39	0.06
0.00	3.00	4.99	0.00	0.0000315534	94.97	32.01	7.44	0.21	0.00	0.36	64.24	88.34	0.11	-0.03	0.39	0.06
0.00	3.00	4.99	0.00	0.0000429871	92.68	32.01	7.44	0.21	0.00	0.29	64.24	88.33	0.14	-0.03	0.39	0.06
0.00	3.00	4.99	0.00	0.0000549132	90.16	32.01	7.44	0.21	0.00	0.24	64.24	88.33	0.17	-0.04	0.39	0.06
0.00	3.00	4.99	0.00	0.0000673542	87.49	32.01	7.43	0.21	0.00	0.21	64.24	88.33	0.20	-0.04	0.39	0.06
0.00	3.00	4.99	0.00	0.0000803338	84.72	32.01	7.43	0.21	0.00	0.18	64.24	88.33	0.23	-0.04	0.39	0.06
0.00	3.00	4.99	0.00	0.0000938763	81.88	32.01	7.43	0.21	0.00	0.17	64.24	88.33	0.25	-0.04	0.39	0.06
0.00	3.01	4.99	0.00	0.0001080077	79.01	32.01	7.43	0.21	0.00	0.15	64.24	88.32	0.28	-0.04	0.39	0.06
0.01	3.01	4.99	0.00	0.0001227548	76.13	32.01	7.42	0.21	0.00	0.14	64.23	88.32	0.30	-0.04	0.39	0.06
0.01	3.01	4.98	0.00	0.0001381458	73.28	32.01	7.42	0.21	0.00	0.13	64.23	88.32	0.33	-0.04	0.39	0.06
0.01	3.01	4.98	0.00	0.0001542101	70.45	32.01	7.42	0.20	0.00	0.12	64.23	88.32	0.35	-0.05	0.39	0.06
0.01	3.01	4.98	0.00	0.0001709785	67.68	32.01	7.41	0.20	0.00	0.12	64.23	88.31	0.37	-0.05	0.39	0.06
0.01	3.01	4.98	0.00	0.0001884828	64.96	32.01	7.41	0.20	0.00	0.11	64.23	88.31	0.39	-0.05	0.39	0.06
0.01	3.01	4.98	0.00	0.0002067566	62.30	32.01	7.41	0.20	0.00	0.11	64.23	88.31	0.41	-0.05	0.39	0.06
0.01	3.01	4.98	0.00	0.0002258343	59.72	32.01	7.40	0.20	0.00	0.11	64.22	88.31	0.43	-0.05	0.39	0.06
0.01	3.01	4.98	0.00	0.0002457521	57.21	32.01	7.40	0.20	0.00	0.10	64.22	88.31	0.45	-0.05	0.38	0.06

Figure 28: **Exemplary tab-delimited .txt file containing generated bioprocess data that can be plotted using the MATLAB®-based app.** The first row should contain the variable names and the following rows the respective data.

This structure is necessary as the MATLAB®-based simulation reads in the table saved in the file and assigns the stored values to properties with the name of the column headers. For example, the values saved in the first column in this example will be saved to a property named “t”, as this is the header of the first column.

4.1.4 Author Information

The fourth tab of the control app is the “Author Information” tab. This tab has no impact on the simulation itself, but contains information regarding the author, licensing of the app source code, and acknowledgments concerning the mathematical model. It is displayed in figure 29.

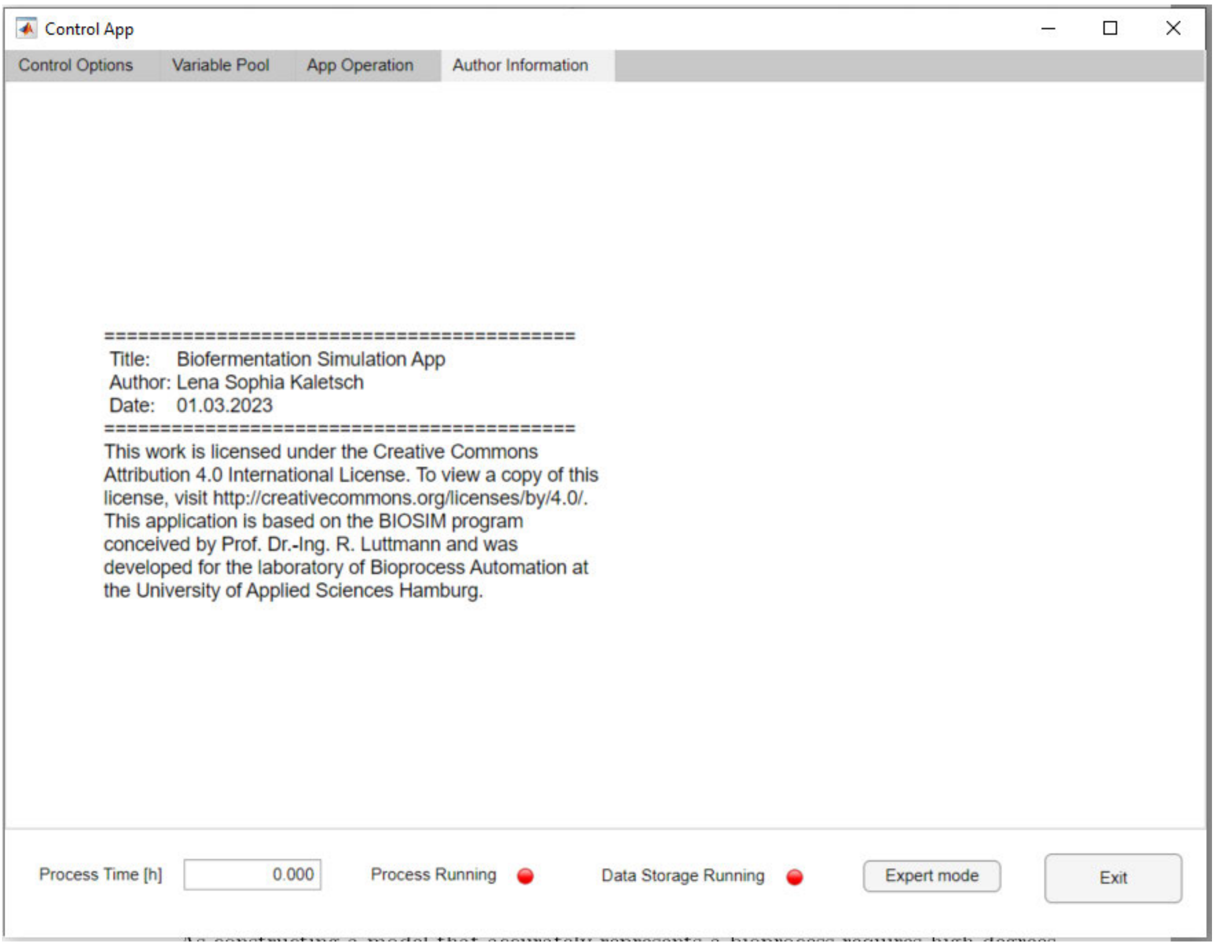


Figure 29: **"Author Information" tab of the control app.** This tab gives information on the author, licensing of the app source code, and acknowledgments regarding the mathematical model.

When the control app is closed, the SISO controllers selected at the time of closing, their setpoints, and the corresponding SPC setpoints as well as the selected checkboxes in the "App Operation" tab are saved to an external .txt file. This file is named "Control-AppOptions.txt" and is used to restore these values once the control app is launched again. If this file does not exist, the control app will launch with default values.

4.2 The simulation app

The simulation app displays the development of the process in the form of a dynamically configurable plot. It also contains a slider controlling the speed at which the simulation is running. The app layout as it appears in expert mode is shown in figure 30.

The simulation app receives input arguments from the control app and thus cannot be launched independently. The starting variables and parameters related to the process control and the mathematical model are saved in the control app as public properties and can be accessed by the simulation app. The relevant variables for which new values are calculated in the model, such as cell density or substrate concentration in the liquid phase, are saved as public properties in the simulation app.

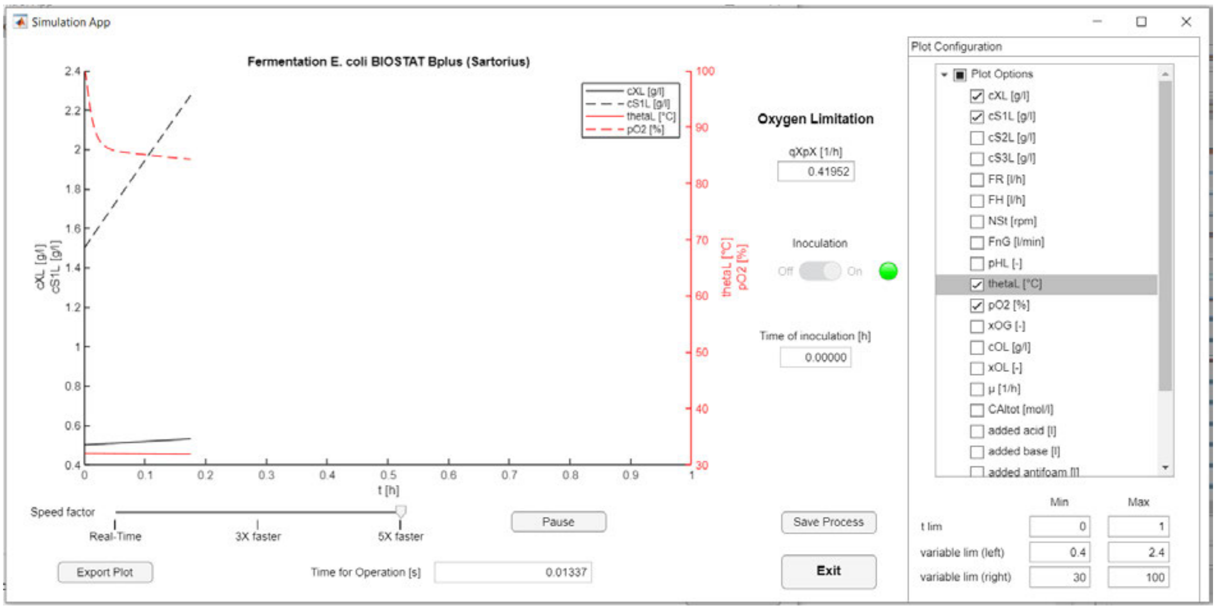


Figure 30: **Layout of the simulation app in expert mode.** The simulation app contains a plot of the variables chosen by the operator and a slider to adjust the simulation speed. It furthermore indicates the time it takes for the calculation of new variables in the time of operation edit field, indicates what type of limitation is the defining one (only visible in expert mode), gives the user the option to inoculate the fermenter, and records the time of inoculation. It also indicates the current growth rate numerically.

Once the simulation is started, a timer object is created which fires repeatedly in intervals specified by the value of Δt. The timer can be paused with the “Pause” button located next to the speed slider. Whenever the timer is triggered, it calls the function *TimerFcn*, which itself calls both the *calculation_func* and the *plot_func* functions. The *calculation_func* function contains the process control equations and those of the mathematical model and calculates the new entries for each process variable. The calculations are based on the values of the previous variable entry and the currently selected process control. For the values interesting to the operator in the given context, the new values are added to the end of a steadily growing array. This means that these variables are defined as public properties and all of their values are saved in a variable history. This history is used to calculate the trends indicated in the “Variable Pool” tab of the control app and for the generation of the plot shown in the simulation app. The *plot_func* function uses the data generated by the *calculation_func* function to draw a plot. Which variables are plotted can be changed in the “Plot Configuration” panel located on the right side of the simulation app window. Changing the variables updates the entire plot and changes the legend entries as well as the y-axis labels.

The “Inoculation” switch is enabled if the operator did not activate the “Inoculation at Start” check box, which causes the fermenter to be inoculated at the start of the process when t = 0 h. The “Inoculation” switch can be flipped at any point during the simulation and the cell concentration in the liquid will be immediately set to the initial cell concentration chosen by the operator. Once the fermenter is inoculated, the lamp next to the switch will turn green and the switch itself will be disabled. Additionally, the time of inoculation is recorded in the “Time of Inoculation [h]” edit field.

The displayed plot can be saved as a JPG, PNG, or TIF in picture and as a PDF file in vector format at any time during the process by clicking the ”Export Plot” button below the “Plot Configuration” panel. If a specific preset was loaded, the preset name will automatically be suggested as the file name, supplemented by the current date and time. This is shown in figure 31.

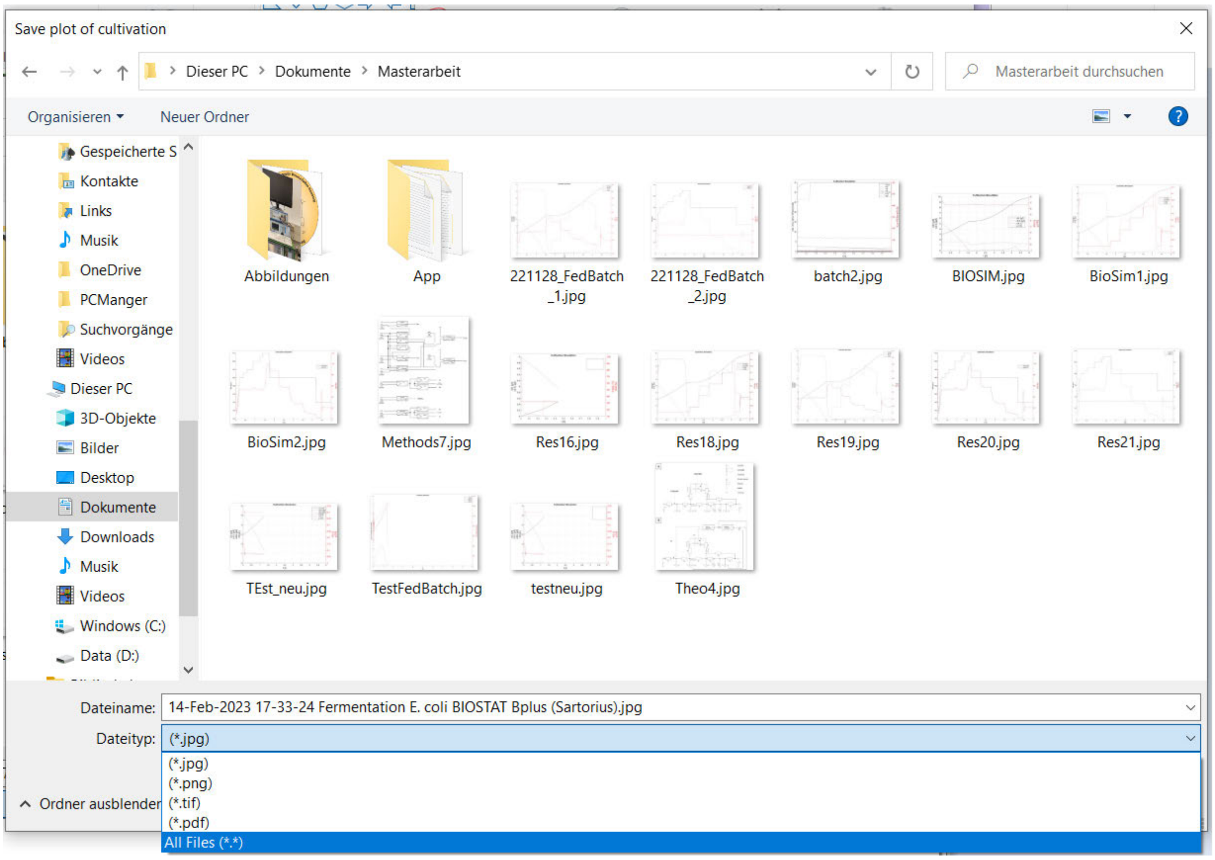


Figure 31: **Dialogue window opening once the operator presses the “Export Plot” button.** The file name will be suggested automatically as the preset if one was loaded, supplemented by the current date and time. The plot can be saved as a JPG, PNG, TIF, or PDF file.

Once the file was saved, a confirmation window will appear telling the operator that saving was successful (figure 32).

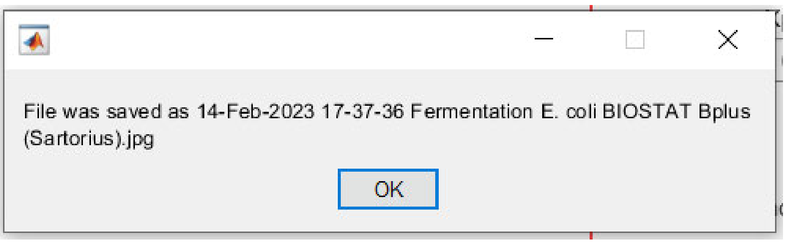


Figure 32: **Confirmation window notifying the operator that the plot was exported successfully under the chosen file name.**

An example of a plot saved as a JPG file is given in figure 33.

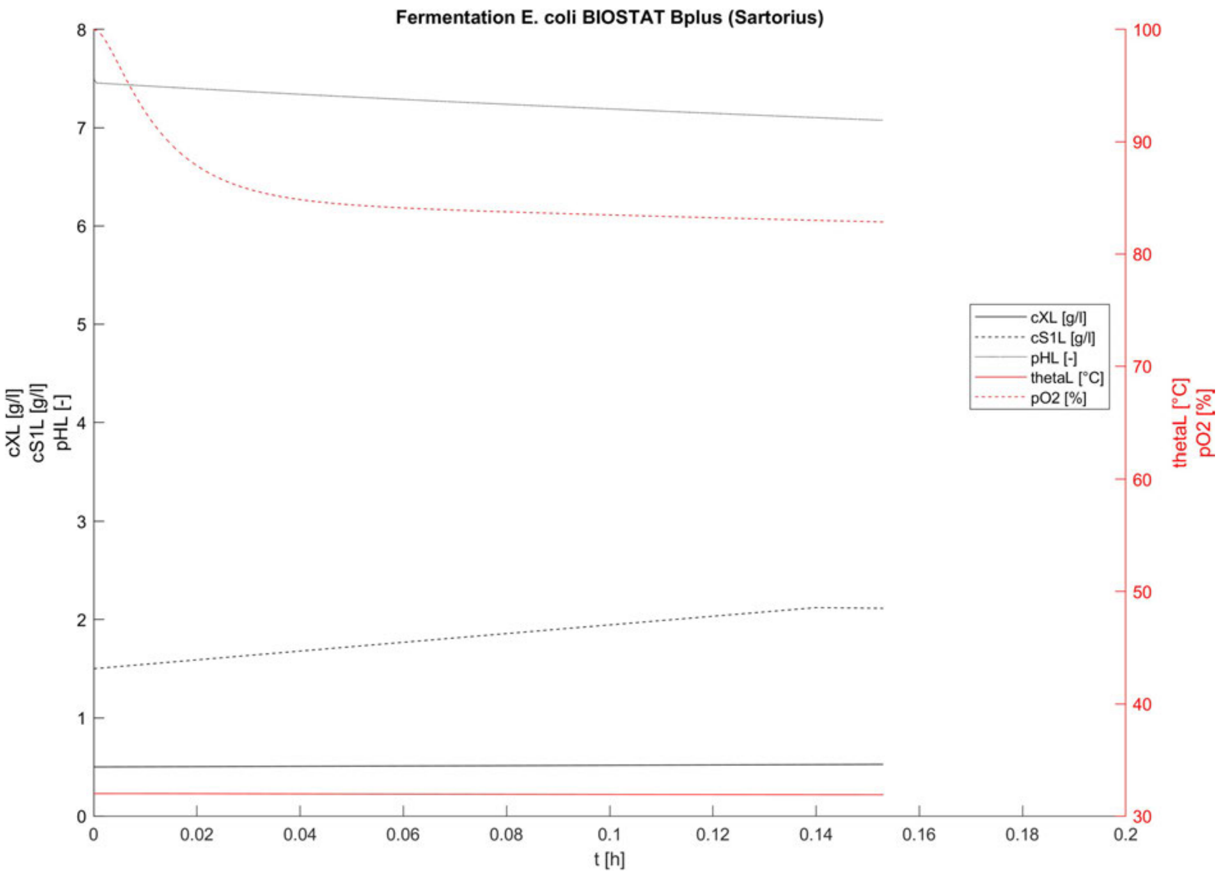


Figure 33: **Exemplary plot exported during a simulated fermentation process.** The plotted variables are dynamically selectable.

If the file is exported as a vector, lines can be adjusted retroactively in programs such as the open-source software Inkscape (Inkscape-Project).

The simulation will terminate automatically after t_{max} was reached. Alternatively, if the liquid volume exceeds V_{Lmax} during the process, the operator will be warned and the simulation will be paused. The operator then has the option to dismiss the warning and either continue the process despite having reached V_{Lmax} or terminate it. This is illustrated in figure 34.

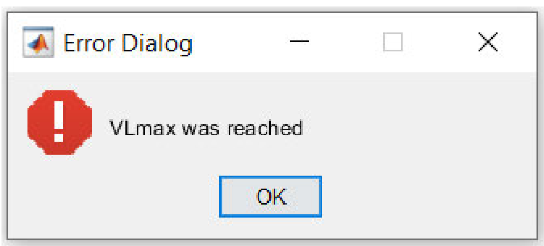


Figure 34: **Window opening to warn the operator that V_{Lmax} was reached.** This pauses the simulation.

4.3 Comparison with BIOSIM

In the following chapter, the MATLAB®-based simulation developed in this work and the BIOSIM program will be compared regarding setup requirements, operation, and finally their generated results. The proficiency of the new simulation to take over the role of BIOSIM will be assessed and significant discrepancies will be discussed.

4.3.1 Setup and operation

The BIOSIM on-line simulation system is set up to run on the operating system RTOS-UH and communicates with the DCU via the process control system UBICON. BIOSIM and UBICON run in parallel and communicate by means of a shared memory area, the UBICON data pool. Both UBICON and BIOSIM are programmed in the high-level language PEARL. This means that the process control is externalized from the simulation and thus, specific hardware is required to ensure full functionality. As the controller operates independently from the simulation, it starts calculating error signals even when the simulation is not running. Thus, for controllers containing an integral part, the integral error will be calculated and summed up continuously since the difference between the setpoint and actual value will stay static. Hence, specific controllers can only be switched on while the simulation is running, as otherwise the controller would disproportionately affect the control variable. As there is only one set of hardware and software available, the number of students that can work with the BIOSIM system at a time is limited. An advantage, however, is the flexibility of this setup regarding exchanging the entire model itself. It is possible to load and compile other mathematical models easily and quickly while not being bound to specific variables. Additionally, operating the simulation via a DCU teaches the students to use such hardware.

In contrast, the simulation established in this work is self-contained. If the app is compiled, not even MATLAB® is required to run it. Furthermore, it may be duplicated indefinitely and can be installed and operated on any computer. This means that copies may be passed onto the students and each student is able to operate their own bioprocess simulation. This can be done at the university or at home. Additionally, the simulation established in this work is easier and quicker to launch and operate. It is also more robust regarding possible errors, as fail-safes, for instance regarding accidental termination or the plausibility of starting variables, have been built into the application. It can be paused and sped up without loss of functionality and the model does not need to be compiled anew if it is changed. The process control is integrated into the mathematical model, which means that errors resulting from continuously operating controllers outside of the simulation do not occur. Furthermore, new presets may be created by changing variables manually or by loading them from an external .txt or .xlsx file. Moreover, the MATLAB® source code of the program is made available to the students, which means that they are able to understand and modify its contents. Currently, exchanging the mathematical model for an entirely different one is not as quick for the MATLAB®-based simulation as it is for BIOSIM. While exchanging the equations of the mathematical model is straightforward, new input and output variables or needed parameters need to be defined manually before they can be referred to in the mathematical model. The same is true for the process control.

4.3.2 Results of the simulation

To further assess the suitability of the simulation at hand to take over the role of the BIOSIM program, the preset of variables used in the BPA special course was loaded in both BIOSIM and the MATLAB®-based simulation. Every parameter was compared between both programs to allow for the best possible assessment. Both simulations were run in parallel, and any control options were employed simultaneously and in the same capacity. After both simulations had run their course, the generated data were saved and plots were generated with the MATLAB®-based simulation to compare process development and differences.

It should be noted that the MATLAB®-based simulation established in this work records more data and thus allows for more variables to be plotted. Hence, the plots that were generated with the MATLAB®-based simulation include the variables x_{OL}, the oxygen mole fraction in the liquid phase, the pH, and ϑ_L.

The plots are illustrated in figures 35 through 38.

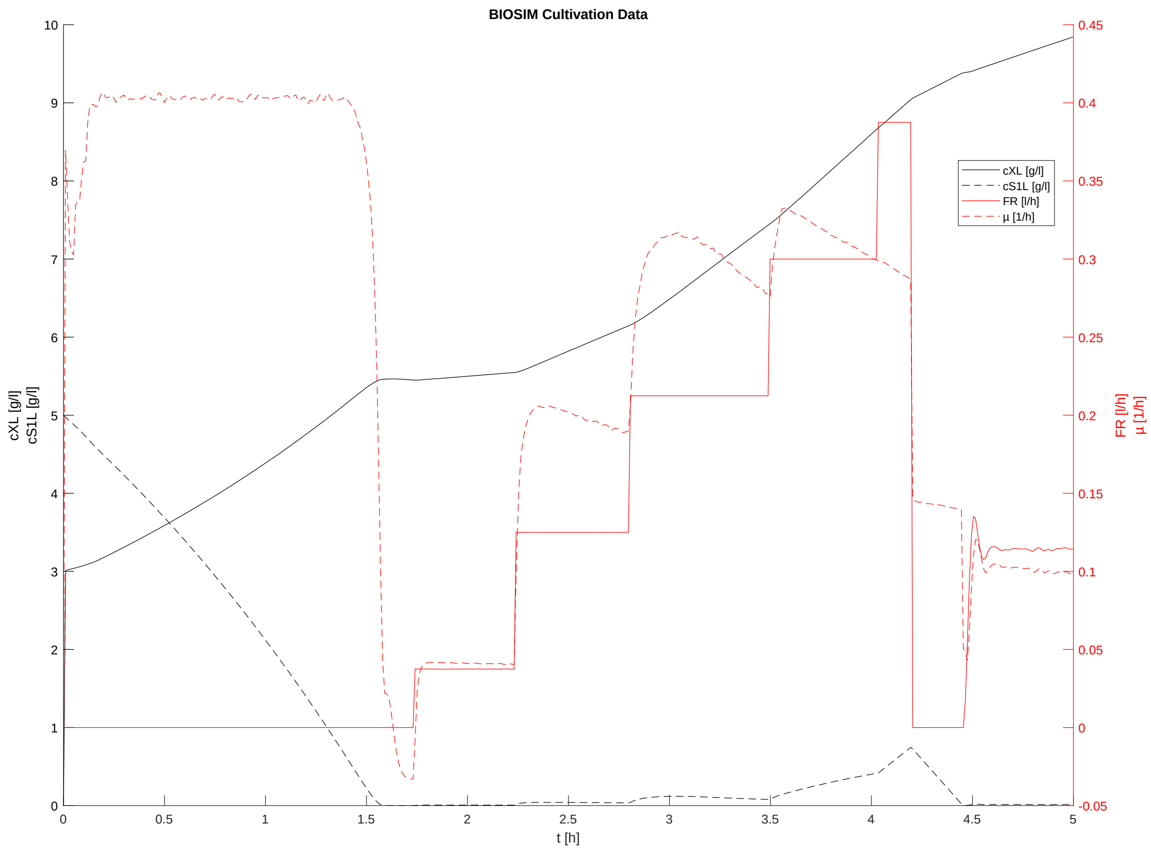


Figure 35: **First plot generated by BIOSIM.** It shows the cell concentration c_{XL} (black), glucose concentration c_{S1L} (black dashed), glucose feed rate F_R (red), and growth rate μ (red dashed).

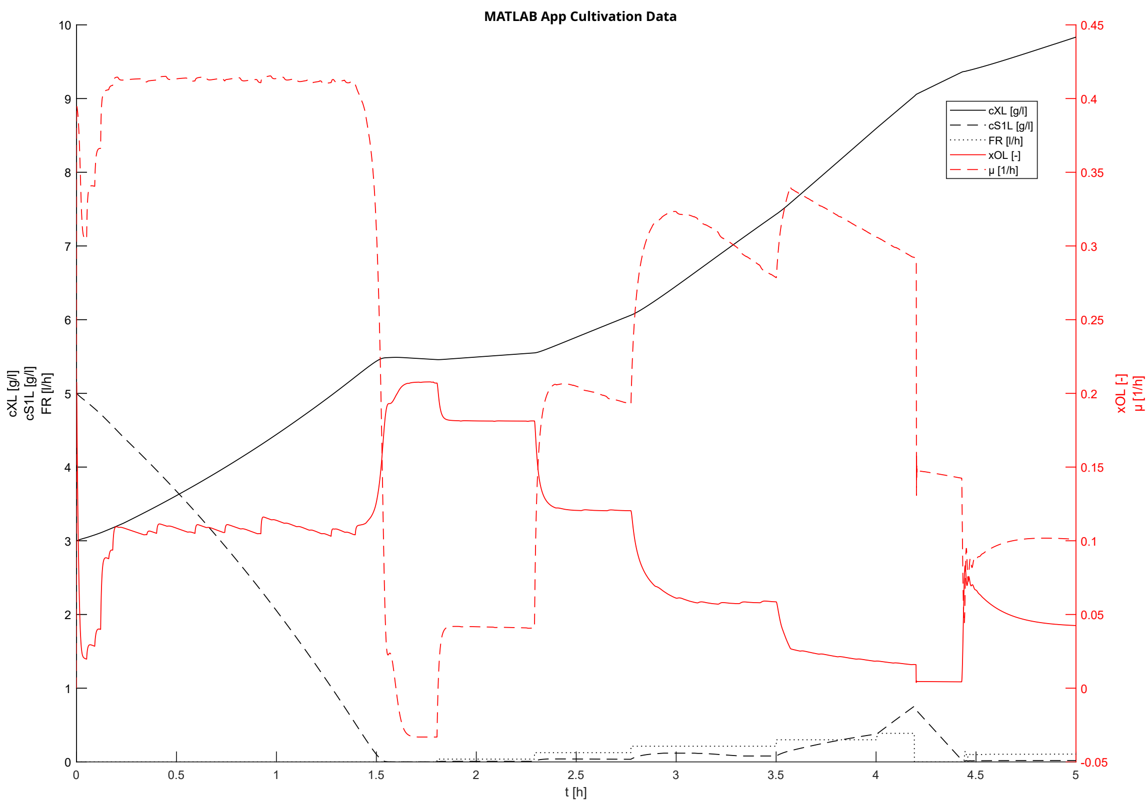


Figure 36: **First plot generated by the MATLAB®-based simulation.** It shows the cell concentration c_{XL} (black), glucose concentration c_{S1L} (black dashed), glucose feed rate F_R (black dotted), oxygen mole fraction in the liquid phase x_{OL} (red), and growth rate μ (red dashed).

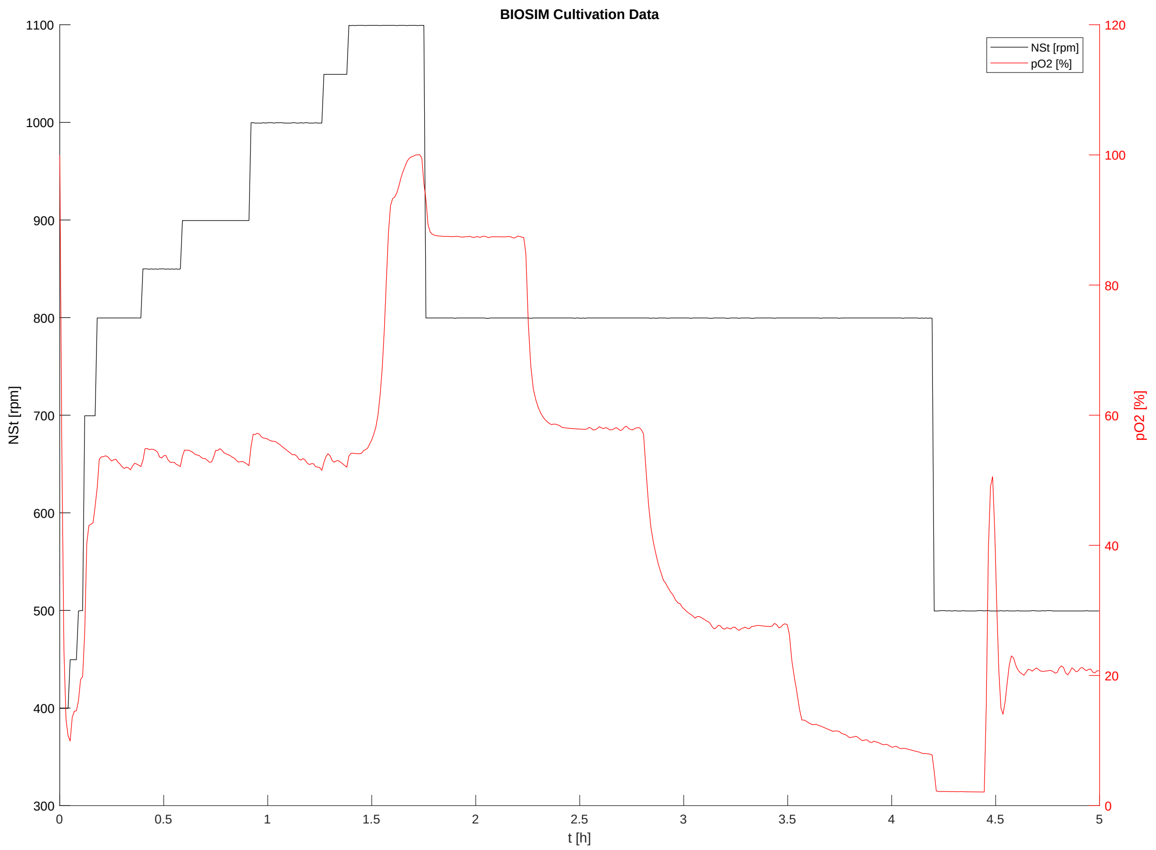


Figure 37: **Second plot generated by BIOSIM.** It shows the stirrer speed N_{St} (black) and the oxygen partial pressure p_{O2} (red).

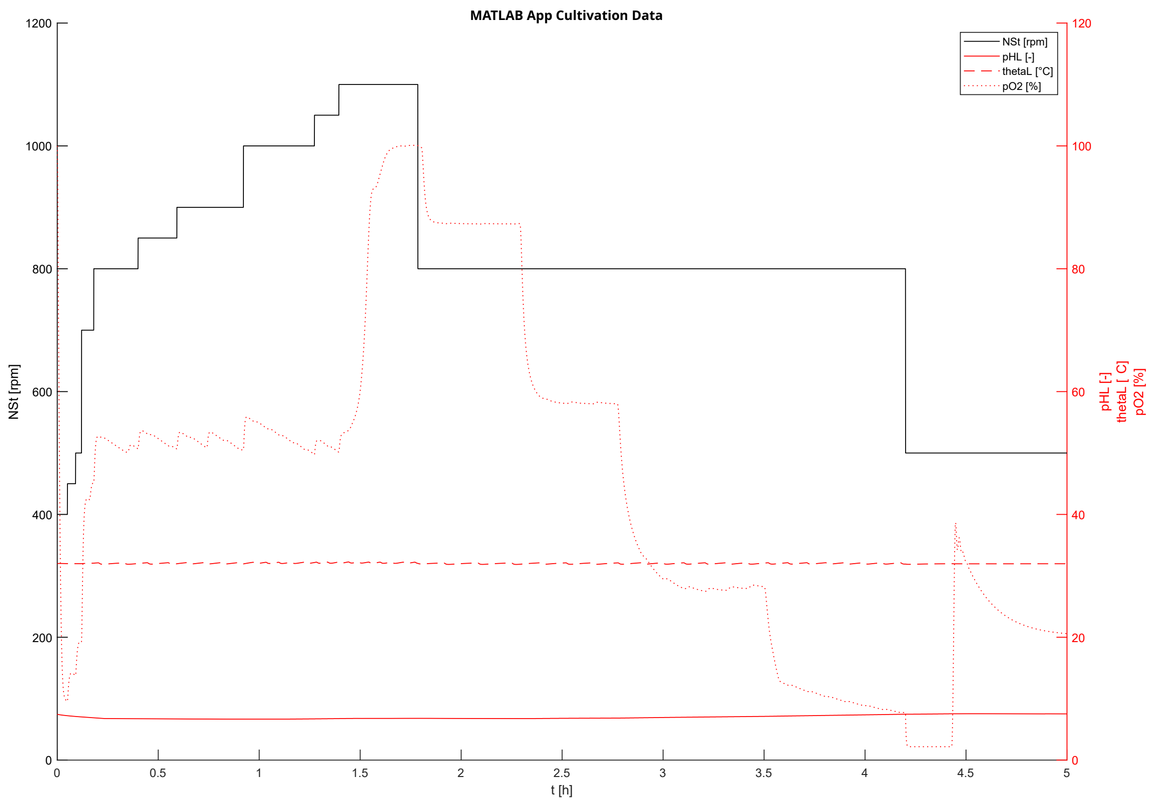


Figure 38: **Second plot generated by the MATLAB®-based simulation.** It shows the stirrer speed N_{St} (black), the pH (red), the temperature of the liquid phase ϑ_L (red dashed), and the oxygen partial pressure p_{O2} (red dotted).

In both simulations, exponential cell growth can be observed in the batch phase, which comes to a halt once the substrate glucose is used up. The growth rate also shows a similar development: it starts around 0.4 1/h and decreases to about 0.3 1/h in the first few minutes. Afterward, it rises again in a stepwise notion. This coincides with the oxygen availability in the liquid phase. It starts at 100 % p_{O2}, which allows for high values of the growth rate μ. While the oxygen is depleted by the cells, oxygen starts to become the limiting factor and μ decreases as well. Once the stirrer speed and aeration rate are increased manually to keep the p_{O2} at about 50 %, oxygen ceases to be the limiting factor. During this period in the batch phase, μ approaches μ_{1max}. Interestingly, both μ curves show the same behavior. Instead of a stable course, μ fluctuates mildly. This is due to the mathematical model. μ is directly calculated from the available substrate or oxygen supply and fluctuates with the temperature or pH as well. As such, minimal changes in environmental conditions are immediately reflected in the course of μ. Since the controllers regulating the pH and the temperature are not capable of keeping their controlled variables stable to the last decimal point and the addition of base, acid, or antifoam influence the concentration of substrate in the liquid phase, these fluctuations cannot be avoided.

Shortly after the 1.5 h mark, the available glucose is used up and the batch phase ends. This causes μ to decrease strongly and, in turn, the p_{O2} to increase as less oxygen is used up. Additionally, in both simulated processes, the μ curve shows the characteristic switch from the metabolization of glucose to acetate. After μ has decreased to about 0.02 1/h, it briefly stops its descent as the previously produced acetate is used up. Once no substrate is available anymore, μ decreases again until it is stabilized below zero as the cells start dying. This is also reflected in the p_{O2} curve: once the cells start taking up acetate, the steep increase of the p_{O2} is halted and proceeds more moderately. Once all available substrate is used up, the p_{O2} curve again increases steeply until the 100 % value is reached, where it then settles. During this time, the cell concentration only increases minimally, which is reflected in the respective curve.

At the 1.75 h mark, the fed-batch is started at 7.5 % of the maximum feed rate or 0.0375 l/h while the stirrer speed is set to 800 rpm and the aeration rate is set to 10 l/h. In both simulated processes, this causes only a slight increase in the glucose concentration, as most of the glucose entering the system is used up by the cells immediately. Furthermore, the growth rate steeply increases to about 0.025 1/h and remains at that overall level, though a slight decrease over time is observable. Consequently, the p_{O2} shows a sharp decrease to about 85 %, where it then stabilizes as the now multiplying cells start taking up oxygen again. As a result, c_{XL} shows an increase as well.

After about 30 min, at the 2.25 h mark, the feed rate is increased again to 25 % or 0.125 l/h. Similarly to the behavior observed previously, μ jumps to 0.2 1/h whereas the p_{O2} drops to 60 %. As the supply of fresh glucose is now higher, more substrate accumulates in the liquid phase and the glucose concentration increases as well. However, once the growth rate has reached its new peak, it decreases again over time. This can be explained by the cells growing rapidly and taking up the available substrate, causing the glucose concentration in the liquid phase to decline. Since the growth rate in this phase is ruled by the substrate concentration and the substrate growth rate is directly mathematically determined based on the available substrate in the liquid phase, a lower substrate concentration immediately translates into a lower growth rate. Concurrently, the curve representing the cell concentration also grows at a higher rate. The fluctuations in the growth rate observable can again be traced back in part to slight changes to the environmental conditions.

Additionally, this time limitations of the simulation come into play. In real fed-batch processes, a balance between cell growth and substrate supply will be reached. In a simulation, new values for every variable relevant to the process are calculated only at set time intervals. This means, that, for example, μ will not react directly to the changing conditions in the liquid phase that occur in these intervals.

To elaborate, the substrate concentration is calculated by solving the respective differential equation. MATLAB® does this by approximating the course of the substrate concentration between the last simulated point in time and the next one via its ode45 solver. The last calculated growth rate is referenced in this equation as a static variable even though if determined by the substrate concentration the growth rate would realistically change with the substrate concentration during this approximation. If the last calculated growth rate was low, more substrate will be calculated to accumulate due to the glucose feed than in a real process. In a real process, the growth rate would increase with higher available substrate concentrations and thus higher rates of substrate depletion occur due to the growing cells. Hence, in this case, the simulation leaves us with more substrate in the liquid phase at the next simulated point in time than what would be present in a real process. Similarly, if the last calculated growth rate was high, more of the substrate entering the system via the feed will be used up in the time interval, leaving us with less substrate in the liquid phase than what would be present in a real process. These amplified changes in substrate concentration, as mentioned earlier, are directly translated into amplified changes in μ. To keep the resulting error as low as possible, the time intervals that need to be bridged should be kept as small as possible while keeping the computing load at a reasonable level.

Another 30 min after the second increase of the feed rate, the feed rate is increased a third time to 42.5 % or 0.2125 l/h. This, again, causes an increase of μ, a decrease of p_{O2}, a higher glucose concentration in the liquid phase, and a stronger exponential growth of the cell concentration in both simulated processes. μ jumps to about 0.325 1/h initially, though the decline that follows is amplified this time. Over the next 30 min, the growth rate drops over 0.05 1/h to 0.27 1/h. This is mirrored in the development of the p_{O2} plot. It decreases to about 25 % in response to the increased feed rate, though the drop is not as sharp as those observed previously. This is because the growth rate does not stay at its new peak and decreases immediately after it is reached. This, in turn, causes the cells to take up less oxygen than if the growth rate remained at the new peak value. Again, the decline of the growth rate can be explained by the exponential cell growth producing an increasing number of cells that take up substrate. Eventually, the feed rate is not high enough to compensate for this and the substrate concentration in the liquid decreases again. This can be observed in both simulated processes. The growth rate shows similar fluctuations to those observed previously and can be traced to the same characteristics of the mathematical model. The cell concentration seems to increase at a higher exponential rate than in the previous feed step, which again can be traced back to the increased growth rate.

After another 30 min, the feed rate is increased once again to 60 % or 0.3 l/h. The previously observed patterns can be seen here as well. The p_{O2} experiences a sharp drop from about 28 % to about 15 % in response to a sharp increase of the growth rate μ, though it should be noted that the p_{O2} in the MATLAB®-based simulation is around 1-2 % higher than that in BIOSIM at times. This could be due to differences in the temperature. The temperature controllers are not completely the same between the two simulations and even temperature differences in the 0.1 °C magnitude have a noticeable impact on the dissolved oxygen concentration. If the temperature is slightly lower, more oxygen is dissolved in the liquid phase and the p_{O2} increases, and vice versa. Since the growth rate is no longer determined by the substrate but by the oxygen concentration, differences in p_{O2} now directly translate into growth rate differences. μ jumps to about 0.35 1/h in the MATLAB®-based simulation and 0.34 1/h in BIOSIM. Still, both curves show the same characteristic course. After the initial rise of μ, it immediately decreases again, this time in a linear fashion. The p_{O2}, similarly, shows a comparable pattern. As the cells multiply, they take up more and more oxygen. This leads to lower oxygen concentrations in the liquid, and since this is now the bottleneck, μ responds in kind. As the substrate concentration is no longer the limiting factor, it is supplied in abundance and starts to accumulate in the liquid phase. As μ is increased in the beginning, an increase in the exponential growth of the cells can be observed once more.

At the 4 h mark, 30 min later, the feed rate is increased a final time to 77.5 % or 0.3875 l/h. As the substrate concentration is no longer the limiting factor, μ does not respond as it did previously and the trends observable (steadily decreasing μ and p_{O2}) continue like they did beforehand. Only the accumulated substrate in the liquid phase grows at a steeper rate, as now more substrate is supplied but not used up.

At the 4.2 h mark, the manual feed is stopped, and the p_{O2}-feed controller is switched on. Additionally, the stirrer speed is reduced to 500 rpm. As there is still substrate available in the liquid phase and the p_{O2} is below the setpoint (20 %), no feed is added by the controller until the substrate is used up. Once the substrate is used up and the cells stop growing, the p_{O2} sharply increases again and the controllers of the MATLAB®-based simulation and the DCU coupled to BIOSIM start the feed.

The DCU controller shows an initial overshoot followed by an undershoot. The controlled variable is then held at the setpoint, though oscillations can be observed. There seems to be a permanent setpoint deviation of about 1-2 % as well. This is a characteristic step response of a PID controller to a changing input (the error between p_{O2} setpoint and actual p_{O2} in this case). The MATLAB®-based controller behaves slightly differently. Here, no overshoot occurs, but the control variable (the feed rate) is gradually increased to let the controlled variable (p_{O2}) approach its setpoint. Furthermore, no oscillations are observable, though it takes longer to reach the setpoint. This is explained by the different controller parameters (K_P, K_I, and K_D) and by the fact that BIOSIM is controlled by an external DCU. As such, all transmitted signals are filtered through a delay. This is mimicked by the PT1 introduced in the MATLAB®-based simulation, though differences will still occur. Hence, the response of BIOSIM and the MATLAB®-based simulation will always be different, even if the controller parameters are the same. However, for the purpose of this work, this is of no consequence, as both systems are capable of meeting the expectations placed upon them.

The p_{O2}-feed control operates by supplying substrate to the cells under substrate-limited conditions to control their growth and thereby their oxygen uptake. By controlling the oxygen uptake, the p_{O2} can consequently be regulated. Due to the nature of this controller, this also means that μ is controlled. This can be observed in both simulations, as the course of μ mimics the course of the feed rate. Once the feed rate and the p_{O2} are stabilized, μ is stabilized as well. Over time, μ will decrease, as more cells will be present in the liquid phase and take up more oxygen. Thus, to achieve the same level of dissolved oxygen, the cells need to grow at a slower rate as they multiply. This can also be observed in both simulations. Consequently, the exponential growth of the cells is slowed which is reflected in the course of c_{XL}.

As demonstrated by these two simulations run in parallel, both yield almost identical results with minor, inconsequential differences. They have similar responses to the same inputs and differences occur only based on the implemented process control. Since the MATLAB®-based simulation is easier to operate and includes additional functionalities, it represents a suitable replacement for BIOSIM and a capable educational software for future students.

4.4 Comparison with other software

Currently, access to educational biofermentation training software is limited. Publications tend to focus on research or industry and comparable simulations are sparse. The software developed by Pörtner and Hass in 2009 is the only educational simulation available to the public and will be compared to the MATLAB®-based simulation developed in this work in the following section [29].

Despite both simulations having similar purposes, they differ in structure. Whereas the Pörtner/Hass simulation aims to walk the user through different preset experimental setups involving different organisms, the simulation developed in this work is more flexible. Here, all process control components are continually accessible, and starting parameters and variables can be set by the user at will. The Pörtner/Hass simulation follows a more guided setup, as the user may choose one experimental setup from a predefined selection. Every option has a specific aim, such as teaching the user how to operate an exponential feed profile. In the following simulation, the user is then able to operate the feed rate, but other control options, such as p_{O2} control or changing the aeration rate, are not available. The accompanying publication is necessary to provide experimental background information. The fermentation simulated in chapter 4.3, for example, could thus not be replicated with the Pörtner/Hass simulation. Consequently, the MATLAB®-based simulation at hand is more customizable regarding the possible experimental courses, but less guided. Additionally, the Pörtner/Hass simulation allows the user to select between three different organisms: yeast, bacteria, and mammalian cells. This is currently not possible in the simulation developed in this work, though it could be added by identifying the necessary process parameters, such as $Y_{X/Ogr}$, and updating them. The pyFOOMB Python package mentioned earlier could be of interest here to identify these global process parameters [19].

Moreover, the simulation at hand includes other functionalities that the Pörtner/Hass simulation does not. In the simulation at hand, the current values of important process variables are given in the form of a plot and as numerical values, and developments can also be indicated in a trend table. What variables are displayed or indicated is decided by the user. In the Pörtner/Hass simulation, the numerical values of concentrations are indicated once the user draws a (virtual) sample and no developments are given. There are also no plots generated for these data. The variables regarding the process control are given in the form of numerical values and can be visualized as a plot. What variables are displayed is not customizable. The simulation at hand also introduces more extensive launch options, such as loading a previously started process from where it was left off and plotting generated data after the process was finished. Lastly, the speed at which the simulation generates new entries is adjustable in the simulation at hand as well, whereas the Pörtner/Hass simulation runs at a predetermined speed.

Concludingly, while both simulations aim to provide training to users in regard to operating a biofermentation process, their approaches vary. The Pörtner/Hass simulation offers a more guided solution with supplementation of the simulation with an accompanying publication. The simulation at hand is more customizable and does not require additional information, but does not currently offer the selection between different microorganisms.

5 Conclusion and outlook

In this work, a standalone application was developed in MATLAB® App Designer capable of simulating batch and fed-batch fermentation processes of *Escherichia coli* K12. This simulation incorporates all relevant process control options and can be used to obtain the same results as the BIOSIM system which it is based upon. Compared to the BIOSIM system, the new MATLAB®-based app also includes additional functionalities as well as improved convenience and accessibility of use. As the generated data can now be exported to a plot directly or even plotted anew once the simulation has concluded, this also renders the need for additional programs obsolete. It was therefore concluded that it is well suited to take the place of the BIOSIM system as the educational software used by students in the BPA special course.

To further enhance the educational potential of the app developed in this work, additional functionalities could be developed that allow for the process control to be taken over by external physical control units or even other virtual ones based on MATLAB® Simulink. The first would allow the students to familiarize themselves with standard equipment used in real fermentations, and the latter would allow for the testing and optimization of different types of controllers. This is of interest to assess and optimize the capabilities of different controllers to respond to the various problems occurring in fermentation processes, or to develop controllers for a specific bioprocess. In this context, introducing random events to the simulation might also be of interest. As of now, the mathematical model will always deliver the same results if the same parameters, initial values, and controller inputs are selected. No two real processes will be the same, as random events such as spontaneous malfunctions of instrumentation or fluctuations of environmental conditions will occur. This is a reality that controllers need to be equipped to handle. If the simulation is to be used to establish process control strategies, this needs to be considered. An example of how this could be implemented was given by Pantano and colleagues [43].

Still, the MATLAB®-based app itself has room for further improvement. The mathematical model, as of now, has to be exchanged manually by altering the simulation code itself. Writing in an additional functionality that exchanges the model from an external file and adding the required parameters could be an interesting addition to be incorporated in the future. This would allow this simulation to be used for the simulation of the fermentation of virtually any microorganism or even completely different types of bioprocesses. Furthermore, the parameters of the microorganism in this model can be adjusted based on data from real processes to simulate these processes as accurately as possible. Once the real process and the simulation show sufficient congruence, the simulation can be used to optimize process control strategies, design, and scale-up or even conduct economic analyses. This would allow for significantly reduced material and time requirements, which ultimately results in much lower overall costs. The pyFOOMB package developed by Hemmerich and colleagues might be of use for the identification of the needed parameters [19]. As mentioned in chapter 1, similar approaches were chosen for the design and assessment of optimal process control strategies for different pharmaceutical substances. Hence, a mathematical simulation is an invaluable tool for process optimization and, in the context of this work, it could be used to improve the antimicrobial peptide yield of *P. pastoris* and *E. coli* fermentations at the heart of the PharmCycle project at the HAW.

The code of the simulation established in this work is free to be used and altered by the students. It hence can also potentially be used to teach practical applications of MATLAB® and the theoretical design of apps utilized in biotechnological contexts. It may, for this reason, also find application in other parts of the curriculum, such as the Analysis, Modeling, and Simulation of Bioprocesses lecture given in the summer semester of the Master's program Pharmaceutical Biotechnology.

As demonstrated, the MATLAB®-based simulation established in this work is suitable to replace the BIOSIM system. Not only that, but it can also potentially find use in other areas than the one it was conceived for, such as optimization of control strategies and controllers as well as other educational purposes. This is illustrated in figure 39. In conclusion, the aim set in the introduction was met, and a fed-batch fermentation simulation for *E. coli* K12 was developed.

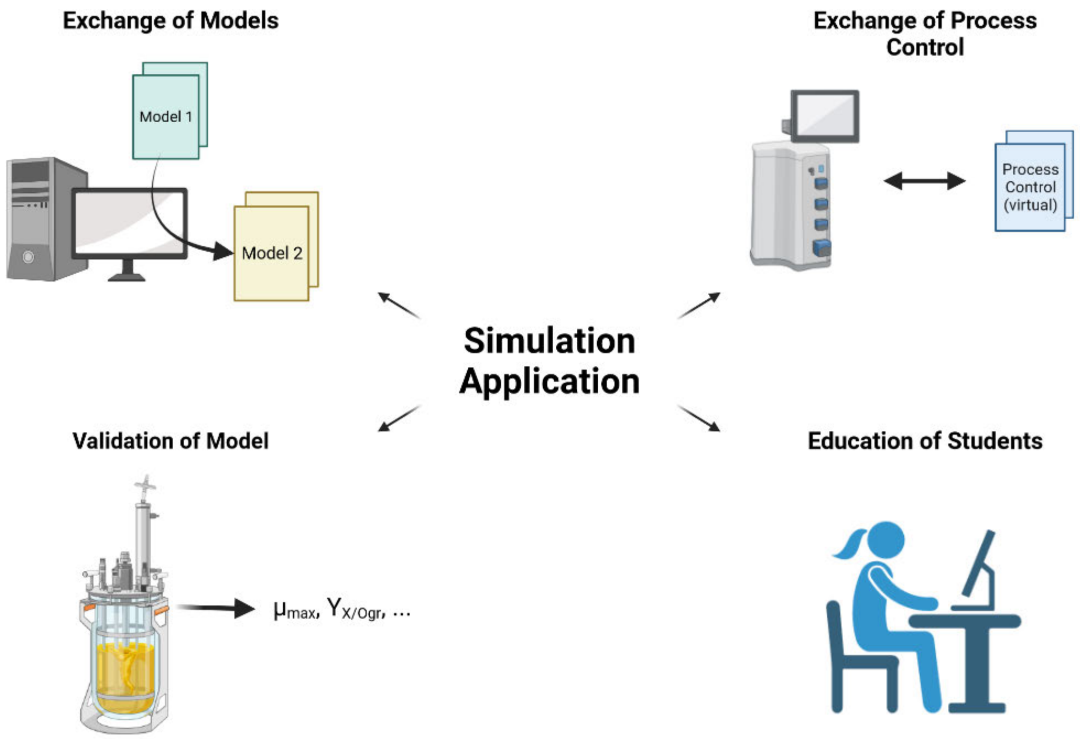


Figure 39: **Outlook for the MATLAB®-based simulation application developed in this work.** Created with BioRender.

List of Variables

$\mathbf{A}_{JK}$	Heat transfer area between thermostat subsystems J and K
$\mathbf{c}_{IK}$	Concentration of component I in subsystem K
$\mathbf{c}_{K}$	Specific heat capacity in thermostat subsystem K
$\mathbf{C}_{IK}$	Molar concentration of component I in subsystem K
$\mathbf{C}_{K}$	Volumetric heat capacity in thermostat subsystem K
$\mathbf{D}_{K}$	Reciprocal mean residence time in thermostat subsystem K
$\mathbf{e}$	Filtered error between controller setpoint and actual value of controlled variable
$\mathbf{e}_{n}$	Normalized error e
$\mathbf{F}_{K}$	Flow rate from subsystem K
$\mathbf{F}_{nG}$	Aeration rate at normalized conditions
$\mathbf{F}_{nI}$	Aeration rate of gas component I at normalized conditions
$\mathbf{k}_{IJI}$	Inhibition constant of substrate J by substrate I
$\mathbf{k}_{J}$	Monod limitation constant of component J
$\mathbf{K}_{a}$	Gain of controller of type a
$\mathbf{K}_{HSt}$	Proportional gain of stirrer heat generation
$\mathbf{K}_{HM}$	Proportional gain of microbial heat generation
$\mathbf{m}_{J}$	Exchange mass in thermostat subsystem J
$\dot{m}$	Mass flow
$\mathbf{M}_{I}$	Mole mass of component I
$\mathbf{N}_{St}$	Stirrer speed
OUR	Oxygen uptake rate
pH	pH value
$\mathbf{q}_{I/X}$	Cell-specific reaction rate of component I
$\mathbf{Q}_{J}$	Supply rate of component J
$\dot{Q}_{St}$	Thermal power of the stirrer

$\dot{Q}_M$	Thermal power of the microorganisms
$\mathbf{R}_C$	Cooling water entry heat resistance
$\mathbf{R}_{JK}$	Overall heat transmission resistance between thermostat subsystem J and K
$\mathbf{R}_T$	Thermostat circulation heat resistance
$\mathbf{t}$	Time
$\mathbf{T}$	Time constant of PT1 delay
$\mathbf{T}_K$	Absolute temperature of subsystem K
$\mathbf{V}_K$	Volume of subsystem K
$\mathbf{x}_{IAIR}$	Mole fraction of component I in air
$\mathbf{x}_{IG}$	Mole fraction of component I in the gas phase
$\mathbf{y}_b$	Control variable output for control element b
$\mathbf{Y}_{X/I}$	Cell mass yield coefficient of component I
$\boldsymbol{\alpha}_J$	Heat transfer coefficient in fluid J
$\boldsymbol{\delta}_{JK}$	Wall thickness between thermostat subsystem J and K
$\boldsymbol{\epsilon}$	Error between controller setpoint and actual value of controlled variable
$\boldsymbol{\vartheta}_K$	Temperature in subsystem K
$\boldsymbol{\kappa}_I$	Environmental growth control function of parameter I
$\boldsymbol{\lambda}_{JIK}$	Thermal conductivity of the wall between thermostat subsystems J and K
$\boldsymbol{\mu}$	Specific growth rate
$\boldsymbol{\rho}_K$	Density of the medium in vessel K
$\boldsymbol{\tau}_{JK}$	Thermal transport time constant between thermostat subsystems J and K

List of Indices

AIR	Air
Al	Alkali/ammonia
C	Cooling water system
C, CO2	Carbon dioxide
D	Double jacket, derivative
E	Reactor environment
gr	Growth fraction
G	Gas phase
H	Heating
I	Inhibition, integral
I/J	Component I per component J
L	Liquid phase
m	Maintenance fraction
max	Maximum value
min	Minimum value
M	Microbial
n	Normalized gas conditions
opt	Optimal value
O, O2	Oxygen
P	proportional
Pi	Product i
R	Feed tank
Si	Substrate i
St	Stirrer
T	Thermostat

T1	Titration tank 1 (acid)
T2	Titration tank 2 (base)
Tc	Thermostat cooling system
Th	Thermostat heating system
w	Setpoint
W	Reactor wall, water
X	Dry biomass

List of Figures

List of Tables

References

[1] Jing Du, Zengyi Shao, and Huimin Zhao. Engineering microbial factories for synthesis of value-added products. *Journal of industrial microbiology & biotechnology*, 38(8): 873 890, 2011. doi: 10.1007/s10295-011-0970-3.

[2] Isuru A. Udugama, Pau C. Lopez, Carina L. Gargalo, Xueliang Li, Christoph Bayer, and Krist V. Gernaey. Digital Twin in biomanufacturing: challenges and opportunities towards its implementation. *Systems Microbiology and Biomanufacturing*, 1(3): 257 274, 2021. ISSN 2662-7655. doi: 10.1007/s43393-021-00024-0.

[3] Yuan-Hang Du, Min-Yu Wang, Lin-Hui Yang, Ling-Ling Tong, Dong-Sheng Guo, and Xiao-Jun Ji. Optimization and Scale-Up of Fermentation Processes Driven by Models. *Bioengineering (Basel, Switzerland)*, 9(9):473, 2022. ISSN 2306-5354. doi: 10.3390/bioengineering9090473.

[4] Ashok Mulchandani, John H.T. Luong, and Scott V.C. Groom. Substrate inhibition kinetics for microbial growth and synthesis of poly-β-hydroxybutyric acid by *Alcaligenes eutrophus* atcc 17697. *Applied Microbiology and Biotechnology*, 30(1), 1989. ISSN 0175-7598. doi: 10.1007/BF00255990.

[5] Poonam Raje and Ashok K. Srivastava. Updated mathematical model and fed-batch strategies for poly-β-Hydroxybutyrate (PHB) production by *Alcaligenes eutrophus*. *Bioresource Technology*, 64(3):185 192, 1998. ISSN 0960-8524. doi: 10.1016/S0960-8524(97)00173-9.

[6] A. Varma and B. O. Palsson. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type *Escherichia coli* W3110. *Applied and environmental microbiology*, 60(10):3724 3731, 1994. ISSN 0099-2240. doi: 10.1128/aem.60.10.3724-3731.1994.

[7] Philipp Noll and Marius Henkel. History and Evolution of Modeling in Biotechnology: Modeling & Simulation, Application and Hardware Performance. *Computational and Structural Biotechnology Journal*, 18:3309 3323, 2020. ISSN 2001-0370. doi: 10.1016/j.csbj.2020.10.018. URL `https://www.sciencedirect.com/science/article/pii/S2001037020304402`.

[8] Arlindo L. Oliveira. Biotechnology, Big Data and Artificial Intelligence. *Biotechnology journal*, 14(8):e1800613, 2019. doi: 10.1002/biot.201800613.

[9] Mohamed Helmy, Derek Smith, and Kumar Selvarajoo. Systems biology approaches integrated with artificial intelligence for optimized metabolic engineering. *Metabolic Engineering Communications*, 11:e00149, 2020. ISSN 2214-0301. doi: 10.1016/j.

mec.2020.e00149. URL https://www.sciencedirect.com/science/article/pii/S2214030120300493.

[10] Dörte Solle, Bernd Hitzmann, Christoph Herwig, Manuel Pereira Remelhe, Sophia Ulonska, Lynn Wuerth, Adrian Prata, and Thomas Steckenreiter. Between the Poles of Data-Driven and Mechanistic Modeling for Process Operation. *Chemie Ingenieur Technik*, 89(5):542 561, 2017. ISSN 0009286X. doi: 10.1002/cite.201600175.

[11] Zoltan Kalman Nagy. Model based control of a yeast fermentation bioreactor using optimally designed artificial neural networks. *Chemical Engineering Journal*, 127(1-3): 95 109, 2007. ISSN 13858947. doi: 10.1016/j.cej.2006.10.015.

[12] Dietmar W. Hutmacher and Harmeet Singh. Computational fluid dynamics for improved bioreactor design and 3D culture. *Trends in biotechnology*, 26(4):166 172, 2008. doi: 10.1016/j.tibtech.2007.11.012.

[13] Stephen Goldrick, Andrei Ştefan, David Lovett, Gary Montague, and Barry Lennox. The development of an industrial-scale fed-batch fermentation simulation. *Journal of biotechnology*, 193:70 82, 2015. doi: 10.1016/j.jbiotec.2014.10.029.

[14] Neba Fabrice Abunde, Nana Yaw Asiedu, and Ahmad Addo. Modeling, simulation and optimal control strategy for batch fermentation processes. *International Journal of Industrial Chemistry*, 10(1):67 76, 2019. ISSN 2228-5970. doi: 10.1007/s40090-019-0172-9.

[15] Tina Tavasoli, Sareh Arjmand, Seyed Omid Ranaei Siadat, Seyed Abbas Shojaosadati, and Abbas Sahebghadam Lotfi. A robust feeding control strategy adjusted and optimized by a neural network for enhancing of alpha 1-antitrypsin production in *Pichia pastoris*. *Biochemical Engineering Journal*, 144:18 27, 2019. ISSN 1369703X. doi: 10.1016/j.bej.2019.01.005.

[16] Lihe Zhang, Bin Chao, and Xu Zhang. Modeling and optimization of microbial lipid fermentation from cellulosic ethanol wastewater by *Rhodotorula glutinis* based on the support vector machine. *Bioresource Technology*, 301:122781, 2020. ISSN 0960-8524. doi: 10.1016/j.biortech.2020.122781. URL https://www.sciencedirect.com/science/article/pii/S096085242030050X.

[17] Christopher Culley, Supreeta Vijayakumar, Guido Zampieri, and Claudio Angione. A mechanism-aware and multiomic machine-learning pipeline characterizes yeast cell growth. *Proceedings of the National Academy of Sciences of the United States of America*, 117(31):18869 18879, 2020. doi: 10.1073/pnas.2002959117.

[18] Yuan-Hang Du, Ling-Ling Tong, Yue Wang, Meng-Zhen Liu, Li Yuan, Xin-Ya Mu, Shao-Jie He, Shi-Xiang Wei, Yi-Dan Zhang, Zi-Lei Chen, Zhi-Dong Zhang, and

Dong-Sheng Guo. Development of a kinetics integrated CFD model for the industrial scale up of DHA fermentation using *Schizochytrium* sp. *AIChE Journal*, 68(9), 2022. ISSN 00011541. doi: 10.1002/aic.17750.

[19] Johannes Hemmerich, Niklas Tenhaef, Wolfgang Wiechert, and Stephan Noack. pyFOOMB: Python framework for object oriented modeling of bioprocesses. *Engineering in life sciences*, 21(3-4):242 257, 2021. ISSN 1618-0240. doi: 10.1002/elsc.202000088.

[20] Rui M. C. Portela, Christos Varsakelis, Anne Richelle, Nikolaos Giannelos, Julia Pence, Sandrine Dessoy, and Moritz von Stosch. When Is an *In Silico* Representation a Digital Twin? A Biopharmaceutical Industry Approach to the Digital Twin Concept. *Advances in biochemical engineering/biotechnology*, 176:35 55, 2021. ISSN 0724-6145. doi: 10.1007/10_2020_138.

[21] Dimitrios Piromalis and Antreas Kantaros. Digital Twins in the Automotive Industry: The Road toward Physical-Digital Convergence. *Applied System Innovation*, 5(4):65, 2022. doi: 10.3390/asi5040065.

[22] Gareth John Macdonald. Biopharma Is Going Digital ... Bit by Bit. *Genetic Engineering & Biotechnology News*, 42(6):46 49, 2022. ISSN 1935-472X. doi: 10.1089/gen.42.06.15.

[23] Igiri Onaji, Divya Tiwari, Payam Soulatiantork, Boyang Song, and Ashutosh Tiwari. Digital twin in manufacturing: conceptual framework and case studies. *International Journal of Computer Integrated Manufacturing*, 35(8):831 858, 2022. ISSN 0951-192X. doi: 10.1080/0951192X.2022.2027014.

[24] Inga Gerlach, Carl-Fredrik Mandenius, and Volker C. Hass. Operator training simulation for integrating cultivation and homogenisation in protein production. *Biotechnology Reports*, 6:91 99, 2015. ISSN 2215-017X. doi: 10.1016/j.btre.2015.03.002. URL `https://www.sciencedirect.com/science/article/pii/S2215017X15000168`.

[25] Inga Gerlach, Simone Brüning, Robert Gustavsson, Carl-Fredrik Mandenius, and Volker C. Hass. Operator training in recombinant protein production using a structured simulator model. *Journal of biotechnology*, 177:53 59, 2014. doi: 10.1016/j.jbiotec.2014.02.022. URL `https://www.sciencedirect.com/science/article/pii/S0168165614001035`.

[26] K. Alvarado, J. Bayona, J. Consuegra, D. Parada, N. Sepúlveda, and G. Gelves. Use of Operational Training Simulation in the Study of Ethanol Operating Conditions: A Powerful Tool for Education and Research Performance Improvement. *Journal of Physics: Conference Series*, 1655(1):012093, 2020. ISSN 1742-6588. doi: 10.1088/1742-6596/1655/1/012093.

[27] Laura Marcano, Finn Aakre Haugen, Ronny Sannerud, and Tiina Komulainen. Review of simulator training practices for industrial operators: How can individual simulator training be enabled? *Safety Science*, 115:414 424, 2019. ISSN 09257535. doi: 10.1016/j.ssci.2019.02.019.

[28] Inga Gerlach, Volker C. Hass, Simone Brüning, and Carl-Fredrik Mandenius. Virtual bioreactor cultivation for operator training and simulation: application to ethanol and protein production. *Journal of Chemical Technology and Biotechnology*, 88(12): 2159 2168, 2013. ISSN 01420356. doi: 10.1002/jctb.4079.

[29] Volker C. Hass and Ralf Pörtner. *Praxis der Bioprozesstechnik: Mit virtuellem Praktikum*. Spektrum, Akad. Verl., Heidelberg, 2. aufl. edition, 2011. ISBN 3827428289.

[30] Reiner Luttmann and Klaus-Uwe Gollmer. On-Line Simulation Techniques for Bioreactor Control Development. In Karl Schügerl and Karl-Heinz Bellgardt, editors, *Bioreaction Engineering*, pages 167 203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-642-64103-9. doi: 10.1007/978-3-642-59735-0_7.

[31] MATLAB - mathworks, 09.10.2022. URL `https://www.mathworks.com/products/matlab.html`.

[32] Why MATLAB - MathWorks, 10.10.2022. URL `https://www.mathworks.com/products/matlab/why-matlab.html`.

[33] Biotech & Pharmaceutical - MATLAB & Simulink Solutions, 09.10.2022. URL `https://www.mathworks.com/solutions/biotech-pharmaceutical.html`.

[34] Matrices and Arrays - MATLAB & Simulink, 10.10.2022. URL `https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html`.

[35] Develop Apps Using App Designer - MATLAB & Simulink, 10.10.2022. URL `https://www.mathworks.com/help/matlab/app-designer.html`.

[36] MATLAB App Designer, 10.10.2022. URL `https://www.mathworks.com/products/matlab/app-designer.html`.

[37] Open App Designer Start Page or existing app file - MATLAB appdesigner, 10.10.2022. URL `https://www.mathworks.com/help/matlab/ref/appdesigner.html`.

[38] How to create a matlab function - video, 23.12.2022. URL `https://www.mathworks.com/videos/functions-and-subfunctions-97413.html`.

[39] Ordinary Differential Equations - MATLAB & Simulink, 10.10.2022. URL `https://www.mathworks.com/help/matlab/ordinary-differential-equations.html`.

[40] Choose an ODE Solver - MATLAB & Simulink, 11.10.2022. URL https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html.

[41] Heinz Unbehauen. *Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*, volume 1, Ed. 14 of *Regelungstechnik*. Vieweg, Braunschweig, 13., verb. aufl. edition, 2007. ISBN 978-3-8348-0230-9. doi: 10.1007/978-3-8348-9196-9.

[42] Byung-Su Ko and Thomas F. Edgar. Performance assessment of cascade control loops. *AIChE Journal*, 46(2):281 291, 2000. ISSN 00011541. doi: 10.1002/aic.690460208.

[43] María N. Pantano, Mario E. Serrano, María C. Fernández, Francisco G. Rossomando, Oscar A. Ortiz, and Gustavo J. E. Scaglia. Multivariable Control for Tracking Optimal Profiles in a Nonlinear Fed-Batch Bioprocess Integrated with State Estimation. *Industrial & Engineering Chemistry Research*, 56(20):6043 6056, 2017. ISSN 0888-5885. doi: 10.1021/acs.iecr.7b00831.

Appendix

The control app

```
classdef ControlAppODE_LuttmannpHTEMPnewexported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                     matlab.ui.Figure
        ExpertmodeButton             matlab.ui.control.Button
        DataStorageRunningLamp       matlab.ui.control.Lamp
        DataStorageRunningLampLabel  matlab.ui.control.Label
        ProcessRunningLamp           matlab.ui.control.Lamp
        ProcessRunningLampLabel      matlab.ui.control.Label
        ProcessTimehEditField        matlab.ui.control.NumericEditField
        ProcessTimehEditFieldLabel   matlab.ui.control.Label
        ExitButton                   matlab.ui.control.Button
        TabGroup                     matlab.ui.container.TabGroup
        ControlOptionsTab            matlab.ui.container.Tab
        AntifoamPanel                matlab.ui.container.Panel
        AntifoamLamp                 matlab.ui.control.Lamp
        AntifoamadditionSwitch       matlab.ui.control.RockerSwitch
        TemperaturePanel             matlab.ui.container.Panel
        tempParametersButton         matlab.ui.control.Button
        temptrendLabel               matlab.ui.control.Label
        TemperatureLamp              matlab.ui.control.Lamp
        Mode_tempDropDown            matlab.ui.control.DropDown
        Mode_tempDropDownLabel       matlab.ui.control.Label
        thetaLCEditField             matlab.ui.control.NumericEditField
        thetaLCEditFieldLabel        matlab.ui.control.Label
        thetaLwCEditField            matlab.ui.control.NumericEditField
        thetaLwCEditFieldLabel       matlab.ui.control.Label
        HeatingSwitch                matlab.ui.control.RockerSwitch
        HeatingSwitchLabel           matlab.ui.control.Label
        CoolingSwitch                matlab.ui.control.RockerSwitch
        CoolingSwitchLabel           matlab.ui.control.Label
        pO2Panel                     matlab.ui.container.Panel
        pO2ParametersButton          matlab.ui.control.Button
        pO2trendLabel                matlab.ui.control.Label
        xOGinwEditField              matlab.ui.control.NumericEditField
        xOGinwEditFieldLabel         matlab.ui.control.Label
        FnGwlminEditField            matlab.ui.control.NumericEditField
        FnGwlminEditFieldLabel       matlab.ui.control.Label
        Mode_pO2DropDown             matlab.ui.control.DropDown
        Mode_pO2DropDownLabel        matlab.ui.control.Label
        pO2Lamp                      matlab.ui.control.Lamp
        FnO2wlminEditField           matlab.ui.control.NumericEditField
        FnO2wlminEditFieldLabel      matlab.ui.control.Label
        FnAIRwlminEditField          matlab.ui.control.NumericEditField
        FnAIRwlminEditFieldLabel     matlab.ui.control.Label
        FRwlhEditField               matlab.ui.control.NumericEditField
        FRwlhEditFieldLabel          matlab.ui.control.Label
        NStwrpmEditField             matlab.ui.control.NumericEditField
        NStwrpmEditFieldLabel        matlab.ui.control.Label
        pO2EditField                 matlab.ui.control.NumericEditField
        pO2EditFieldLabel            matlab.ui.control.Label
        pO2wEditField                matlab.ui.control.NumericEditField
        pO2wEditFieldLabel           matlab.ui.control.Label
        pHPanel                      matlab.ui.container.Panel
        pHParametersButton           matlab.ui.control.Button
        pHtrendLabel                 matlab.ui.control.Label
        Mode_pHDropDown              matlab.ui.control.DropDown
        Mode_pHDropDownLabel         matlab.ui.control.Label
        pHLamp                       matlab.ui.control.Lamp
        pHLEditField                 matlab.ui.control.NumericEditField
        pHLEditFieldLabel            matlab.ui.control.Label
        pHwEditFieldLabel            matlab.ui.control.Label
        pHwEditField                 matlab.ui.control.NumericEditField
        AcidSwitch                   matlab.ui.control.RockerSwitch
        AcidSwitchLabel              matlab.ui.control.Label
        AlkaliSwitch                 matlab.ui.control.RockerSwitch
        AlkaliSwitchLabel            matlab.ui.control.Label
        LiquidWeightPanel            matlab.ui.container.Panel
        LWParametersButton           matlab.ui.control.Button
        mLtrendLabel                 matlab.ui.control.Label
        FHwEditField                 matlab.ui.control.NumericEditField
        FHwEditFieldLabel            matlab.ui.control.Label
        Mode_harvestDropDown         matlab.ui.control.DropDown
        Mode_harvestDropDownLabel    matlab.ui.control.Label
```

```
        LiquidWeightLamp              matlab.ui.control.Lamp
        mLkgEditField                 matlab.ui.control.NumericEditField
        mLkgEditFieldLabel            matlab.ui.control.Label
        mLwkgEditField                matlab.ui.control.NumericEditField
        mLwkgEditFieldLabel           matlab.ui.control.Label
        HarvestSwitch                 matlab.ui.control.RockerSwitch
        HarvestSwitchLabel            matlab.ui.control.Label
        VariablePoolTab               matlab.ui.container.Tab
        FHEditField                   matlab.ui.control.NumericEditField
        FHEditFieldLabel              matlab.ui.control.Label
        xOGinEditField                matlab.ui.control.NumericEditField
        xOGinEditFieldLabel           matlab.ui.control.Label
        FnGlminEditField              matlab.ui.control.NumericEditField
        FnGlminEditFieldLabel         matlab.ui.control.Label
        NStrpmEditField               matlab.ui.control.NumericEditField
        NStrpmEditFieldLabel          matlab.ui.control.Label
        FRlhEditField                 matlab.ui.control.NumericEditField
        FRlhEditFieldLabel            matlab.ui.control.Label
        FT2baselhEditFieldLabel       matlab.ui.control.Label
        FT2baselhEditField            matlab.ui.control.NumericEditField
        FT1acidlhEditFieldLabel       matlab.ui.control.Label
        FT1acidlhEditField            matlab.ui.control.NumericEditField
        LiquidVolumePanel             matlab.ui.container.Panel
        addedfeedlEditField           matlab.ui.control.NumericEditField
        addedfeedlEditFieldLabel      matlab.ui.control.Label
        addedAFlEditField             matlab.ui.control.NumericEditField
        addedAFlEditFieldLabel        matlab.ui.control.Label
        addedacidlEditField           matlab.ui.control.NumericEditField
        addedacidlEditFieldLabel      matlab.ui.control.Label
        addedbaselEditField           matlab.ui.control.NumericEditField
        addedbaselEditFieldLabel      matlab.ui.control.Label
        VLlEditField                  matlab.ui.control.NumericEditField
        VLlEditFieldLabel             matlab.ui.control.Label
        Tree                          matlab.ui.container.CheckBoxTree
        DisplayableVariablesNode      matlab.ui.container.TreeNode
        VLlNode                       matlab.ui.container.TreeNode
        cXLglNode                     matlab.ui.container.TreeNode
        cS1LglNode                    matlab.ui.container.TreeNode
        cS2LglNode                    matlab.ui.container.TreeNode
        cS3LglNode                    matlab.ui.container.TreeNode
        thetaLCNode                   matlab.ui.container.TreeNode
        pO2Node                       matlab.ui.container.TreeNode
        pHLNode                       matlab.ui.container.TreeNode
        xOGNode                       matlab.ui.container.TreeNode
        xCGNode                       matlab.ui.container.TreeNode
        RQmolmolNode                  matlab.ui.container.TreeNode
        QO2maxglhNode                 matlab.ui.container.TreeNode
        QCO2maxglhNode                matlab.ui.container.TreeNode
        OTRglhNode                    matlab.ui.container.TreeNode
        CTRglhNode                    matlab.ui.container.TreeNode
        NStrpmNode                    matlab.ui.container.TreeNode
        FnGlminNode                   matlab.ui.container.TreeNode
        xOGinNode                     matlab.ui.container.TreeNode
        FRlhNode                      matlab.ui.container.TreeNode
        UITable                       matlab.ui.control.Table
        AppOperationTab               matlab.ui.container.Tab
        SimulationStartOptionsLabel   matlab.ui.control.Label
        DataStorageOptionsLabel       matlab.ui.control.Label
        PreviouslyRecordedDataLabel   matlab.ui.control.Label
        StartingVariableOptionsLabel  matlab.ui.control.Label
        DataStorageatStartCheckBox    matlab.ui.control.CheckBox
        PlotPreviousDataButton        matlab.ui.control.Button
        LoadProcessButton             matlab.ui.control.Button
        InoculationatStartCheckBox    matlab.ui.control.CheckBox
        SelectedFileEditField         matlab.ui.control.EditField
        SelectedFileEditFieldLabel    matlab.ui.control.Label
        deltathEditField              matlab.ui.control.NumericEditField
        deltathEditFieldLabel         matlab.ui.control.Label
        StartSimulationButton         matlab.ui.control.Button
        DataStorageStartButton        matlab.ui.control.Button
        SaveStartingVariablesButton   matlab.ui.control.Button
        EditStartingVariablesButton   matlab.ui.control.Button
        ReadInStartingVariablesButton matlab.ui.control.Button
        AuthorInformationTab          matlab.ui.container.Tab
        Label                         matlab.ui.control.Label
    end

    % This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
    properties (Access = private)
        DialogApp  % Starting Variables Dialog box app
```

```
        DialogApp2 % pH Controller Parameter Dialog box app
        DialogApp3 % Temperture Controller Parameter Dialog box app
        DialogApp4 % pO2 Controller Parameter Dialog box app
        DialogApp5 % Liquid Weight Controller Parameter Dialog box app
        SimulationApp  % Simulation Window
    end

    properties (Access = public)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Constants concerning aeration
        xOAIR    = 0.2094; % Molar concentration of oxygen in AIR [-]
        xCAIR    = 0.0003; % Molar concentration of carbondioxide in AIR [-]
        HnO2     = 1.448*10^6; % O2 Henry constant under normal conditions [N*m/kg]
        HO20     = 1.44*10^6; % O2 Henry constant [N*m/kg]
        HnCO2    = 3.001*10^4; % CO2 Henry constant under normal conditions [N*m/kg]
        HCO20    = 3.001*10^4; % CO2 Henry constant [N*m/kg]
        VnM      = 22.412; % Gas volume under normal conditions[l/mol]
        MO2      = 32; % Molar mass of O2 [g/mol]
        MCO2     = 44; % Molar mass of CO2 [g/mol]
        TnG      = 273.15; % Gas temperature under normal conditions [K]
        pnG      = 1.0133*10^5; % Gas pressure under normal conditions [N/m^2]
        R        = 8314; % Gas constant [N*m/(kmol*K)]
        K1HO2    = -2.723*10^(-2); % Normed parameter of O2 Bunsen coefficient [°C^(-1)]
        K2HO2    = 5.627*10^(-4); % Normed parameter of O2 Bunsen coefficient [°C^(-2)]
        K3HO2    = -6.597*10^(-6); % Normed parameter of O2 Bunsen coefficient [°C^(-3)]
        K4HO2    = 3.283*10^(-8); % Normed parameter of O2 Bunsen coefficient [°C^(-4)]
        K1HCO2   = -3.889*10^(-2); % Normed parameter of CO2 Bunsen coefficient [°C^(-1)]
        K2HCO2   = 9.442*10^(-4); % Normed parameter of CO2 Bunsen coefficient [°C^(-2)]
        K3HCO2   = -1.328*10^(-5); % Normed parameter of CO2 Bunsen coefficient [°C^(-3)]
        K4HCO2   = 8.105*10^(-8); % Normed parameter of CO2 Bunsen coefficient [°C^(-4)]

        CH0      = 10^(-7); % pH norm concentration [mol/l]
        pH0      = 7.5; % Initial pH at t = 0 h [-]
        cH2O     = 1.167; % Specific heat capacity of water [h/(kgK)]

        thetaL0  = 32.0; % Initial temperature in liquid phase at t = 0 h [°C]
        thetaD0  = 32.0; % Initial temperature in double jacket at t = 0 h [°C]

        alphaH   = 5000; % Heat transfer coefficient of the heating medium [W/(m^2*K)]
        alphaTh  = 1000; % Heat transfer coefficient of the temperature medium [W/(m^2*K)]
        deltaHTh = 1.0*10^(-3); % Wall diameter between H and Th [m]
        lamdaHTh = 380.0; % Thermal conductivity of the wall between H and Th [W/(m*K)]

        deltahv  = 2170.0; % Evaporation enthalpie of water [kJ/kg]
        mdotCmax = 1500.0; % Maximum cooling-water mass flux 8 [kg/h]
        thetaCin = 15.0; % Temperature cooling-water inlet [°C]
        mC       = 1.0; % Mass of the cooling system [kg]
        alphaC   = 3700; % Heat transfer coefficient of cooling-water [W/(m^2K)]
        deltaCT  = 3*10^(-3); % Exchanging diameter of the heat exchanger [m]
        lamdaCT  = 46.0; % Thermal conductivity of the exchange wall [W/(mK)]
        ACT      = 0.5; % Surface of the heat exchanger [m^2]

        VH       = 0.3*10^(-3); % Volume of heating system [m^3]
        rhoH     = 1.6831; % Density of heating system [kg/m^3]
        rhoH2O   = 998.2; % Density of water at 20°C [kg/m^3]
        mTc      = 1.0; % Mass of cooling system [kg]
        alphaT   = 3700; % Heat transfer coefficient of the temperated water [W/(m^2K)]

        PHmax    = 10000; % Maximum electrical heating power (10 l reactor) [W]
        mdotHmax = 1500; % Maximum steam heating flux [kg/h]
        thetaHin = 134.0; % Temperature of steam at inlet (saturated steam) [°C]
        AHTh     = 0.13; % Surface double jacket/reactor interior [m^2]

        mD       = 23.5; % Liquid mass in double jacket [kg]
        alphaD   = 2000; % Heat transfer coefficient double jacket-wall [W/(m^2K)]
        deltaDU  = 4.0*10^(-3); % Diameter double jacket outer wall [m]
        lamdaDU  = 46.0; % Coefficient of thermal conductivity double jacket outer wall [W/(mK)]
        ADU      = 1.4; % Surface double jacket - environment [m^2]
        deltaDL  = 4.0*10^(-3); % Diameter doubble jacket inner wall [m]
        lamdaDL  = 46.0; % Coefficient of thermal conductivity double jacket inner wall [W/(mK)]
        ADL      = 1.13; % Surface double jacket - liquid phase [m^2]

        rhoL     = 1.0; % Density of liquid in reactor [kg/l]
        alphaL   = 4000; % Heat transfer coefficient liquid - wall [W/(m^2K)]
        deltaLU  = 4.0*10^(-3); % Wall-diameter liquid - environment [m]
        lamdaLU  = 46.0; % Thermal conductivity liquid - environment [W/(m^2K)]
        ALU      = 0.077; % Surface liquid - environment [m^2]
        KHSt     = 2.72*10^(-8) % Proportionality coefficient stirrer heat evolution [Wmin^3/l]
        KHM      = 4.036; % Proportionality coefficient microorganism heat evolution [Wh/g]

        mWD      = 56.0; % Mass double jacket outside +0.5 double jacket inside [kg]
        mWL      = 24.0; % Mass reactor wall -0.5 double jacket inside [kg]
```

```
        cW       = 0.139; % Specific heat capacity of the wall [Wh/(kgK)]

        thetaU   = 25.0; % Environmental temperature in reactor [°C]
        alphaU   = 10; % Heat transfer coefficient environment - wall [W/(m^2 K)]
        eta0     = 1.002*10^(-3); % Viscosity of water [Ns/m^2]
        eta1     = 10^(-3); % Viscosity at cXLeta [Ns/m^2]
        cXLeta   = 100; % Refering cell concentration [g/l]

        mdotT    = 1500; % Mass flux in temperature cycle [kg/h]
        KC1      = 4.31*10^(-7); % 1. CO2 dissociation constant [mol/l]
        KC2      = 5.61*10^(-11); % 2. CO2 dissociation constant [mol/l]
        KB1      = 7.52*10^(-3); % 1. buffer dissociation constant (H3PO4 to H2PO4-) [mol/l]
        KB2      = 6.23*10^(-8); % 2. buffer dissociation constant (H2PO4- to HPO4--) [mol/l]
        KB3      = 2.2.*10^(-13); % 3. buffer dissociation constant (HPO4-- to PO4---) [mol/l]
        KP       = 1.76*10^(-5); % Acetate (product) dissociation constant (CH3COOH to CH3COO-) [mol/l]
        KAl      = 1.76*10^(-5); % Ammonia (pH base) dissociation constant (NH3 to NH4+) [mol/l]
        KAc1     = 1.54*10^(-2); % 1. titration acid dissociation constant (H2SO3 to HSO3-) [mol/l]
        KAc2     = 1.02*10^(-7); % 2. tritration acid dissociation constant (HSO3- to SO3--) [mol/l]
        MAc      = 82.0; % Molar mass of titration acid [g/mol]
        MAl      = 17.0; % Molar mass of ammonia [g/mol]
        MP       = 60; % Molar mass of product [g/mol]
        VT10     = 1.0; % Acid reservoir volume at t = 0 h [l]
        VT20     = 1.0; % Base reservoir volume at t = 0 h [l]
        kLamin   = 2.5; % Minimum value of kLa [1/h]
        kLamax   = 1469; % Maximum value of kLa [1/min]
        alpha    = 0.5; % Parameter for kLa calculation [-]
        beta     = 0.5; % Parameter for kLa calculation [-]
        gamma    = -0.05; % Parameter for kLa calculation [-]
        KAlvol   = 4.62*10^(-3); % Stripping constant ammonia [-]

        % Constants concerning the model organism
        my1opt   = 0.46; % Maximum specific growth rate glucose [1/h]
        my2opt   = 0.35; % Maximum specific growth rate glycerol [1/h]
        my3opt   = 0.1; % Maximum specific growth rate acetate [1/h]

        qOpXm    = 5.0*10^(-3); % Specific O2 maintenance rate [1/h]
        qS1pXm   = 6.0*10^(-2); % Spcific substrate maintenance rate glucose [1/h]

        kS1      = 0.04; % Substrate limitation constant glucose [g/l]
        kS2      = 0.05; % Substrate limitation constant glycerol [g/l]
        kS3      = 0.06; % Substrate limitation constant acetate [g/l]
        kI21     = 1.8*10^(-3); % Inhibition of glycerine uptake in presence of glucose [g/l]
        kI31     = 1.8*10^(-3); % Inhibition of acetate uptake in presence of glucose [g/l]
        kI32     = 3*10^(-3); % Inhibition of acetate uptake due to glycerine [g/l]
        kIPO     = 0.5*10^(-3); % Inhibition of acetate production due to O2 [g/l]
        kO       = 4*10^(-4); % Oxygen limitation constant [g/l]

        yXpS1gr  = 0.553; % Substrate growth yield coefficient glucose [-]
        yXpS2gr  = 0.4; % Substrate growth yield coefficient glycerine [-]
        yXpS3gr  = 0.2; % Substrate growth yield coefficient acetate [-]
        yXpOgr   = 1.367; % Oxygen growth yield coefficient [-]
        yPpS1    = 0.2; % Product yield coefficient glucose [-]
        yPpS2    = 0.1; % Product yield coefficient glycerine [-]
        yPpS3    = 1.0; % Product yield coefficient acetate [-]
        yAlpXgr  = 0.185; % Ammonia yield coefficient cells [-]
        yAcpXgr  = 0.0; % Titration acid yield coefficient cells [-]
        yCpO     = 1.375; % Cell internal respiratory behavior [-]

        deltaCpO = 0.69; % Relation CO2/O2 transition [-]
        yXpS     = 0.21921; % Yield coefficient of gram cells per gram substrate [-]
        yXpO     = 0.40601; % Yield coefficient of gram cells per gram oxygen [-]

        % Constants concerning pH and heat dependency of growth
        thetaLmingr = 5; % Minimum liquid temperature required for growth [°C]
        thetaLmaxgr = 47; % Maximum liquid temperature required for growth [°C]
        thetaLoptgr = 35; % Optimal temperature required for groeth [°C]
        thetaDJ_WP  = 32; % Added by me, working point of cascade temperature slave controller [°C]

        CPLtot0     = 0; % Produced acid in liquid phase t = 0 h [mol/l]
        CB1Ltot0    = 0.098; % Molar buffer acid concentration in liquid phase t = 0 h [mol/l]
        CB2Ltot0    = 0.03; % Molar buffer base concentration in liquid phase t = 0 h [mol/l]
        CAlLtot0    = 0.059; % Ammonia concentration in liquid phase t = 0 h [mol/l]
        CAcLtot0    = 0; % Titration acid in liquid phase t = 0 h [mol/l]
        CAcT1tot    = 2.44; % pH-acid reservoir [mol/l]
        pHLmingr    = 4; % Minimum pH required for growth [-]
        pHLmaxgr    = 9; % Maximum pH required for growth [-]
        pHLoptgr    = 6.8; % Optimal pH required for growth [-]

        % Constants of the antifoam system
        qhpX     = 0.1; % Cellspecific foam build up rate [1/(g*h)]
        tauF0    = 0.0125; % Time constant of foam formation at cXL = 0 g/l [h]
        KFpX     = 0.09; % Foam consistency factor [1/g]
```

```
        KAF      = 4.0; % Cell antifoam adsorption constant [(g*h)/l]
        VAF      = 0.975; % Activity decline due to antifoam addition [-]
        AAFtast  = 100; % Antifoam-activity gain per addition step Ttast [1/h]
        Ttast    = 8.333*10^(-5); % Addition unit Ttast [h]

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Parameters given by system
        % Calibration parameters
        pGcal    = 1.5*10^5; % Pressure at calibration [n/m^2]
        xOGcal   = 0.2094; % Molar concentration of oxygen at calibration [-]
        xCGin    = 0.004; % % Molar concentration of carbon dioxide in aeration [-]

        % Parameters of reservoirs
        FRmax    = 0.5; % Maximum feed pump rate [l/h]
        cS1R     = 200; % Glucose concentration in reservoir [g/l]
        cOR      = 0.0; % Dissolved oxygen concentration in reservoir [g/l]
        CCRtot   = 0.0; % Total CO2 concentration in reservoir [mol/l]
        VR0      = 2.0; % Maximum volume of feed reservoir at t = 0 h [l]
        cS2R     = 0; % Concentration of glycerine in glucose reservoir [g/l]
        cXR      = 0; % Concentration of cells in glucose reservoir [g/l]
        CPRtot   = 0; % Concentration of product in glucose reservoir [g/l]

        CAlT2tot = 16.47; % pH-base reservoir concentration (ammonia) [mol/l]
        CCT1tot  = 0.0; % Total CO2 concentration in acid reservoir [mol/l]
        CCT2tot  = 0.0; % Total CO2 concentration in base reservoir [mol/l]
        cOT1     = 0.0; % Dissolved O2 concentration in acid reservoir [g/l]
        cOT2     = 0.0; % Dissolved O2 concentration in base reservoir [g/l]

        % Parameters of Mass Flow Controller
        FnAIRmax = 20.0; % Maximumm possible air aeration rate [l/min]
        FnO2max  = 5.0; % Maximum possible O2 aeration rate [l/min]
        FnN2max  = 5.0; % Maximum possible N2 aeration rate [l/min]
        FnCO2max = 2.0; % Maximum possible CO2 aeration rate [l/min]
        FnGmax   = 27.0; % Summed up maximum aeration rate [l/min]

        % Parameters for agitation control
        NStmax   = 1500; % Maximum possible stirrer speed [1/min]

        % Parameters for measuring system
        TMpO2    = 2.5*10^(-3); % Time constant pO2 measurement system [h] 2.5E-3 h = 9 s
        TMxO2    = 4.2*10^(-3); % Time constant xO2 measurement system [h] 4.2E-3 h = 15 s*/
        TMxCO2   = 4.2*10^(-3); % Time constant xCO2 measurement system [h]
        TMpH     = 4.2*10^(-3); % Time constant pH measurement system [h]

        % Parameters for pumps
        FHmax     = 5; % Maximum harvest pump rate [l/h]
        FFmax     = 0.1; % Maximum filtrate pump rate [l/h]
        FT1max    = 1.0; % Maximum titration rate of the acid pump [l/h]
        FT2max    = 1.0; % Maximum titration rate of the alkali pump [l/h]

        pO20      = 100; % Partial pressure of dissolved oxygen at t = 0 h [%]
        cOL100    = 1.0133*10^5*0.2094/(1.44*10^6); % O2-concentration in liquid phase at 100 % pO2-indication [g/l]
        cOLmax    = 10^5/(1.44*10^6); % Maximum potential O2-concentration in liquid phase [g/l]

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Parameters chosen by operator
        VLmin     = 5.0; % Minimum volume in reactor [l]
        VL0       = 10; % Initial volumme in reactor [l]
        VLmax     = 11; % Maximum volume in reactor [l]
        LWw       = 14; % Liquid weight setpoint [kg]
        tmax      = 5.0; % Time limit for cultivation [h]

        cS1L0     = 5.0; % Initial glucose concentration t = 0 h [g/l]
        cS2L0     = 0.0; % Initial glycerine concentration t = 0 h [g/l]
        cS3L0     = 0; % Initial acetate concentration t = 0 h [g/l]
        cXL0      = 3.0; % Initial cell concentration t = 0 h [g/l]

        FHrelw    = 20; % Harvest rate [% maximum pump rate]

        NStw      = 400; % Setpoint for agitation speed [1/min]
        FnAIRw    = 7.5; % Aeration rate AIR extern [l/min]
        FnN2w     = 0.0; % Aeration rate N2 extern [l/min]
        FnO2w     = 0.0; % Aeration rate O2 extern [l/min]
        FnCO2w    = 0.0; % Aeration rate CO2 extern [l/min]
        FnGw      = 6; % Value of chosen aeration rate [l/min]
        FRw       = 0.0; % Value of chosen constant feed rate [l/h]
        pO2w      = 20; % Value of pO2 setpoint [%]
        pHw       = 6.7; % Value of pH Setpoint [-]
        thetaLw   = 32; % Value of temperature Setpoint [°C]

        deltapGw  = 0.5; % Pressure setpoint [bar]
        ypH_SET   = 0.0; % Setpoint for difference between measured pH and wanted pH [-]
```

```

        KP_pH   = 10000 % Gain of P part of pH master controller [-]
        KP_pH2a = 1.0 % Gain of P part of acid pH slave controller [-]
        KP_pH2b = -1.0 % Gain of P part of base pH slave controller [-]

        KI_temp1  = 0.1 % Gain of I part of temperature master controller [-]
        KP_temp1  = 8.0 % Gain of P part of temperature master controller [-]
        KP_temp2h = 0.01 % Gain of P part of temperature heating slave controller [-]
        KP_temp2c = 10.0 % Gain of P part of temperature heating slave controller [-]

        KI_agi = 1000; % Gain of I part of agitation controller [-]
        KP_agi = 10; % Gain of P part of agitation controller [-]
        KD_agi = 0.015 % Gain of D part of agitation controller [-]

        KI_feed = -15.0 % Gain of I part of feed controller [-]
        KP_feed = -2.0 % Gain of P part of feed controller [-]
        KD_feed  = -0.009 % Gain of D part of feed controller [-]

        KI_aeration = 0.003 % Gain of I part of aeration controller [-]
        KP_aeration = 20.0 % Gain of P part of aeration controller [-]
        KD_aeration = 0.0188 % Gain of D part of aeration controller [-]

        KI_gasmix  = 0.003 % Gain of I part of gasmix controller [-]
        KP_gasmix  = 0.4 % Gain of P part of gasmix controller [-]
        KD_gasmix  = 0.0005 % Gain of D part of gasmix controller [-]

        KI_LW = 0.3 % Gain of I part of liquid weight controller [-]
        KP_LW = 10.0 % Gain of P part of liquid weight controller [-]
        KD_LW = 1.0 % Gain of D part of liquid weight controller [-]


        % Flags for different control options (turned off = 0 by default
        % when starting the control app)
        motor        = 1; % Signal for stirrer to be turned on/off
        aeration     = 1; % Signal for aeration to be turned on/off
        air          = 0; % Signal for aeration with AIR
        N2           = 0; % Signal for aeration with N2
        O2           = 0; % Signal for aeration with O2
        CO2          = 0; % Signal for aeration with CO2
        feed         = 0; % Signal for feed to be turned on/off
        harvest      = 0; % Signal for harvest to be turned on/off
        antifoam     = 0; % Signal for antifoam addition
        alkali       = 0; % Signal for alkali addition
        acid         = 0; % Signal for acid addition
        cooling      = 0; % Signal for cooling
        heating      = 0; % Signal for heating
        pH_mode      = 0; % Signal for automatic or manual pH control
        temp_mode    = 0; % Signal for automatic or manual temperature control
        pO2_agi      = 0; % Signal for pO2 agitation control
        pO2_feed     = 0; % Signal for pO2 feed control
        pO2_aeration = 0; % Signal for pO2 aeration control
        pO2_gasmix   = 0; % Signal for pO2 gasmix control
        LW_harvest   = 0; % Signal for liquid weight harvest control

        InocStart    = 0; % Signal for inoculation at start of simulation

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Parameters needed for internal simulation
        deltat    = 0.0005; % Time increments of simulation [h]
        dfn       = "Data"; % Data File Name
        pfn       = "Parameter"; % Parameter File Name
        dfnx % Data File Name plus timestamp
        pfnx % Parameter File Name plus timestamp
        filter    = {'*.xlsx';'*.txt'}; % Define loadable files
        erasestr  = {'.xlsx','.xls','.txt'}; % Define file endings that should be erased for display
        Version   = "Sophia's new and improved cultivation simulation <3" % Version of simulation app

        mds          = 0; % Data Saving Flag
        SelectedFile = "none"; % File selected for variable storage
        FileName     = "none"; % File Name ofloaded initial values
        DataLoaded   = 0; % Flag for data loading event

        ExpertMode   = 1; % Flag to set whether or not expert mode is accessible
        Password     = "BPA1" % Password for expert mode
        X            = "none" % Password message box
        SnapShot     = 0; % Flag for process start from snapshot of previous simulation
        PrevPlot     = 0; % Flag for plotting previous data

    end

    methods (Access = public)
        % Define a public function accessible for the simulation app that
```

```
        % refreshes the trend table
        function UITablefunc(app)
            app.TreeCheckedNodesChanged(app)
        end

    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.X = inputdlg('Please enter the password to access expert mode:');
            if app.X == app.Password
                uiwait(msgbox('Expert mode is now accessible.'));
            else
                uiwait(msgbox('Incorrect password. Expert mode is not accessible.'));
                app.ExpertMode = 0;
                app.pHParametersButton.Visible   = "off";
                app.tempParametersButton.Visible = "off";
                app.pO2ParametersButton.Visible  = "off";
                app.LWParametersButton.Visible   = "off";
            end

            % Disable non-needed switches and edit fields upon startup
            app.CoolingSwitch.Enable = "Off";
            app.HeatingSwitch.Enable = "Off";

            app.AcidSwitch.Enable   = "Off";
            app.AlkaliSwitch.Enable = "Off";

            app.FnGwlminEditField.Visible      = "off";
            app.FnGwlminEditFieldLabel.Visible = "off";

            app.xOGinwEditField.Visible      = "off";
            app.xOGinwEditFieldLabel.Visible = "off";

            % Make trend labels invisible upon app launch
            app.pHtrendLabel.Text   = "";
            app.temptrendLabel.Text = "";
            app.pO2trendLabel.Text  = "";
            app.mLtrendLabel.Text   = "";

            % Define ItemsData here as double scalars, since defining the
            % ItemsData in the Appdesigner UI results in them becoming
            % character vectors
            app.AntifoamadditionSwitch.ItemsData = [0,1];
            app.AlkaliSwitch.ItemsData   = [0,1];
            app.AcidSwitch.ItemsData     = [0,1];
            app.CoolingSwitch.ItemsData  = [0,1];
            app.HeatingSwitch.ItemsData  = [0,1];

            % Initialize lamp colors for active and unactive controls
            app.AntifoamLamp.Color     = "green";
            app.pHLamp.Color           = "green";
            app.TemperatureLamp.Color  = "green";
            app.LiquidWeightLamp.Color = "red";
            app.pO2Lamp.Color          = "red";

            if isfile("ControlAppOptions.txt")
                a = readcell("ControlAppOptions.txt");

                app.Mode_pHDropDown.Value      = a(1);
                app.Mode_pHDropDownValueChanged(app)

                app.Mode_tempDropDown.Value    = a(2);
                app.Mode_tempDropDownValueChanged(app)

                app.Mode_pO2DropDown.Value     = a(3);
                app.Mode_pO2DropDownValueChanged(app)

                app.Mode_harvestDropDown.Value = a(4);
                app.Mode_harvestDropDownValueChanged(app)

                a = cell2mat(a(5:end));

                app.NStw   = a(1);
                app.pHw    = a(2);
                app.FnAIRw = a(3);
                app.FnO2w  = a(4);
                app.pO2w   = a(5);
                app.LWw    = a(6);
```

```
                app.FnGw    = a(7);
                app.deltat = a(8);

                a = logical(a);

                app.DataStorageatStartCheckBox.Value = a(9);
                app.DataStorageatStartCheckBoxValueChanged(app)

                app.InoculationatStartCheckBox.Value = a(10);
                app.InoculationatStartCheckBoxValueChanged(app)
            end


            % Display initial values and setpoints upon control app startup
            app.NStwrpmEditField.Value    = app.NStw;
            app.FRwlhEditField.Value      = app.FRw;
            app.FHwEditField.Value        = app.FHrelw;
            app.FnAIRwlminEditField.Value = app.FnAIRw;
            app.FnO2wlminEditField.Value  = app.FnO2w;
            app.thetaLwCEditField.Value   = app.thetaLw;
            app.pHwEditField.Value        = app.pHw;
            app.pO2wEditField.Value       = app.pO2w;
            app.NStrpmEditField.Value     = app.NStw;
            app.mLwkgEditField.Value      = app.LWw;
            app.FnGwlminEditField.Value   = app.FnGw;
            app.deltathEditField.Value    = app.deltat;

            % Set up column descriptions upon startup
            app.UITable.ColumnName = {'Variable name','Current Value','Trend'};

            % Define window name
            app.UIFigure.Name = "Control App";
        end

        % Callback function: ExitButton, UIFigure
        function UIFigureCloseRequest(app, event)
            % Create window that asks the user to confirm closing the
            % simulation
            YN = uiconfirm(app.UIFigure,'Do you want to exit the program?', 'Close request');
            if strcmpi(YN,'OK')
                % Check if Figure App or Dialog App are open and delete them if
                % they are
                e = evalin('base','who'); % Get all the variable names present in the workspace

                if ismember('app.SimulationApp',e) % Check if FigureApp is one of them
                    delete(app.SimulationApp)
                end

                if ismember('app.DialogApp',e) % Check if Dialog Box is one of them
                    delete(app.DialogApp)
                end

                a = [cellstr(app.Mode_pHDropDown.Value) cellstr(app.Mode_tempDropDown.Value) cellstr(app.Mode_pO2DropDown.
                     Value) cellstr(app.Mode_harvestDropDown.Value)];
                b = num2cell([app.NStw app.pHw app.FnAIRw app.FnO2w app.pO2w app.LWw app.FnGw app.deltat double(app.
                     DataStorageatStartCheckBox.Value) double(app.InoculationatStartCheckBox.Value)]);

                a = [a b];
                writecell(a,"ControlAppOptions.txt");

                delete(app)
            end
        end

        % Button pushed function: StartSimulationButton
        function StartSimulationButtonPushed(app, event)
            if app.StartSimulationButton.Text == "Start Simulation"

                % Set Process Loading to zero
                app.SnapShot = 0;

                % Change button text
                app.StartSimulationButton.Text = "Stop Simulation";

                % Change indication light to green
                app.ProcessRunningLamp.Color = "green";

                % Display starting variables in edit fields
                app.NStwrpmEditField.Value        = app.NStw;
                app.FnAIRwlminEditField.Value     = app.FnAIRw;
                app.FnO2wlminEditField.Value      = app.FnO2w;
                app.FRwlhEditField.Value          = app.FRw;
```

```
                app.FHwEditField.Value            = app.FHrelw;

                % Open Simulation Window
                app.SimulationApp = FigureApp_ODESystem_Luttmann_pHTEMP(app);

                % Call figure app's public function
                cycle_func(app.SimulationApp);

                % Disable starting value edit button
                app.EditStartingVariablesButton.Enable = "off";

            else

                % Stop Simulation Timer
%                 stop(app.SimulationApp.Timer)

                % Delete Figure App
                delete(app.SimulationApp);

                % Change indication light to green
                app.ProcessRunningLamp.Color = "red";

                % Change Button Text back
                app.StartSimulationButton.Text = "Start Simulation";

                % Enable starting value edit button
                app.EditStartingVariablesButton.Enable = "on";
            end
        end

        % Button pushed function: ReadInStartingVariablesButton
        function ReadInStartingVariablesButtonPushed(app, event)
            % Read in variables from selected file
            app.SelectedFile = uigetfile(app.filter);
            if app.SelectedFile ~= "none"
                app.DataLoaded    = 1;
                app.FileName      = erase(convertCharsToStrings(app.SelectedFile),app.erasestr);
                strings           = ["cXL0","cS1L0","FRw","NStw","FnGw","VLmax","cS1R","VR0","tmax"];
                ReadData          = readcell(app.SelectedFile);

                app.SelectedFileEditField.Value = app.SelectedFile;

                % Initialize a stop execution variable that contains zeros with
                % as many entries as we have starting variables
                StopExecution(1:length(strings)) = 0;

                % Separate Header from Data
                Header = ReadData(1,:);
                Data   = ReadData(2,:);

                % Convert data type to string and then to numbers
                Data = string(Data);
                Data = str2double(Data);

                % Preallocate empty vector with as many zeros as we have
                % starting variables
                A = zeros(1:length(strings));

                % Sort Variables by Header into the order cXL0, cS1L0, FRw, NStw, FnGw, VLmax, cS1R, VR0 and tmax and store
                     the values in A
                for i = 1:length(strings)
                    idx = strcmp(Header,strings(i));
                    % Error message if string was not found in file
                    if sum(idx) == 0
                        errordlg(sprintf('Check Spelling of %s',strings(i)),'Variable Name Spelling Error');
                        StopExecution(i) = 1;
                    else
                        A(i) = Data(idx);

                        % Test variables for plausibility and give error
                        % message for values that are not a number (NaN)
                        if sum(isnan(A)) == 1
                            errordlg('Variables cannot be a character','Variable Type Error');
                            StopExecution(i) = 1;
                        end
                    end
                end

                % If no problem was identified, set the initial values and setpoints to the
                % ones found in the loaded file
                if StopExecution(1) == 0
                    app.cXL0  = A(1);
```

```
                end
                if StopExecution(2) == 0
                    app.cS1L0 = A(2);
                end
                if StopExecution(3) == 0
                    app.FRw   = A(3);
                end
                if StopExecution(4) == 0
                    app.NStw  = A(4);
                end
                if StopExecution(5) == 0
                    app.FnGw  = A(5);
                end
                if StopExecution(6) == 0
                    app.VLmax = A(6);
                end
                if StopExecution(7) == 0
                    app.cS1R  = A(7);
                end
                if StopExecution(8) == 0
                    app.VR0   = A(8);
                end
                if StopExecution(9) == 0
                    app.tmax  = A(9);
                end
            end

            app.DialogApp = DialogBox1(app);
        end

        % Button pushed function: SaveStartingVariablesButton
        function SaveStartingVariablesButtonPushed(app, event)
            % Choose a file to store variables in
            app.SelectedFile = uigetfile({'*.xlsx';'*.txt'},'File Selection','Select a File to save your parameters in');
            app.FileName     = erase(convertCharsToStrings(app.SelectedFile),app.erasestr);

            if app.SelectedFile ~= 0
                app.SelectedFileEditField.Value = app.SelectedFile;

                % Establish headers and current values
                strings = cellstr(["cXL0","cS1L0","FRw","NStw","FnGw","VLmax","cS1R","VR0","tmax"]);
                values = num2cell([app.cXL0 app.cS1L0 app.FRw app.NStw app.FnGw app.VLmax app.cS1R app.VR0 app.tmax]);

                % Read in raw data
                if app.SelectedFile ~= "none"
                    ReadData = readcell(app.SelectedFile);

                    % If file is not empty, update values
                    if isempty(ReadData) == 0

                        % Find each strings entry in raw data and change the data to
                        % the updated value
                        for i = 1:length(strings)
                            p = strcmp(ReadData(1,:),strings(i));
                            [x,y] = find(p == 1);
                            ReadData(x+1,y) = values(i);
                        end

                        % Save the updated values
                        writecell(ReadData,app.SelectedFile)
                    else
                        ReadData = [strings; values];
                        writecell(ReadData,app.SelectedFile)
                    end


                end
            end
        end

        % Button pushed function: EditStartingVariablesButton
        function EditStartingVariablesButtonPushed(app, event)
            % Disable the Edit Starting Variables button while dialog box is open
            app.EditStartingVariablesButton.Enable = 'off';

            % Open the options dialog and pass inputs
            app.DialogApp = DialogBox1(app);
        end

        % Button pushed function: DataStorageStartButton
        function DataStorageStartButtonPushed(app, event)
            if app.DataStorageStartButton.Text == "Data Storage Start"
```

```

            if app.StartSimulationButton.Text == "Stop Simulation"
                % Set the data saving flag to on
                app.mds = 1;
                app.DataStorageRunningLamp.Color = "green";

                % Define file names including the date and time of creation
                app.dfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of data file
                app.pfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of parameter file

                % Write the chosen parameters to the parameter file
                writeParameterFile(app.SimulationApp,sprintf("%s",app.Version));
                writeParameterFile(app.SimulationApp," ");
                writeParameterFile(app.SimulationApp,"Chosen Parameters");
                writeParameterFile(app.SimulationApp,sprintf("cXL0          : %3.2f g",app.cXL0));
                writeParameterFile(app.SimulationApp,sprintf("cS1L0          : %3.2f g",app.cS1L0));
                writeParameterFile(app.SimulationApp,sprintf("FRw          : %3.2f 1/h",app.FRw));
                writeParameterFile(app.SimulationApp,sprintf("NStw          : %3.2f g/g",app.NStw));
                writeParameterFile(app.SimulationApp,sprintf("FnGw          : %3.2f 1/h",app.FnGw));
                writeParameterFile(app.SimulationApp,sprintf("VLmax          : %3.2f 1/h",app.VLmax));
                writeParameterFile(app.SimulationApp,sprintf("cS1R          : %3.2f 1/h",app.cS1R));
                writeParameterFile(app.SimulationApp,sprintf("VR0          : %3.2f 1/h",app.VR0));
                writeParameterFile(app.SimulationApp,sprintf("tmax          : %3.2f 1/h",app.tmax));

                % Define headers for the file columns
                writeDataFile(app.SimulationApp,"Time                ;           t [s];        cXL [g/l];        cS1L [g/l
                     ];        cS2L [g/l];        cS3L [g/l];        pO2 [%];        thetaL [°C];        pHL [-];
                     xOG [-];        xCG [-];        VL [l];        NSt [rpm];        FnG [l/min];        FR [l/h];
                            deltapG [bar];        my [1/h];        CAltot [mol/l]");

                % Change text on push button
                app.DataStorageStartButton.Text = "Data Storage Stop";
            end
        else
            % If button is pressed again, data saving is stopped and the
            % data flag set to off
            app.DataStorageatStartCheckBox.Value = 0;
            app.mds = 0;
            app.DataStorageRunningLamp.Color = "red";
            app.DataStorageStartButton.Text = "Data Storage Start";
        end
    end

    % Value changed function: NStwrpmEditField
    function NStwrpmEditFieldValueChanged(app, event)
        % Change stirrer speed setpoint when edit field value changes
        % and turn on the motor flag if the value is larger than zero
        app.NStw = app.NStwrpmEditField.Value;
        if app.NStw > 0
            app.motor = 1;
        else
            app.motor = 0;
        end
    end

    % Value changed function: FnAIRwlminEditField
    function FnAIRwlminEditFieldValueChanged(app, event)
        % Change air aeration rate setpoint when edit field value changes
        % and turn on the aeration flag if the value is larger than
        % zero
        app.FnAIRw = app.FnAIRwlminEditField.Value;
        if app.FnAIRw > 0
            app.air      = 1;
            app.aeration = 1;
        else
            app.air = 0;
            if app.O2 == 0
                app.aeration = 0;
            else
                app.aeration = 1;
            end
        end
    end

    % Value changed function: FnO2wlminEditField
    function FnO2wlminEditFieldValueChanged(app, event)
        % Change O2 aeration rate setpoint when edit field value changes
        % and turn on the aeration flag if the value is larger than
        % zero
        app.FnO2w = app.FnO2wlminEditField.Value;
        if app.FnO2w > 0
            app.O2       = 1;
```

```
                app.aeration = 1;
            else
                app.O2 = 0;
                if app.air == 0
                    app.aeration = 0;
                else
                    app.aeration = 1;
                end
            end
        end

        % Value changed function: FRwlhEditField
        function FRwlhEditFieldValueChanged(app, event)
            % Change feed rate setpoint when edit field value changes
            % and turn on the feed flag if the value is larger than zero
            app.FRw = app.FRwlhEditField.Value;
            if app.FRw > 0
                app.feed = 1;
            else
                app.feed = 0;
            end
        end

        % Value changed function: FHwEditField
        function FHwEditFieldValueChanged(app, event)
            % Change harvest rate setpoint when edit field value changes
            % and turn on the harvest flag if the value is larger than zero
            app.FHrelw = app.FHwEditField.Value;
            if app.FHrelw > 0
                app.harvest = 1;
            else
                app.harvest = 0;
            end
        end

        % Value changed function: AntifoamadditionSwitch
        function AntifoamadditionSwitchValueChanged(app, event)
            % Set antifoam flag and lamp indication to on if the antifoam
            % control is activated
            app.antifoam = app.AntifoamadditionSwitch.Value;
            if app.antifoam == 1
                app.AntifoamLamp.Color = "green";
            else
                app.AntifoamLamp.Color = "red";
            end
        end

        % Value changed function: AlkaliSwitch
        function AlkaliSwitchValueChanged(app, event)
            app.alkali = app.AlkaliSwitch.Value; % Set flag for base addition if alkali switch is turned
        end

        % Value changed function: AcidSwitch
        function AcidSwitchValueChanged(app, event)
            app.acid = app.AcidSwitch.Value; % Set flag for acid addition if acid switch is turned
        end

        % Value changed function: CoolingSwitch
        function CoolingSwitchValueChanged(app, event)
            app.cooling = app.CoolingSwitch.Value; % Set flag for cooling if cooling switch is turned
        end

        % Value changed function: HeatingSwitch
        function HeatingSwitchValueChanged(app, event)
            app.heating = app.HeatingSwitch.Value; % Set flag for heating if heating switch is turned
        end

        % Value changed function: pO2wEditField
        function pO2wEditFieldValueChanged(app, event)
            app.pO2w = app.pO2wEditField.Value; % Change pO2 setpoint if the edit field value is changed
        end

        % Value changed function: thetaLwCEditField
        function thetaLwCEditFieldValueChanged(app, event)
            app.thetaLw = app.thetaLwCEditField.Value; % Change temperature setpoint if the edit field value is changed
        end

        % Value changed function: mLwkgEditField
        function mLwkgEditFieldValueChanged(app, event)
            app.LWw = app.mLwkgEditField.Value; % Change liquid weight setpoint if the edit field value is changed
        end

```

```
        % Callback function: Tree
        function TreeCheckedNodesChanged(app, event)
            if app.StartSimulationButton.Text == "Stop Simulation"
                % Load variables of interest into the Matrix M and create a
                % cell array containing the respective name at the same
                % index
                M      = [(app.SimulationApp.VL)' (app.SimulationApp.cXL)' (app.SimulationApp.cS1L)' (app.SimulationApp.cS2L)' (app.SimulationApp.cS3L)' (app.SimulationApp.thetaL)' (app.SimulationApp.pO2)' (app.SimulationApp.pHL)' (app.SimulationApp.xOG)' (app.SimulationApp.xCG)' (app.SimulationApp.RQ)' (app.SimulationApp.QO2max)' (app.SimulationApp.QCO2max)' (app.SimulationApp.OTR)' (app.SimulationApp.CTR)' (app.SimulationApp.NSt)' (app.SimulationApp.FnG)' (app.SimulationApp.xOGin)' (app.SimulationApp.FR)'];
                tabstr = {'VL [l]' 'cXL [g/l]' 'cS1L [g/l]' 'cS2L [g/l]' 'cS3L [g/l]' 'thetaL [°C]' 'pO2 [%]' 'pHL [-]' 'xOG [-]' 'xCG [-]' 'RQ [-]' 'QO2max [g/(l*h)]' 'QCO2max [g/(l*h)]' 'OTR [g/(l*h)]' 'CTR [g/(l*h)]' 'NSt [rpm]' 'FnG [l/min]' 'xOGin [-]' 'FR [l/h]'};

                a    = [8 6 7 1];
                b    = ["pHtrendLabel" "temptrendLabel" "pO2trendLabel" "mLtrendLabel"];

                checkedNodes = app.Tree.CheckedNodes;
                if ~isempty(checkedNodes)
                    % The nodes contain numeric NodeData from 1 to 19,
                    % corresponding to the column index of the same variables in the
                    % previously defined M, which is now loaded
                    data = [checkedNodes.NodeData];
                    n    = 1;

                    % For every variable that can theoretically be selected
                    % (15 in total), cycle through the loop below
                    for i = 1:19
                        if exist("data","var") == 1 && any(data == i)
                            % Interpolate the last 11 entires as a slpine
                            % or if not enough data has been generated yet
                            % interpolate all existing entries of the
                            % variable
                            if length(app.SimulationApp.t) < 12
                                pp = spline(app.SimulationApp.t,M(:,i));
                            else
                                pp = spline(app.SimulationApp.t((end-10):end),M((end-10):end,i));
                            end
                            % Extract the details from the spline, a piece-wise polynomial, by breaking it apart
                            [breaks,coefs,l,k,d] = unmkpp(pp);
                            % Create a new piece-wise polynomial
                            % corresponding to the derivative of the spline
                            pp2 = mkpp(breaks,repmat(k-1:-1:1,d*l,1).*coefs(:,1:k-1),d);
                            val = ppval(pp2,app.SimulationApp.t(end));
                            % If derivative at last entry is not zero, indicate increase or decrease with an arrow
                            if val > 0
                                ind = "$\nearrow\$";
                            elseif val < 0
                                ind = "$\searrow\$";
                            else
                                ind = "$\rightarrow\$";
                            end
                            % Convert value to cell type and write the
                            % value and indication with the variable name
                            % to the table
                            help          = M(end,i);
                            help          = num2cell(help);
                            data_tab(1,n) = help;
                            data_tab(2,n) = cellstr(ind);
                            headers(1,n)  = tabstr(i);
                            n = n+1;

                            for c = 1:length(a)
                                prop = b{c};
                                if i == a(c)
                                    app.(prop).Text = cellstr(ind);
                                end
                            end
                        end
                    end

                    if exist("headers","var") == 1
                        % If entries were generated, display them in the UI
                        % table
                        app.UITable.Data = ([headers;data_tab])';
                    end

                else
                    % Delete data in table if no nodes are selected
                    app.UITable.Data = [];
                end
```

```

                for c = 1:length(a)
                    prop = b{c};
                    if ~any(data == a(c))
                        app.(prop).Text = "";
                    end
                end
            end
        end

        % Value changed function: Mode_pHDropDown
        function Mode_pHDropDownValueChanged(app, event)
            % Select pH control mode and enable/disable manual controls
            % accordingly
            pHmode = app.Mode_pHDropDown.Value;
            if pHmode == "Manual"
                app.pH_mode             = 1;
                app.AcidSwitch.Enable   = "On";
                app.AlkaliSwitch.Enable = "On";
                app.pHLamp.Color        = "red";
            else
                app.pH_mode             = 0;
                app.AcidSwitch.Enable   = "Off";
                app.AlkaliSwitch.Enable = "Off";
                app.pHLamp.Color        = "green";
            end
        end

        % Value changed function: Mode_tempDropDown
        function Mode_tempDropDownValueChanged(app, event)
            % Select temperature control mode and enable/disable manual controls
            % accordingly
            tempmode = app.Mode_tempDropDown.Value;
            if tempmode == "Manual"
                app.temp_mode             = 1;
                app.HeatingSwitch.Enable  = "On";
                app.CoolingSwitch.Enable  = "On";
                app.TemperatureLamp.Color = "red";
            else
                app.temp_mode             = 0;
                app.HeatingSwitch.Enable  = "Off";
                app.CoolingSwitch.Enable  = "Off";
                app.TemperatureLamp.Color = "green";
            end
        end

        % Value changed function: Mode_pO2DropDown
        function Mode_pO2DropDownValueChanged(app, event)
            % Select pO2 control mode, set flags and enable/disable manual controls
            % accordingly
            pO2mode = app.Mode_pO2DropDown.Value;
            if pO2mode == "pO2-agitation"
                app.pO2_agi                           = 1;
                app.pO2_feed                          = 0;
                app.pO2_aeration                      = 0;
                app.pO2_gasmix                        = 0;
                app.FRwlhEditField.Enable             = "on";
                app.NStwrpmEditField.Enable           = "off";
                app.FHwEditField.Enable               = "on";
                app.pO2Lamp.Color                     = "green";
                app.NStwrpmEditFieldLabel.Visible     = "off";
                app.NStwrpmEditField.Visible          = "off";
                app.FRwlhEditFieldLabel.Visible       = "on";
                app.FRwlhEditField.Visible            = "on";
                app.FnGwlminEditField.Visible         = "off";
                app.FnGwlminEditFieldLabel.Visible    = "off";
                app.FnAIRwlminEditFieldLabel.Visible  = "on";
                app.FnAIRwlminEditField.Visible       = "on";
                app.FnO2wlminEditFieldLabel.Visible   = "on";
                app.FnO2wlminEditField.Visible        = "on";
                app.xOGinwEditField.Visible           = "off";
                app.xOGinwEditFieldLabel.Visible      = "off";
            elseif pO2mode == "pO2-feed"
                app.pO2_agi                           = 0;
                app.pO2_feed                          = 1;
                app.pO2_aeration                      = 0;
                app.pO2_gasmix                        = 0;
                app.FRwlhEditField.Enable             = "off";
                app.NStwrpmEditField.Enable           = "on";
                app.FHwEditField.Enable               = "on";
                app.pO2Lamp.Color                     = "green";
                app.NStwrpmEditFieldLabel.Visible     = "on";
```

```
                app.NStwrpmEditField.Visible          = "on";
                app.FRwlhEditFieldLabel.Visible       = "off";
                app.FRwlhEditField.Visible            = "off";
                app.FnGwlminEditField.Visible         = "off";
                app.FnGwlminEditFieldLabel.Visible    = "off";
                app.FnAIRwlminEditFieldLabel.Visible  = "on";
                app.FnAIRwlminEditField.Visible       = "on";
                app.FnO2wlminEditFieldLabel.Visible   = "on";
                app.FnO2wlminEditField.Visible        = "on";
                app.xOGinwEditField.Visible           = "off";
                app.xOGinwEditFieldLabel.Visible      = "off";
            elseif pO2mode == "Manual"
                app.pO2_agi                           = 0;
                app.pO2_feed                          = 0;
                app.pO2_aeration                      = 0;
                app.pO2_gasmix                        = 0;
                app.FRwlhEditField.Enable             = "on";
                app.NStwrpmEditField.Enable           = "on";
                app.FHwEditField.Enable               = "on";
                app.pO2Lamp.Color                     = "red";
                app.NStwrpmEditFieldLabel.Visible     = "on";
                app.NStwrpmEditField.Visible          = "on";
                app.FRwlhEditFieldLabel.Visible       = "on";
                app.FRwlhEditField.Visible            = "on";
                app.FnGwlminEditField.Visible         = "off";
                app.FnGwlminEditFieldLabel.Visible    = "off";
                app.FnAIRwlminEditFieldLabel.Visible  = "on";
                app.FnAIRwlminEditField.Visible       = "on";
                app.FnO2wlminEditFieldLabel.Visible   = "on";
                app.FnO2wlminEditField.Visible        = "on";
                app.xOGinwEditField.Visible           = "off";
                app.xOGinwEditFieldLabel.Visible      = "off";
            elseif  pO2mode == "pO2-aeration"
                app.pO2_agi                           = 0;
                app.pO2_feed                          = 0;
                app.pO2_aeration                      = 1;
                app.pO2_gasmix                        = 0;
                app.FRwlhEditField.Enable             = "on";
                app.NStwrpmEditField.Enable           = "on";
                app.FHwEditField.Enable               = "on";
                app.pO2Lamp.Color                     = "green";
                app.NStwrpmEditFieldLabel.Visible     = "on";
                app.NStwrpmEditField.Visible          = "on";
                app.FRwlhEditFieldLabel.Visible       = "on";
                app.FRwlhEditField.Visible            = "on";
                app.FnGwlminEditField.Visible         = "off";
                app.FnGwlminEditFieldLabel.Visible    = "off";
                app.FnAIRwlminEditFieldLabel.Visible  = "off";
                app.FnAIRwlminEditField.Visible       = "off";
                app.FnO2wlminEditFieldLabel.Visible   = "off";
                app.FnO2wlminEditField.Visible        = "off";
                app.xOGinwEditField.Visible           = "off";
                app.xOGinwEditFieldLabel.Visible      = "off";
            else
                app.pO2_agi                           = 0;
                app.pO2_feed                          = 0;
                app.pO2_aeration                      = 0;
                app.pO2_gasmix                        = 1;
                app.FRwlhEditField.Enable             = "on";
                app.NStwrpmEditField.Enable           = "on";
                app.FHwEditField.Enable               = "on";
                app.pO2Lamp.Color                     = "green";
                app.NStwrpmEditFieldLabel.Visible     = "on";
                app.NStwrpmEditField.Visible          = "on";
                app.FRwlhEditFieldLabel.Visible       = "on";
                app.FRwlhEditField.Visible            = "on";
                app.FnGwlminEditField.Visible         = "on";
                app.FnGwlminEditFieldLabel.Visible    = "on";
                app.FnAIRwlminEditFieldLabel.Visible  = "off";
                app.FnAIRwlminEditField.Visible       = "off";
                app.FnO2wlminEditFieldLabel.Visible   = "off";
                app.FnO2wlminEditField.Visible        = "off";
                app.xOGinwEditField.Visible           = "on";
                app.xOGinwEditFieldLabel.Visible      = "on";
            end
        end

        % Value changed function: Mode_harvestDropDown
        function Mode_harvestDropDownValueChanged(app, event)
            % Select liquid weight control mode and enable/disable manual controls
            % accordingly
            LWmode = app.Mode_harvestDropDown.Value;
```

```
            if LWmode == "Harvest"
                app.LW_harvest                      = 1;
                app.LiquidWeightLamp.Color          = "green";
                app.FHwEditField.Enable             = "off";
                app.HarvestSwitch.Enable            = "off";
            else
                app.LW_harvest                      = 0;
                app.LiquidWeightLamp.Color          = "red";
                app.FHwEditField.Enable             = "on";
                app.HarvestSwitch.Enable            = "on";
            end
        end

        % Value changed function: deltathEditField
        function deltathEditFieldValueChanged(app, event)
            app.deltat = app.deltathEditField.Value; % Change delta t if the edit field value changes
        end

        % Value changed function: FnGwlminEditField
        function FnGwlminEditFieldValueChanged(app, event)
            app.FnGw = app.FnGwlminEditField.Value; % Change aeration rate setpoint if edit field value changes
        end

        % Value changed function: InoculationatStartCheckBox
        function InoculationatStartCheckBoxValueChanged(app, event)
            app.InocStart = app.InoculationatStartCheckBox.Value;
        end

        % Button pushed function: LoadProcessButton
        function LoadProcessButtonPushed(app, event)
            app.SnapShot = 1;

            % Change button text
            app.StartSimulationButton.Text = "Stop Simulation";

            % Change indication light to green
            app.ProcessRunningLamp.Color = "green";

            % Display starting variables in edit fields
            app.NStwrpmEditField.Value        = app.NStw;
            app.FnAIRwlminEditField.Value     = app.FnAIRw;
            app.FnO2wlminEditField.Value      = app.FnO2w;
            app.FRwlhEditField.Value          = app.FRw;
            app.FHwEditField.Value            = app.FHrelw;

            % Open Simulation Window
            app.SimulationApp = FigureApp_ODESystem_Luttmann_pHTEMP(app);

            % Call figure app's public function
            cycle_func(app.SimulationApp);

            % Disable starting value edit button
            app.EditStartingVariablesButton.Enable = "off";
        end

        % Button pushed function: PlotPreviousDataButton
        function PlotPreviousDataButtonPushed(app, event)
            % Open Simulation Window
            app.SimulationApp = FigureApp_ODESystem_Luttmann_pHTEMP(app);

            app.PrevPlot = 1;

            prev_plot(app.SimulationApp);
        end

        % Value changed function: DataStorageatStartCheckBox
        function DataStorageatStartCheckBoxValueChanged(app, event)
            app.mds = app.DataStorageatStartCheckBox.Value;

            if app.mds == 1
                app.DataStorageRunningLamp.Color = "green";

                % Define file names including the date and time of creation
                app.dfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of data file
                app.pfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of parameter file

                % Change text on push button
                app.DataStorageStartButton.Text = "Data Storage Stop";
            else
                app.DataStorageRunningLamp.Color = "red";
                app.DataStorageStartButton.Text  = "Data Storage Start";
            end
```

```
        end

        % Button pushed function: pHParametersButton
        function pHParametersButtonPushed(app, event)
            % Disable the Edit Starting Variables button while dialog box is open
            app.pHParametersButton.Enable = 'off';

            % Open the options dialog and pass inputs
            app.DialogApp2 = DialogBox2(app);
        end

        % Button pushed function: tempParametersButton
        function tempParametersButtonPushed(app, event)
            % Disable the Edit Starting Variables button while dialog box is open
            app.tempParametersButton.Enable = 'off';

            % Open the options dialog and pass inputs
            app.DialogApp3 = DialogBox3(app);
        end

        % Button pushed function: pO2ParametersButton
        function pO2ParametersButtonPushed(app, event)
            % Disable the Edit Starting Variables button while dialog box is open
            app.pO2ParametersButton.Enable = 'off';

            % Open the options dialog and pass inputs
            app.DialogApp4 = DialogBox4(app);
        end

        % Button pushed function: LWParametersButton
        function LWParametersButtonPushed(app, event)
           % Disable the Edit Starting Variables button while dialog box is open
            app.LWParametersButton.Enable = 'off';

            % Open the options dialog and pass inputs
            app.DialogApp5 = DialogBox5(app);
        end

        % Value changed function: pHwEditField
        function pHwEditFieldValueChanged(app, event)
            app.pHw = app.pHwEditField.Value;
        end

        % Button pushed function: ExpertmodeButton
        function ExpertmodeButtonPushed(app, event)
            if app.ExpertMode == 0
                app.X = inputdlg('Please enter the password to access expert mode:');
                if app.X == app.Password
                    app.ExpertMode = 1;
                    app.pHParametersButton.Visible    = "on";
                    app.tempParametersButton.Visible = "on";
                    app.pO2ParametersButton.Visible  = "on";
                    app.LWParametersButton.Visible    = "on";
                    uiwait(msgbox('Expert mode is now accessible.'));
                else
                    uiwait(msgbox('Incorrect password. Expert mode is not accessible.'));
                end
            end
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 874 645];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create TabGroup
            app.TabGroup = uitabgroup(app.UIFigure);
            app.TabGroup.Position = [1 79 874 567];

            % Create ControlOptionsTab
            app.ControlOptionsTab = uitab(app.TabGroup);
            app.ControlOptionsTab.Title = 'Control Options';
            app.ControlOptionsTab.BackgroundColor = [1 1 1];
```

```
            % Create LiquidWeightPanel
            app.LiquidWeightPanel = uipanel(app.ControlOptionsTab);
            app.LiquidWeightPanel.Title = 'Liquid Weight';
            app.LiquidWeightPanel.BackgroundColor = [1 1 1];
            app.LiquidWeightPanel.FontWeight = 'bold';
            app.LiquidWeightPanel.FontSize = 14;
            app.LiquidWeightPanel.Position = [0 124 873 92];

            % Create HarvestSwitchLabel
            app.HarvestSwitchLabel = uilabel(app.LiquidWeightPanel);
            app.HarvestSwitchLabel.HorizontalAlignment = 'center';
            app.HarvestSwitchLabel.FontWeight = 'bold';
            app.HarvestSwitchLabel.Position = [447 17 50 22];
            app.HarvestSwitchLabel.Text = 'Harvest';

            % Create HarvestSwitch
            app.HarvestSwitch = uiswitch(app.LiquidWeightPanel, 'rocker');
            app.HarvestSwitch.Orientation = 'horizontal';
            app.HarvestSwitch.Tooltip = {'Turn on harvest at maximum harvest rate.'};
            app.HarvestSwitch.Position = [524 20 36 16];

            % Create mLwkgEditFieldLabel
            app.mLwkgEditFieldLabel = uilabel(app.LiquidWeightPanel);
            app.mLwkgEditFieldLabel.HorizontalAlignment = 'right';
            app.mLwkgEditFieldLabel.Position = [249 40 54 22];
            app.mLwkgEditFieldLabel.Text = 'mLw [kg]';

            % Create mLwkgEditField
            app.mLwkgEditField = uieditfield(app.LiquidWeightPanel, 'numeric');
            app.mLwkgEditField.ValueDisplayFormat = '%.1f';
            app.mLwkgEditField.ValueChangedFcn = createCallbackFcn(app, @mLwkgEditFieldValueChanged, true);
            app.mLwkgEditField.Tooltip = {'Enter a setpoint for the liquid weight value.'};
            app.mLwkgEditField.Position = [242 18 65 22];

            % Create mLkgEditFieldLabel
            app.mLkgEditFieldLabel = uilabel(app.LiquidWeightPanel);
            app.mLkgEditFieldLabel.HorizontalAlignment = 'right';
            app.mLkgEditFieldLabel.Position = [340 41 44 22];
            app.mLkgEditFieldLabel.Text = 'mL [kg]';

            % Create mLkgEditField
            app.mLkgEditField = uieditfield(app.LiquidWeightPanel, 'numeric');
            app.mLkgEditField.ValueDisplayFormat = '%.1f';
            app.mLkgEditField.Editable = 'off';
            app.mLkgEditField.Position = [329 18 65 22];

            % Create LiquidWeightLamp
            app.LiquidWeightLamp = uilamp(app.LiquidWeightPanel);
            app.LiquidWeightLamp.Position = [51 18 20 20];

            % Create Mode_harvestDropDownLabel
            app.Mode_harvestDropDownLabel = uilabel(app.LiquidWeightPanel);
            app.Mode_harvestDropDownLabel.HorizontalAlignment = 'right';
            app.Mode_harvestDropDownLabel.Position = [127 40 82 22];
            app.Mode_harvestDropDownLabel.Text = 'Mode_harvest';

            % Create Mode_harvestDropDown
            app.Mode_harvestDropDown = uidropdown(app.LiquidWeightPanel);
            app.Mode_harvestDropDown.Items = {'Manual', 'Harvest'};
            app.Mode_harvestDropDown.ValueChangedFcn = createCallbackFcn(app, @Mode_harvestDropDownValueChanged, true);
            app.Mode_harvestDropDown.Tooltip = {'Turn on/off liquid weight control (off per default).'};
            app.Mode_harvestDropDown.Position = [118 17 100 22];
            app.Mode_harvestDropDown.Value = 'Manual';

            % Create FHwEditFieldLabel
            app.FHwEditFieldLabel = uilabel(app.LiquidWeightPanel);
            app.FHwEditFieldLabel.HorizontalAlignment = 'right';
            app.FHwEditFieldLabel.Position = [617 18 54 22];
            app.FHwEditFieldLabel.Text = 'FHw [%] ';

            % Create FHwEditField
            app.FHwEditField = uieditfield(app.LiquidWeightPanel, 'numeric');
            app.FHwEditField.Limits = [0 100];
            app.FHwEditField.ValueDisplayFormat = '%.1f';
            app.FHwEditField.ValueChangedFcn = createCallbackFcn(app, @FHwEditFieldValueChanged, true);
            app.FHwEditField.Tooltip = {'Enter a setpoint for the harvest rate.'};
            app.FHwEditField.Position = [692 18 61 22];

            % Create mLtrendLabel
            app.mLtrendLabel = uilabel(app.LiquidWeightPanel);
            app.mLtrendLabel.Position = [401 18 50 22];
```

```
            app.mLtrendLabel.Text = 'mLtrend';

            % Create LWParametersButton
            app.LWParametersButton = uibutton(app.LiquidWeightPanel, 'push');
            app.LWParametersButton.ButtonPushedFcn = createCallbackFcn(app, @LWParametersButtonPushed, true);
            app.LWParametersButton.Tooltip = {'Adjust the gains of the liquid weight controller.'};
            app.LWParametersButton.Position = [772 17 83 23];
            app.LWParametersButton.Text = 'Parameters';

            % Create pHPanel
            app.pHPanel = uipanel(app.ControlOptionsTab);
            app.pHPanel.Title = 'pH';
            app.pHPanel.BackgroundColor = [1 1 1];
            app.pHPanel.FontWeight = 'bold';
            app.pHPanel.FontSize = 14;
            app.pHPanel.Position = [0 435 873 109];

            % Create AlkaliSwitchLabel
            app.AlkaliSwitchLabel = uilabel(app.pHPanel);
            app.AlkaliSwitchLabel.HorizontalAlignment = 'center';
            app.AlkaliSwitchLabel.FontWeight = 'bold';
            app.AlkaliSwitchLabel.Position = [460 53 38 22];
            app.AlkaliSwitchLabel.Text = 'Alkali';

            % Create AlkaliSwitch
            app.AlkaliSwitch = uiswitch(app.pHPanel, 'rocker');
            app.AlkaliSwitch.Orientation = 'horizontal';
            app.AlkaliSwitch.ValueChangedFcn = createCallbackFcn(app, @AlkaliSwitchValueChanged, true);
            app.AlkaliSwitch.Tooltip = {'Turn on alkali pump at maximum pump rate.'};
            app.AlkaliSwitch.Position = [527 55 36 16];

            % Create AcidSwitchLabel
            app.AcidSwitchLabel = uilabel(app.pHPanel);
            app.AcidSwitchLabel.HorizontalAlignment = 'center';
            app.AcidSwitchLabel.FontWeight = 'bold';
            app.AcidSwitchLabel.Position = [466 18 31 22];
            app.AcidSwitchLabel.Text = 'Acid';

            % Create AcidSwitch
            app.AcidSwitch = uiswitch(app.pHPanel, 'rocker');
            app.AcidSwitch.Orientation = 'horizontal';
            app.AcidSwitch.ValueChangedFcn = createCallbackFcn(app, @AcidSwitchValueChanged, true);
            app.AcidSwitch.Tooltip = {'Turn on acid pump at maximum pump rate.'};
            app.AcidSwitch.Position = [528 20 36 16];

            % Create pHwEditField
            app.pHwEditField = uieditfield(app.pHPanel, 'numeric');
            app.pHwEditField.ValueDisplayFormat = '%.1f';
            app.pHwEditField.ValueChangedFcn = createCallbackFcn(app, @pHwEditFieldValueChanged, true);
            app.pHwEditField.Tooltip = {'Enter a setpoint for the pH value.'};
            app.pHwEditField.Position = [243 34 64 22];

            % Create pHwEditFieldLabel
            app.pHwEditFieldLabel = uilabel(app.pHPanel);
            app.pHwEditFieldLabel.HorizontalAlignment = 'right';
            app.pHwEditFieldLabel.Position = [257 55 30 22];
            app.pHwEditFieldLabel.Text = 'pHw';

            % Create pHLEditFieldLabel
            app.pHLEditFieldLabel = uilabel(app.pHPanel);
            app.pHLEditFieldLabel.HorizontalAlignment = 'right';
            app.pHLEditFieldLabel.Position = [347 53 27 22];
            app.pHLEditFieldLabel.Text = 'pHL';

            % Create pHLEditField
            app.pHLEditField = uieditfield(app.pHPanel, 'numeric');
            app.pHLEditField.ValueDisplayFormat = '%.1f';
            app.pHLEditField.Editable = 'off';
            app.pHLEditField.Position = [329 34 65 22];

            % Create pHLamp
            app.pHLamp = uilamp(app.pHPanel);
            app.pHLamp.Position = [51 36 20 20];

            % Create Mode_pHDropDownLabel
            app.Mode_pHDropDownLabel = uilabel(app.pHPanel);
            app.Mode_pHDropDownLabel.HorizontalAlignment = 'right';
            app.Mode_pHDropDownLabel.Position = [139 58 58 22];
            app.Mode_pHDropDownLabel.Text = 'Mode_pH';

            % Create Mode_pHDropDown
            app.Mode_pHDropDown = uidropdown(app.pHPanel);
```

```
            app.Mode_pHDropDown.Items = {'Auto', 'Manual'};
            app.Mode_pHDropDown.ValueChangedFcn = createCallbackFcn(app, @Mode_pHDropDownValueChanged, true);
            app.Mode_pHDropDown.Tooltip = {'Turn on/off pH control (on per default).'};
            app.Mode_pHDropDown.Position = [118 34 100 22];
            app.Mode_pHDropDown.Value = 'Auto';

            % Create pHtrendLabel
            app.pHtrendLabel = uilabel(app.pHPanel);
            app.pHtrendLabel.Position = [405 35 48 22];
            app.pHtrendLabel.Text = 'pHtrend';

            % Create pHParametersButton
            app.pHParametersButton = uibutton(app.pHPanel, 'push');
            app.pHParametersButton.ButtonPushedFcn = createCallbackFcn(app, @pHParametersButtonPushed, true);
            app.pHParametersButton.Tooltip = {'Adjust the gains of the pH controller.'};
            app.pHParametersButton.Position = [772 33 83 23];
            app.pHParametersButton.Text = 'Parameters';

            % Create pO2Panel
            app.pO2Panel = uipanel(app.ControlOptionsTab);
            app.pO2Panel.Title = 'pO2';
            app.pO2Panel.BackgroundColor = [1 1 1];
            app.pO2Panel.FontWeight = 'bold';
            app.pO2Panel.FontSize = 14;
            app.pO2Panel.Position = [0 215 873 118];

            % Create pO2wEditFieldLabel
            app.pO2wEditFieldLabel = uilabel(app.pO2Panel);
            app.pO2wEditFieldLabel.HorizontalAlignment = 'right';
            app.pO2wEditFieldLabel.Position = [247 60 58 22];
            app.pO2wEditFieldLabel.Text = 'pO2w [%]';

            % Create pO2wEditField
            app.pO2wEditField = uieditfield(app.pO2Panel, 'numeric');
            app.pO2wEditField.ValueDisplayFormat = '%.1f';
            app.pO2wEditField.ValueChangedFcn = createCallbackFcn(app, @pO2wEditFieldValueChanged, true);
            app.pO2wEditField.Tooltip = {'Enter a setpoint for the pO2 value.'};
            app.pO2wEditField.Position = [243 38 65 22];

            % Create pO2EditFieldLabel
            app.pO2EditFieldLabel = uilabel(app.pO2Panel);
            app.pO2EditFieldLabel.HorizontalAlignment = 'right';
            app.pO2EditFieldLabel.Position = [337 59 49 22];
            app.pO2EditFieldLabel.Text = 'pO2 [%]';

            % Create pO2EditField
            app.pO2EditField = uieditfield(app.pO2Panel, 'numeric');
            app.pO2EditField.ValueDisplayFormat = '%.1f';
            app.pO2EditField.Editable = 'off';
            app.pO2EditField.Position = [329 38 65 22];

            % Create NStwrpmEditFieldLabel
            app.NStwrpmEditFieldLabel = uilabel(app.pO2Panel);
            app.NStwrpmEditFieldLabel.HorizontalAlignment = 'right';
            app.NStwrpmEditFieldLabel.Position = [442 65 65 22];
            app.NStwrpmEditFieldLabel.Text = 'NStw [rpm]';

            % Create NStwrpmEditField
            app.NStwrpmEditField = uieditfield(app.pO2Panel, 'numeric');
            app.NStwrpmEditField.Limits = [0 1500];
            app.NStwrpmEditField.ValueDisplayFormat = '%.1f';
            app.NStwrpmEditField.ValueChangedFcn = createCallbackFcn(app, @NStwrpmEditFieldValueChanged, true);
            app.NStwrpmEditField.Tooltip = {'Enter a setpoint for the stirrer speed.'};
            app.NStwrpmEditField.Position = [525 65 62 22];

            % Create FRwlhEditFieldLabel
            app.FRwlhEditFieldLabel = uilabel(app.pO2Panel);
            app.FRwlhEditFieldLabel.HorizontalAlignment = 'right';
            app.FRwlhEditFieldLabel.Position = [453 37 53 22];
            app.FRwlhEditFieldLabel.Text = 'FRw [l/h]';

            % Create FRwlhEditField
            app.FRwlhEditField = uieditfield(app.pO2Panel, 'numeric');
            app.FRwlhEditField.Limits = [0 0.5];
            app.FRwlhEditField.ValueDisplayFormat = '%.4f';
            app.FRwlhEditField.ValueChangedFcn = createCallbackFcn(app, @FRwlhEditFieldValueChanged, true);
            app.FRwlhEditField.Tooltip = {'Enter a setpoint for the feed rate.'};
            app.FRwlhEditField.Position = [525 37 62 22];

            % Create FnAIRwlminEditFieldLabel
            app.FnAIRwlminEditFieldLabel = uilabel(app.pO2Panel);
            app.FnAIRwlminEditFieldLabel.HorizontalAlignment = 'right';
```

```
            app.FnAIRwlminEditFieldLabel.Position = [594 65 83 22];
            app.FnAIRwlminEditFieldLabel.Text = 'FnAIRw [l/min]';

            % Create FnAIRwlminEditField
            app.FnAIRwlminEditField = uieditfield(app.pO2Panel, 'numeric');
            app.FnAIRwlminEditField.Limits = [0 15];
            app.FnAIRwlminEditField.ValueDisplayFormat = '%.1f';
            app.FnAIRwlminEditField.ValueChangedFcn = createCallbackFcn(app, @FnAIRwlminEditFieldValueChanged, true);
            app.FnAIRwlminEditField.Tooltip = {'Enter a setpoint for the air aeration rate.'};
            app.FnAIRwlminEditField.Position = [693 65 62 22];

            % Create FnO2wlminEditFieldLabel
            app.FnO2wlminEditFieldLabel = uilabel(app.pO2Panel);
            app.FnO2wlminEditFieldLabel.HorizontalAlignment = 'right';
            app.FnO2wlminEditFieldLabel.Position = [598 37 79 22];
            app.FnO2wlminEditFieldLabel.Text = 'FnO2w [l/min]';

            % Create FnO2wlminEditField
            app.FnO2wlminEditField = uieditfield(app.pO2Panel, 'numeric');
            app.FnO2wlminEditField.Limits = [0 10];
            app.FnO2wlminEditField.ValueDisplayFormat = '%.1f';
            app.FnO2wlminEditField.ValueChangedFcn = createCallbackFcn(app, @FnO2wlminEditFieldValueChanged, true);
            app.FnO2wlminEditField.Tooltip = {'Enter a setpoint for the O2 aeration rate.'};
            app.FnO2wlminEditField.Position = [693 37 62 22];

            % Create pO2Lamp
            app.pO2Lamp = uilamp(app.pO2Panel);
            app.pO2Lamp.Position = [53 39 20 20];

            % Create Mode_pO2DropDownLabel
            app.Mode_pO2DropDownLabel = uilabel(app.pO2Panel);
            app.Mode_pO2DropDownLabel.HorizontalAlignment = 'center';
            app.Mode_pO2DropDownLabel.Position = [136 62 65 22];
            app.Mode_pO2DropDownLabel.Text = 'Mode_pO2';

            % Create Mode_pO2DropDown
            app.Mode_pO2DropDown = uidropdown(app.pO2Panel);
            app.Mode_pO2DropDown.Items = {'Manual', 'pO2-feed', 'pO2-agitation', 'pO2-aeration', 'pO2-gasmix'};
            app.Mode_pO2DropDown.ValueChangedFcn = createCallbackFcn(app, @Mode_pO2DropDownValueChanged, true);
            app.Mode_pO2DropDown.Tooltip = {'Turn on/off pO2 control and select between different control modes (off per
                 default).'};
            app.Mode_pO2DropDown.Position = [118 39 100 22];
            app.Mode_pO2DropDown.Value = 'Manual';

            % Create FnGwlminEditFieldLabel
            app.FnGwlminEditFieldLabel = uilabel(app.pO2Panel);
            app.FnGwlminEditFieldLabel.HorizontalAlignment = 'right';
            app.FnGwlminEditFieldLabel.Position = [433 9 73 22];
            app.FnGwlminEditFieldLabel.Text = 'FnGw [l/min]';

            % Create FnGwlminEditField
            app.FnGwlminEditField = uieditfield(app.pO2Panel, 'numeric');
            app.FnGwlminEditField.Limits = [0 25];
            app.FnGwlminEditField.ValueDisplayFormat = '%.1f';
            app.FnGwlminEditField.ValueChangedFcn = createCallbackFcn(app, @FnGwlminEditFieldValueChanged, true);
            app.FnGwlminEditField.Tooltip = {'Enter a setpoint for the overall aeration rate.'};
            app.FnGwlminEditField.Position = [525 8 62 22];

            % Create xOGinwEditFieldLabel
            app.xOGinwEditFieldLabel = uilabel(app.pO2Panel);
            app.xOGinwEditFieldLabel.HorizontalAlignment = 'right';
            app.xOGinwEditFieldLabel.Position = [615 9 62 22];
            app.xOGinwEditFieldLabel.Text = 'xOGinw [-]';

            % Create xOGinwEditField
            app.xOGinwEditField = uieditfield(app.pO2Panel, 'numeric');
            app.xOGinwEditField.Limits = [0 1];
            app.xOGinwEditField.ValueDisplayFormat = '%.4f';
            app.xOGinwEditField.Editable = 'off';
            app.xOGinwEditField.Tooltip = {'Calculated setpoint for the molar fraction of O2 in the gas phase at the reactor
                 inlet.'};
            app.xOGinwEditField.Position = [693 9 62 22];

            % Create pO2trendLabel
            app.pO2trendLabel = uilabel(app.pO2Panel);
            app.pO2trendLabel.Position = [405 39 55 22];
            app.pO2trendLabel.Text = 'pO2trend';

            % Create pO2ParametersButton
            app.pO2ParametersButton = uibutton(app.pO2Panel, 'push');
            app.pO2ParametersButton.ButtonPushedFcn = createCallbackFcn(app, @pO2ParametersButtonPushed, true);
            app.pO2ParametersButton.Tooltip = {'Adjust the gains of the pO2 controllers.'};
```

```
            app.pO2ParametersButton.Position = [772 36 83 23];
            app.pO2ParametersButton.Text = 'Parameters';

            % Create TemperaturePanel
            app.TemperaturePanel = uipanel(app.ControlOptionsTab);
            app.TemperaturePanel.Title = 'Temperature';
            app.TemperaturePanel.BackgroundColor = [1 1 1];
            app.TemperaturePanel.FontWeight = 'bold';
            app.TemperaturePanel.FontSize = 14;
            app.TemperaturePanel.Position = [0 331 873 106];

            % Create CoolingSwitchLabel
            app.CoolingSwitchLabel = uilabel(app.TemperaturePanel);
            app.CoolingSwitchLabel.HorizontalAlignment = 'center';
            app.CoolingSwitchLabel.FontWeight = 'bold';
            app.CoolingSwitchLabel.Position = [451 49 50 22];
            app.CoolingSwitchLabel.Text = 'Cooling';

            % Create CoolingSwitch
            app.CoolingSwitch = uiswitch(app.TemperaturePanel, 'rocker');
            app.CoolingSwitch.Orientation = 'horizontal';
            app.CoolingSwitch.ValueChangedFcn = createCallbackFcn(app, @CoolingSwitchValueChanged, true);
            app.CoolingSwitch.Tooltip = {'Turn on cooling water flow at maximum flow rate.'};
            app.CoolingSwitch.Position = [527 51 36 16];

            % Create HeatingSwitchLabel
            app.HeatingSwitchLabel = uilabel(app.TemperaturePanel);
            app.HeatingSwitchLabel.HorizontalAlignment = 'center';
            app.HeatingSwitchLabel.FontWeight = 'bold';
            app.HeatingSwitchLabel.Position = [450 12 50 22];
            app.HeatingSwitchLabel.Text = 'Heating';

            % Create HeatingSwitch
            app.HeatingSwitch = uiswitch(app.TemperaturePanel, 'rocker');
            app.HeatingSwitch.Orientation = 'horizontal';
            app.HeatingSwitch.ValueChangedFcn = createCallbackFcn(app, @HeatingSwitchValueChanged, true);
            app.HeatingSwitch.Tooltip = {'Turn on heating at maximum heating power.'};
            app.HeatingSwitch.Position = [527 15 36 16];

            % Create thetaLwCEditFieldLabel
            app.thetaLwCEditFieldLabel = uilabel(app.TemperaturePanel);
            app.thetaLwCEditFieldLabel.HorizontalAlignment = 'right';
            app.thetaLwCEditFieldLabel.Position = [241 49 71 22];
            app.thetaLwCEditFieldLabel.Text = 'thetaLw [°C]';

            % Create thetaLwCEditField
            app.thetaLwCEditField = uieditfield(app.TemperaturePanel, 'numeric');
            app.thetaLwCEditField.ValueDisplayFormat = '%.1f';
            app.thetaLwCEditField.ValueChangedFcn = createCallbackFcn(app, @thetaLwCEditFieldValueChanged, true);
            app.thetaLwCEditField.Tooltip = {'Enter a setpoint for the temperature value.'};
            app.thetaLwCEditField.Position = [243 28 65 22];

            % Create thetaLCEditFieldLabel
            app.thetaLCEditFieldLabel = uilabel(app.TemperaturePanel);
            app.thetaLCEditFieldLabel.HorizontalAlignment = 'right';
            app.thetaLCEditFieldLabel.Position = [331 49 65 22];
            app.thetaLCEditFieldLabel.Text = 'thetaL [°C] ';

            % Create thetaLCEditField
            app.thetaLCEditField = uieditfield(app.TemperaturePanel, 'numeric');
            app.thetaLCEditField.ValueDisplayFormat = '%.1f';
            app.thetaLCEditField.Editable = 'off';
            app.thetaLCEditField.Position = [329 28 65 22];

            % Create Mode_tempDropDownLabel
            app.Mode_tempDropDownLabel = uilabel(app.TemperaturePanel);
            app.Mode_tempDropDownLabel.HorizontalAlignment = 'right';
            app.Mode_tempDropDownLabel.Position = [129 52 69 22];
            app.Mode_tempDropDownLabel.Text = 'Mode_temp';

            % Create Mode_tempDropDown
            app.Mode_tempDropDown = uidropdown(app.TemperaturePanel);
            app.Mode_tempDropDown.Items = {'Auto', 'Manual'};
            app.Mode_tempDropDown.ValueChangedFcn = createCallbackFcn(app, @Mode_tempDropDownValueChanged, true);
            app.Mode_tempDropDown.Tooltip = {'Turn on/off temperature control (on per default).'};
            app.Mode_tempDropDown.Position = [118 29 100 22];
            app.Mode_tempDropDown.Value = 'Auto';

            % Create TemperatureLamp
            app.TemperatureLamp = uilamp(app.TemperaturePanel);
            app.TemperatureLamp.Position = [52 28 20 20];

```

```
            % Create temptrendLabel
            app.temptrendLabel = uilabel(app.TemperaturePanel);
            app.temptrendLabel.Position = [405 29 59 22];
            app.temptrendLabel.Text = 'temptrend';

            % Create tempParametersButton
            app.tempParametersButton = uibutton(app.TemperaturePanel, 'push');
            app.tempParametersButton.ButtonPushedFcn = createCallbackFcn(app, @tempParametersButtonPushed, true);
            app.tempParametersButton.Tooltip = {'Adjust the gains of the temperature controller.'};
            app.tempParametersButton.Position = [772 27 83 23];
            app.tempParametersButton.Text = 'Parameters';

            % Create AntifoamPanel
            app.AntifoamPanel = uipanel(app.ControlOptionsTab);
            app.AntifoamPanel.Title = 'Antifoam';
            app.AntifoamPanel.BackgroundColor = [1 1 1];
            app.AntifoamPanel.FontWeight = 'bold';
            app.AntifoamPanel.FontSize = 14;
            app.AntifoamPanel.Position = [0 1 873 124];

            % Create AntifoamadditionSwitch
            app.AntifoamadditionSwitch = uiswitch(app.AntifoamPanel, 'rocker');
            app.AntifoamadditionSwitch.ValueChangedFcn = createCallbackFcn(app, @AntifoamadditionSwitchValueChanged, true);
            app.AntifoamadditionSwitch.Tooltip = {'Turn on/off antifoam control (on per default).'};
            app.AntifoamadditionSwitch.Position = [160 36 16 36];
            app.AntifoamadditionSwitch.Value = 'On';

            % Create AntifoamLamp
            app.AntifoamLamp = uilamp(app.AntifoamPanel);
            app.AntifoamLamp.Position = [52 44 20 20];

            % Create VariablePoolTab
            app.VariablePoolTab = uitab(app.TabGroup);
            app.VariablePoolTab.Title = 'Variable Pool';
            app.VariablePoolTab.BackgroundColor = [1 1 1];

            % Create UITable
            app.UITable = uitable(app.VariablePoolTab);
            app.UITable.ColumnName = {'Column 1'; 'Column 2'; 'Column 3'; 'Column 4'};
            app.UITable.RowName = {};
            app.UITable.Tooltip = {'Trend table populated by selecting variables from the variable list to the left.'};
            app.UITable.Position = [205 1 322 517];

            % Create Tree
            app.Tree = uitree(app.VariablePoolTab, 'checkbox');
            app.Tree.Tooltip = {'Select variables to generate a trend table entry for.'};
            app.Tree.Position = [20 15 164 489];

            % Create DisplayableVariablesNode
            app.DisplayableVariablesNode = uitreenode(app.Tree);
            app.DisplayableVariablesNode.Text = 'Displayable Variables';

            % Create VLlNode
            app.VLlNode = uitreenode(app.DisplayableVariablesNode);
            app.VLlNode.NodeData = 1;
            app.VLlNode.Text = 'VL [l]';

            % Create cXLglNode
            app.cXLglNode = uitreenode(app.DisplayableVariablesNode);
            app.cXLglNode.NodeData = 2;
            app.cXLglNode.Text = 'cXL [g/l]';

            % Create cS1LglNode
            app.cS1LglNode = uitreenode(app.DisplayableVariablesNode);
            app.cS1LglNode.NodeData = 3;
            app.cS1LglNode.Text = 'cS1L [g/l]';

            % Create cS2LglNode
            app.cS2LglNode = uitreenode(app.DisplayableVariablesNode);
            app.cS2LglNode.NodeData = 4;
            app.cS2LglNode.Text = 'cS2L [g/l]';

            % Create cS3LglNode
            app.cS3LglNode = uitreenode(app.DisplayableVariablesNode);
            app.cS3LglNode.NodeData = 5;
            app.cS3LglNode.Text = 'cS3L [g/l]';

            % Create thetaLCNode
            app.thetaLCNode = uitreenode(app.DisplayableVariablesNode);
            app.thetaLCNode.NodeData = 6;
            app.thetaLCNode.Text = 'thetaL [°C]';

```

```
            % Create pO2Node
            app.pO2Node = uitreenode(app.DisplayableVariablesNode);
            app.pO2Node.NodeData = 7;
            app.pO2Node.Text = 'pO2 [%]';

            % Create pHLNode
            app.pHLNode = uitreenode(app.DisplayableVariablesNode);
            app.pHLNode.NodeData = 8;
            app.pHLNode.Text = 'pHL [-]';

            % Create xOGNode
            app.xOGNode = uitreenode(app.DisplayableVariablesNode);
            app.xOGNode.NodeData = 9;
            app.xOGNode.Text = 'xOG [-]';

            % Create xCGNode
            app.xCGNode = uitreenode(app.DisplayableVariablesNode);
            app.xCGNode.NodeData = 10;
            app.xCGNode.Text = 'xCG [-]';

            % Create RQmolmolNode
            app.RQmolmolNode = uitreenode(app.DisplayableVariablesNode);
            app.RQmolmolNode.NodeData = 11;
            app.RQmolmolNode.Text = 'RQ [mol/mol]';

            % Create QO2maxglhNode
            app.QO2maxglhNode = uitreenode(app.DisplayableVariablesNode);
            app.QO2maxglhNode.NodeData = 12;
            app.QO2maxglhNode.Text = 'QO2max [g/(l*h)]';

            % Create QCO2maxglhNode
            app.QCO2maxglhNode = uitreenode(app.DisplayableVariablesNode);
            app.QCO2maxglhNode.NodeData = 13;
            app.QCO2maxglhNode.Text = 'QCO2max [g/(l*h)]';

            % Create OTRglhNode
            app.OTRglhNode = uitreenode(app.DisplayableVariablesNode);
            app.OTRglhNode.NodeData = 14;
            app.OTRglhNode.Text = 'OTR [g/(l*h)]';

            % Create CTRglhNode
            app.CTRglhNode = uitreenode(app.DisplayableVariablesNode);
            app.CTRglhNode.NodeData = 15;
            app.CTRglhNode.Text = 'CTR [g/l*h)]';

            % Create NStrpmNode
            app.NStrpmNode = uitreenode(app.DisplayableVariablesNode);
            app.NStrpmNode.NodeData = 16;
            app.NStrpmNode.Text = 'NSt [rpm]';

            % Create FnGlminNode
            app.FnGlminNode = uitreenode(app.DisplayableVariablesNode);
            app.FnGlminNode.NodeData = 17;
            app.FnGlminNode.Text = 'FnG [l/min]';

            % Create xOGinNode
            app.xOGinNode = uitreenode(app.DisplayableVariablesNode);
            app.xOGinNode.NodeData = 18;
            app.xOGinNode.Text = 'xOGin [-]';

            % Create FRlhNode
            app.FRlhNode = uitreenode(app.DisplayableVariablesNode);
            app.FRlhNode.NodeData = 19;
            app.FRlhNode.Text = 'FR [l/h]';

            % Assign Checked Nodes
            app.Tree.CheckedNodesChangedFcn = createCallbackFcn(app, @TreeCheckedNodesChanged, true);

            % Create LiquidVolumePanel
            app.LiquidVolumePanel = uipanel(app.VariablePoolTab);
            app.LiquidVolumePanel.Title = 'Liquid Volume';
            app.LiquidVolumePanel.BackgroundColor = [1 1 1];
            app.LiquidVolumePanel.FontWeight = 'bold';
            app.LiquidVolumePanel.FontSize = 14;
            app.LiquidVolumePanel.Position = [580 213 193 174];

            % Create VLlEditFieldLabel
            app.VLlEditFieldLabel = uilabel(app.LiquidVolumePanel);
            app.VLlEditFieldLabel.FontWeight = 'bold';
            app.VLlEditFieldLabel.Position = [16 116 35 22];
            app.VLlEditFieldLabel.Text = 'VL [l]';
```

```
            % Create VLlEditField
            app.VLlEditField = uieditfield(app.LiquidVolumePanel, 'numeric');
            app.VLlEditField.ValueDisplayFormat = '%.1f';
            app.VLlEditField.Editable = 'off';
            app.VLlEditField.FontWeight = 'bold';
            app.VLlEditField.Position = [105 116 71 22];

            % Create addedbaselEditFieldLabel
            app.addedbaselEditFieldLabel = uilabel(app.LiquidVolumePanel);
            app.addedbaselEditFieldLabel.Position = [15 91 81 22];
            app.addedbaselEditFieldLabel.Text = 'added base [l]';

            % Create addedbaselEditField
            app.addedbaselEditField = uieditfield(app.LiquidVolumePanel, 'numeric');
            app.addedbaselEditField.ValueDisplayFormat = '%.1f';
            app.addedbaselEditField.Editable = 'off';
            app.addedbaselEditField.Position = [105 91 71 22];

            % Create addedacidlEditFieldLabel
            app.addedacidlEditFieldLabel = uilabel(app.LiquidVolumePanel);
            app.addedacidlEditFieldLabel.Position = [15 66 77 22];
            app.addedacidlEditFieldLabel.Text = 'added acid [l]';

            % Create addedacidlEditField
            app.addedacidlEditField = uieditfield(app.LiquidVolumePanel, 'numeric');
            app.addedacidlEditField.ValueDisplayFormat = '%.1f';
            app.addedacidlEditField.Editable = 'off';
            app.addedacidlEditField.Position = [105 66 71 22];

            % Create addedAFlEditFieldLabel
            app.addedAFlEditFieldLabel = uilabel(app.LiquidVolumePanel);
            app.addedAFlEditFieldLabel.Position = [15 41 70 22];
            app.addedAFlEditFieldLabel.Text = 'added AF [l]';

            % Create addedAFlEditField
            app.addedAFlEditField = uieditfield(app.LiquidVolumePanel, 'numeric');
            app.addedAFlEditField.ValueDisplayFormat = '%.1f';
            app.addedAFlEditField.Editable = 'off';
            app.addedAFlEditField.Position = [105 41 71 22];

            % Create addedfeedlEditFieldLabel
            app.addedfeedlEditFieldLabel = uilabel(app.LiquidVolumePanel);
            app.addedfeedlEditFieldLabel.Position = [15 16 78 22];
            app.addedfeedlEditFieldLabel.Text = 'added feed [l]';

            % Create addedfeedlEditField
            app.addedfeedlEditField = uieditfield(app.LiquidVolumePanel, 'numeric');
            app.addedfeedlEditField.ValueDisplayFormat = '%.1f';
            app.addedfeedlEditField.Editable = 'off';
            app.addedfeedlEditField.Position = [105 16 71 22];

            % Create FT1acidlhEditField
            app.FT1acidlhEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.FT1acidlhEditField.ValueDisplayFormat = '%.3f';
            app.FT1acidlhEditField.Editable = 'off';
            app.FT1acidlhEditField.Position = [581 476 65 22];

            % Create FT1acidlhEditFieldLabel
            app.FT1acidlhEditFieldLabel = uilabel(app.VariablePoolTab);
            app.FT1acidlhEditFieldLabel.Position = [580 496 83 22];
            app.FT1acidlhEditFieldLabel.Text = 'FT1 (acid) [l/h]';

            % Create FT2baselhEditField
            app.FT2baselhEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.FT2baselhEditField.ValueDisplayFormat = '%.3f';
            app.FT2baselhEditField.Editable = 'off';
            app.FT2baselhEditField.Position = [581 429 65 22];

            % Create FT2baselhEditFieldLabel
            app.FT2baselhEditFieldLabel = uilabel(app.VariablePoolTab);
            app.FT2baselhEditFieldLabel.Position = [581 449 87 22];
            app.FT2baselhEditFieldLabel.Text = 'FT2 (base) [l/h]';

            % Create FRlhEditFieldLabel
            app.FRlhEditFieldLabel = uilabel(app.VariablePoolTab);
            app.FRlhEditFieldLabel.Position = [681 496 44 22];
            app.FRlhEditFieldLabel.Text = 'FR [l/h]';

            % Create FRlhEditField
            app.FRlhEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.FRlhEditField.ValueDisplayFormat = '%.3f';
            app.FRlhEditField.Editable = 'off';
```

```
            app.FRlhEditField.Position = [681 476 65 22];

            % Create NStrpmEditFieldLabel
            app.NStrpmEditFieldLabel = uilabel(app.VariablePoolTab);
            app.NStrpmEditFieldLabel.Position = [583 99 56 22];
            app.NStrpmEditFieldLabel.Text = 'NSt [rpm]';

            % Create NStrpmEditField
            app.NStrpmEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.NStrpmEditField.ValueDisplayFormat = '%.1f';
            app.NStrpmEditField.Editable = 'off';
            app.NStrpmEditField.Position = [581 79 65 22];

            % Create FnGlminEditFieldLabel
            app.FnGlminEditFieldLabel = uilabel(app.VariablePoolTab);
            app.FnGlminEditFieldLabel.Position = [581 149 64 22];
            app.FnGlminEditFieldLabel.Text = 'FnG [l/min]';

            % Create FnGlminEditField
            app.FnGlminEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.FnGlminEditField.ValueDisplayFormat = '%.1f';
            app.FnGlminEditField.Editable = 'off';
            app.FnGlminEditField.Position = [581 129 65 22];

            % Create xOGinEditFieldLabel
            app.xOGinEditFieldLabel = uilabel(app.VariablePoolTab);
            app.xOGinEditFieldLabel.Position = [683 149 54 22];
            app.xOGinEditFieldLabel.Text = 'xOGin [-]';

            % Create xOGinEditField
            app.xOGinEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.xOGinEditField.ValueDisplayFormat = '%.4f';
            app.xOGinEditField.Editable = 'off';
            app.xOGinEditField.Position = [681 129 65 22];

            % Create FHEditFieldLabel
            app.FHEditFieldLabel = uilabel(app.VariablePoolTab);
            app.FHEditFieldLabel.Position = [681 449 46 22];
            app.FHEditFieldLabel.Text = 'FH [%] ';

            % Create FHEditField
            app.FHEditField = uieditfield(app.VariablePoolTab, 'numeric');
            app.FHEditField.Limits = [0 100];
            app.FHEditField.ValueDisplayFormat = '%.3f';
            app.FHEditField.Editable = 'off';
            app.FHEditField.Position = [681 429 65 22];

            % Create AppOperationTab
            app.AppOperationTab = uitab(app.TabGroup);
            app.AppOperationTab.Title = 'App Operation';
            app.AppOperationTab.BackgroundColor = [1 1 1];

            % Create ReadInStartingVariablesButton
            app.ReadInStartingVariablesButton = uibutton(app.AppOperationTab, 'push');
            app.ReadInStartingVariablesButton.ButtonPushedFcn = createCallbackFcn(app, @ReadInStartingVariablesButtonPushed, true);
            app.ReadInStartingVariablesButton.Tooltip = {'Read in new starting variables from an excel or text file'};
            app.ReadInStartingVariablesButton.Position = [197 126 155 23];
            app.ReadInStartingVariablesButton.Text = 'Read-In Starting Variables';

            % Create EditStartingVariablesButton
            app.EditStartingVariablesButton = uibutton(app.AppOperationTab, 'push');
            app.EditStartingVariablesButton.ButtonPushedFcn = createCallbackFcn(app, @EditStartingVariablesButtonPushed, true);
            app.EditStartingVariablesButton.Tooltip = {'Edit or display the currently loaded starting variables'};
            app.EditStartingVariablesButton.Position = [197 166 155 23];
            app.EditStartingVariablesButton.Text = 'Edit Starting Variables';

            % Create SaveStartingVariablesButton
            app.SaveStartingVariablesButton = uibutton(app.AppOperationTab, 'push');
            app.SaveStartingVariablesButton.ButtonPushedFcn = createCallbackFcn(app, @SaveStartingVariablesButtonPushed, true);
            app.SaveStartingVariablesButton.Tooltip = {'Save currently selected starting variables to an excel or text file'};
            app.SaveStartingVariablesButton.Position = [197 88 155 23];
            app.SaveStartingVariablesButton.Text = 'Save Starting Variables';

            % Create DataStorageStartButton
            app.DataStorageStartButton = uibutton(app.AppOperationTab, 'push');
            app.DataStorageStartButton.ButtonPushedFcn = createCallbackFcn(app, @DataStorageStartButtonPushed, true);
            app.DataStorageStartButton.Position = [195 381 155 35];
            app.DataStorageStartButton.Text = 'Data Storage Start';

```

```
            % Create StartSimulationButton
            app.StartSimulationButton = uibutton(app.AppOperationTab, 'push');
            app.StartSimulationButton.ButtonPushedFcn = createCallbackFcn(app, @StartSimulationButtonPushed, true);
            app.StartSimulationButton.FontSize = 18;
            app.StartSimulationButton.FontWeight = 'bold';
            app.StartSimulationButton.Tooltip = {'Start a new simulation with the current starting variables'};
            app.StartSimulationButton.Position = [521 356 155 60];
            app.StartSimulationButton.Text = 'Start Simulation';

            % Create deltathEditFieldLabel
            app.deltathEditFieldLabel = uilabel(app.AppOperationTab);
            app.deltathEditFieldLabel.HorizontalAlignment = 'right';
            app.deltathEditFieldLabel.Position = [521 269 55 22];
            app.deltathEditFieldLabel.Text = 'delta t [h]';

            % Create deltathEditField
            app.deltathEditField = uieditfield(app.AppOperationTab, 'numeric');
            app.deltathEditField.Limits = [0 0.0005];
            app.deltathEditField.ValueChangedFcn = createCallbackFcn(app, @deltathEditFieldValueChanged, true);
            app.deltathEditField.Position = [595 269 81 22];

            % Create SelectedFileEditFieldLabel
            app.SelectedFileEditFieldLabel = uilabel(app.AppOperationTab);
            app.SelectedFileEditFieldLabel.HorizontalAlignment = 'right';
            app.SelectedFileEditFieldLabel.Position = [41 17 75 22];
            app.SelectedFileEditFieldLabel.Text = 'Selected File';

            % Create SelectedFileEditField
            app.SelectedFileEditField = uieditfield(app.AppOperationTab, 'text');
            app.SelectedFileEditField.Editable = 'off';
            app.SelectedFileEditField.Tooltip = {'The last loaded file is displayed here.'};
            app.SelectedFileEditField.Position = [131 17 397 22];

            % Create InoculationatStartCheckBox
            app.InoculationatStartCheckBox = uicheckbox(app.AppOperationTab);
            app.InoculationatStartCheckBox.ValueChangedFcn = createCallbackFcn(app, @InoculationatStartCheckBoxValueChanged,
                true);
            app.InoculationatStartCheckBox.Tooltip = {'Check this box to start the simulation with an already inoculated
                fermenter. If this box is not checked, the fermenter can be inoculated later in the simulation by toggling
                the "Inoculation" switch next to the plot window.'};
            app.InoculationatStartCheckBox.Text = 'Inoculation at Start';
            app.InoculationatStartCheckBox.Position = [523 295 122 60];

            % Create LoadProcessButton
            app.LoadProcessButton = uibutton(app.AppOperationTab, 'push');
            app.LoadProcessButton.ButtonPushedFcn = createCallbackFcn(app, @LoadProcessButtonPushed, true);
            app.LoadProcessButton.FontSize = 14;
            app.LoadProcessButton.FontWeight = 'bold';
            app.LoadProcessButton.Tooltip = {'Load a process from a MATLAB file'};
            app.LoadProcessButton.Position = [519 169 155 26];
            app.LoadProcessButton.Text = 'Load Process';

            % Create PlotPreviousDataButton
            app.PlotPreviousDataButton = uibutton(app.AppOperationTab, 'push');
            app.PlotPreviousDataButton.ButtonPushedFcn = createCallbackFcn(app, @PlotPreviousDataButtonPushed, true);
            app.PlotPreviousDataButton.FontSize = 14;
            app.PlotPreviousDataButton.FontWeight = 'bold';
            app.PlotPreviousDataButton.Tooltip = {'Plot generated data of a previous simulation from an excel or text file'};
            app.PlotPreviousDataButton.Position = [519 123 155 26];
            app.PlotPreviousDataButton.Text = 'Plot Previous Data';

            % Create DataStorageatStartCheckBox
            app.DataStorageatStartCheckBox = uicheckbox(app.AppOperationTab);
            app.DataStorageatStartCheckBox.ValueChangedFcn = createCallbackFcn(app, @DataStorageatStartCheckBoxValueChanged,
                true);
            app.DataStorageatStartCheckBox.Tooltip = {'Check this box to start the data storage as soon as the simulation is
                launched. If this box is not checked, data storage can be started manually at any time by pressing the "Data
                 Storage Start" button.'};
            app.DataStorageatStartCheckBox.Text = 'Data Storage at Start';
            app.DataStorageatStartCheckBox.Position = [195 345 135 22];
            app.DataStorageatStartCheckBox.Value = true;

            % Create StartingVariableOptionsLabel
            app.StartingVariableOptionsLabel = uilabel(app.AppOperationTab);
            app.StartingVariableOptionsLabel.FontSize = 14;
            app.StartingVariableOptionsLabel.Position = [195 208 160 22];
            app.StartingVariableOptionsLabel.Text = 'Starting Variable Options';

            % Create PreviouslyRecordedDataLabel
            app.PreviouslyRecordedDataLabel = uilabel(app.AppOperationTab);
            app.PreviouslyRecordedDataLabel.FontSize = 14;
            app.PreviouslyRecordedDataLabel.Position = [519 208 168 22];
```

```
            app.PreviouslyRecordedDataLabel.Text = 'Previously Recorded Data';

            % Create DataStorageOptionsLabel
            app.DataStorageOptionsLabel = uilabel(app.AppOperationTab);
            app.DataStorageOptionsLabel.FontSize = 14;
            app.DataStorageOptionsLabel.Position = [195 428 140 22];
            app.DataStorageOptionsLabel.Text = 'Data Storage Options';

            % Create SimulationStartOptionsLabel
            app.SimulationStartOptionsLabel = uilabel(app.AppOperationTab);
            app.SimulationStartOptionsLabel.FontSize = 14;
            app.SimulationStartOptionsLabel.Position = [519 428 156 22];
            app.SimulationStartOptionsLabel.Text = 'Simulation Start Options';

            % Create AuthorInformationTab
            app.AuthorInformationTab = uitab(app.TabGroup);
            app.AuthorInformationTab.Title = 'Author Information';
            app.AuthorInformationTab.BackgroundColor = [1 1 1];

            % Create Label
            app.Label = uilabel(app.AuthorInformationTab);
            app.Label.WordWrap = 'on';
            app.Label.FontSize = 14;
            app.Label.Position = [72 129 366 272];
            app.Label.Text = {'=========================================='; ' Title:     Biofermentation Simulation App'; '
                 Author: Lena Sophia Kaletsch'; ' Date:    01.03.2023'; '=========================================='; 'This
                 work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this
                 license, visit http://creativecommons.org/licenses/by/4.0/. This application is based on the BIOSIM program
                 conceived by Prof. Dr.-Ing. R. Luttmann and was developed for the laboratory of Bioprocess Automation at the
                  University of Applied Sciences Hamburg.'};

            % Create ExitButton
            app.ExitButton = uibutton(app.UIFigure, 'push');
            app.ExitButton.ButtonPushedFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);
            app.ExitButton.Position = [758 25 100 36];
            app.ExitButton.Text = 'Exit';

            % Create ProcessTimehEditFieldLabel
            app.ProcessTimehEditFieldLabel = uilabel(app.UIFigure);
            app.ProcessTimehEditFieldLabel.HorizontalAlignment = 'right';
            app.ProcessTimehEditFieldLabel.Position = [21 35 95 22];
            app.ProcessTimehEditFieldLabel.Text = 'Process Time [h]';

            % Create ProcessTimehEditField
            app.ProcessTimehEditField = uieditfield(app.UIFigure, 'numeric');
            app.ProcessTimehEditField.ValueDisplayFormat = '%.3f';
            app.ProcessTimehEditField.Position = [131 35 100 22];

            % Create ProcessRunningLampLabel
            app.ProcessRunningLampLabel = uilabel(app.UIFigure);
            app.ProcessRunningLampLabel.HorizontalAlignment = 'right';
            app.ProcessRunningLampLabel.Position = [262 35 97 22];
            app.ProcessRunningLampLabel.Text = 'Process Running';

            % Create ProcessRunningLamp
            app.ProcessRunningLamp = uilamp(app.UIFigure);
            app.ProcessRunningLamp.Position = [374 39 12 12];
            app.ProcessRunningLamp.Color = [1 0 0];

            % Create DataStorageRunningLampLabel
            app.DataStorageRunningLampLabel = uilabel(app.UIFigure);
            app.DataStorageRunningLampLabel.HorizontalAlignment = 'right';
            app.DataStorageRunningLampLabel.Position = [430 34 124 22];
            app.DataStorageRunningLampLabel.Text = 'Data Storage Running';

            % Create DataStorageRunningLamp
            app.DataStorageRunningLamp = uilamp(app.UIFigure);
            app.DataStorageRunningLamp.Position = [570 38 12 12];
            app.DataStorageRunningLamp.Color = [1 0 0];

            % Create ExpertmodeButton
            app.ExpertmodeButton = uibutton(app.UIFigure, 'push');
            app.ExpertmodeButton.ButtonPushedFcn = createCallbackFcn(app, @ExpertmodeButtonPushed, true);
            app.ExpertmodeButton.Position = [626 33 100 23];
            app.ExpertmodeButton.Text = 'Expert mode';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
```

```
    methods (Access = public)

        % Construct app
        function app = ControlAppODE_LuttmannpHTEMPnewexported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The simulation app

```
classdef FigureAppODESystemLuttmannpHTEMPexported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        ExportPlotButton                matlab.ui.control.Button
        SaveProcessButton               matlab.ui.control.Button
        PlotConfigurationPanel          matlab.ui.container.Panel
        variablelimrightminEditField    matlab.ui.control.NumericEditField
        variablelimleftminEditField     matlab.ui.control.NumericEditField
        tlimminEditField                matlab.ui.control.NumericEditField
        MinLabel                        matlab.ui.control.Label
        MaxLabel                        matlab.ui.control.Label
        variablelimrightEditField       matlab.ui.control.NumericEditField
        variablelimrightEditFieldLabel  matlab.ui.control.Label
        variablelimleftEditField        matlab.ui.control.NumericEditField
        variablelimleftEditFieldLabel   matlab.ui.control.Label
        tlimEditField                   matlab.ui.control.NumericEditField
        tlimEditFieldLabel              matlab.ui.control.Label
        Tree                            matlab.ui.container.CheckBoxTree
        PlotOptionsNode                 matlab.ui.container.TreeNode
        cXLglNode                       matlab.ui.container.TreeNode
        cS1LglNode                      matlab.ui.container.TreeNode
        cS2LglNode                      matlab.ui.container.TreeNode
        cS3LglNode                      matlab.ui.container.TreeNode
        FRlhNode                        matlab.ui.container.TreeNode
        FHlhNode                        matlab.ui.container.TreeNode
        NStrpmNode                      matlab.ui.container.TreeNode
        FnGlminNode                     matlab.ui.container.TreeNode
        pHLNode                         matlab.ui.container.TreeNode
        thetaLCNode                     matlab.ui.container.TreeNode
        pO2Node                         matlab.ui.container.TreeNode
        xOGNode                         matlab.ui.container.TreeNode
        cOLglNode                       matlab.ui.container.TreeNode
        xOLNode                         matlab.ui.container.TreeNode
        hNode                           matlab.ui.container.TreeNode
        CAltotmollNode                  matlab.ui.container.TreeNode
        addedacidlNode                  matlab.ui.container.TreeNode
        addedbaselNode                  matlab.ui.container.TreeNode
        addedantifoamlNode              matlab.ui.container.TreeNode
        addedfeedlNode                  matlab.ui.container.TreeNode
        VLlNode                         matlab.ui.container.TreeNode
        xOGinNode                       matlab.ui.container.TreeNode
        pO2wNode                        matlab.ui.container.TreeNode
        TimeofinoculationhEditField     matlab.ui.control.NumericEditField
        TimeofinoculationhEditFieldLabel  matlab.ui.control.Label
        LimitationTypeLabel             matlab.ui.control.Label
        qXpX1hEditField                 matlab.ui.control.NumericEditField
        qXpX1hEditFieldLabel            matlab.ui.control.Label
        Lamp                            matlab.ui.control.Lamp
        InoculationSwitch               matlab.ui.control.Switch
        InoculationSwitchLabel          matlab.ui.control.Label
        TimeforOperationsEditField      matlab.ui.control.NumericEditField
        TimeforOperationsEditFieldLabel  matlab.ui.control.Label
        SpeedfactorSlider               matlab.ui.control.Slider
        SpeedfactorSliderLabel          matlab.ui.control.Label
        PauseButton                     matlab.ui.control.Button
        ExitButton                      matlab.ui.control.Button
        UIAxes                          matlab.ui.control.UIAxes
    end


    properties (Access = public)
        CallingApp % Define main app

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Balances of the liquid phase system
        cXL % Cell concentration in liquid at time t [g/l]
        cS1L % Glucose concentration in liquid at time t [g/l]
        cS2L % Glycerine concentration in liquid at time t [g/l]
        cOL % Oxygen concentration in liquid at time t [g/l]
        CPLtot % Product concentration in liquid phase at time t [mol/l]
        CB1Ltot % pH-buffer acid KH2PO4 at time t [mol/l]
        CB2Ltot % pH-buffer base (NH4)2HPO4 at time t [mol/l]
        CAlLtot % pH-base titration NH3 and nitrogen source at time t [mol/l]
        CAcLtot % pH-acid titration and C-source at time t [mol/l]
        CCLtot % Total CO2 at time t [mol/l]

        cCLmax0 % Maximum concentration of CO2 in the liquid phase at time t = 0 h [g/l]
```

```
        % Balances of the anti foam system
        hF % Relative foam height at time t [%]
        AAF % anti foam activity at time t [-]

        % Balances of the temperature system
        thetaD % Temperature of double jacket at time t [°C]
        thetaL % Temperature of liquid at time t [°C]

        % Balance for the measuring systems
        pO2 % Oxygen saturation of liquid phase at time t [%]
        xO2 % Molar concentration of oxygen in off gas at time t [-]
        xCO2 % Molar concentration of carbondioxide in off gas at time t [-]
        pH % pH value at time t [-]
        pHL % pH in liquid phase at time t [-]

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Quasistationary balances of the liquid system
        cS3L % Acetate concentration in liquid at time point at time t [g/l]

        % Quasistationary balances of the gas system
        xOG % Molar concentration of O2 in gas phase at time t [-]
        xOL % Molar concentration of O2 in liquid phase at time t [-]
        xCG % Molar concentration of CO2 in gas phase at time t [-]
        xOGin % Molar concentration of O2 at reactor inlet [-]

        % Quasistationary balances of the temperature system
        thetaC % Temperature of cooling system at time t [°C]
        thetaTh % Temperature of heating system at time t [°C]
        thetaTc % Temperature of cooling down system at time t [°C]
        thetaDJ = 32.0; % Temperature of double jacket [°C]

        phiTcC
        mHmax

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Parameters calculated based on set constants
        % Growth parameters
        mySm % Substrate maintenance growth rate [1/h]
        myOm % Oxygen maintenance growth rate [1/h]

        qS1pXmax % Maximum glucose uptake rate of cells at time t [1/h]
        qS2pXmax % Maximum glycerine uptake rate of cells at time t [1/h]
        qS3pXmax % Maximum acetate uptake rate of cells at time t [1/h]

        qOpXmax % Maximum oxygen uptake rate of cells at time t [1/h]

        qXpX % Growth rate of cells at time t [1/h]

        % Temperature system parameters
        kCT % Coefficient of heat transmission cooling system - exchange system [W/(m2K)]
        tauCTc % Time constant cooling system

        RT % Heating resistance [K/W]
        Qny % Maximum amount of evaporation heat [kJ]

        DTc % Dilution rate of temperature flux [1/h]
        tauTcC % Time constant cooling system [h]
        CHmax % Maximum volumetric heat capacity of heating system [Ws/K]
        kHTh % Coefficient of heat transmission [kJ/m^2]

        DD % Diution rate of temperature flux [1/h]
        CD % Volumetric heat capacity of double jacket content and share of wall [Wh/K]
        kDL % Heat transmission coefficient double  jacket - liquid phase [W/(m^2*K)]
        kDU % Heat transmission coefficient double jacket - environment [W/(m^2*K)]
        kLU % Heat transmission coefficient liquid phase - environment [W/m^2*K)]
        tauDU % Time constant double jacket - environment [h]
        tauDL % Time constant double jacket - liquid phase [h]
        tauD % Time constant of double jacket system [h]

        % pH system parameters: dimensionless dissociation constants
        ApH
        BpH
        CpH
        DpH
        EpH
        FpH
        GpH
        HpH
        IpH

        pG % Pressure [N*m/kg]
```

```
        deltapG % Pressure difference to normal pressure [N*m/kg]

        FnG % Aeration rate at time t [l/min]
        QO2max % Maximum oxygen supply rate at time t [g/(l*h)]
        QCO2max % Maximum carbondioxide supply rate at time t [g/(l*h)]
        kLa % Volume refered O2-transition coefficient at time t [1/h]
        OTRmax % Theoretical maximum oxygen transfer rate at time t [g/(l*h)]
        OTR % Actual oxygen transfer rate at time t [g/(l*h)]
        OURmax % Theoretical maximum oxygen uptake rate at time t [g/(l*h)]
        OUR % Actual oxygen uptake rate at time t [g/(l*h)]
        NSt % Stirrer speed at time t [1/min]
        RQ % Respiratory Quotient at time t [-]
        OURm % Oxygen uptake rate for maintenance purposes [g/(l*h)]

        cE      = 0; % Difference between temperature setpoint and measured temperature [°C]
        ce      = 0; % Normalized difference cE [%]

        cEpH    = 0; % Difference between pH setpoint and measured pH filtered through a time delay of first order

        cP_Part      % P part of temperature master controller
        cI_Part = 0; % I part of temperature master controller

        cE_agi = 0; % Difference between pO2 setpoint and measured pO2 [%]
        ce_agi = 0; % Normalized difference cE_agi [-]
        cP_agi      % P part of pO2-agitation master controller [-]
        cI_agi = 0; % I part of pO2-agitation master controller [-]
        cD_agi      % D part of pO2-agitation master controller [-]

        cE_feed = 0; % Difference between feed rate setpoint and current feed rate [l/h]
        ce_feed = 0; % Normalized difference cE_feed [-]
        cP_feed      % P part of feed rate controller [-]
        cI_feed = 0; % I part of feed rate controller [-]
        cD_feed      % D part of feed rate controller [-]
        yfeed   = 0; % Output of feed controller

        cE_aeration = 0; % Difference between pO2 setpoint and measured pO2 [%]
        ce_aeration = 0; % Normalized difference cE_aeration [-]
        cP_aeration      % P part of aeration controller [-]
        cI_aeration = 0; % I part of aeration controller [-]
        cD_aeration      % D part of aeration controller [-]

        cE_gasmix = 0; % Difference between pO2 setpoint and measured pO2 [%]
        ce_gasmix = 0; % Normalized difference cE_gasmix [-]
        cP_gasmix      % P part of pO2-gasmix controller [-]
        cI_gasmix = 0; % I part of pO2-gasmix controller [-]
        cD_gasmix      % D part of pO2-gasmix controller [-]

        cE_LW = 0; % Difference between liquid weight setpoint and actual liquid weight [kg]
        ce_LW = 0; % Normalized difference cE_LW [-]
        cI_LW = 0; % I part of liquid weight controller [-]

        pGw % Pressure Set-point [N*m/kg]
        pO2w % Setpoint for pO2 defined in the control app [-]

        inoculum = 0; % Flag for inoculation start
        inoc_occ = 0; % Flag that activates once inoculation has happened

        VL % Volume at time point [l]
        VT1 % Volume of acid reservoir at time point [l]
        VT2 % Volume of base reservoir at time point [l]
        Vfeed % Volume of added feed [l]
        Vacid % Volume of added acid [l]
        Vbase % Volume of added base [l]
        FR % Feed Rate at time point [l/h]
        FH % Relative harvest rate [% maximum pump rate]
        VR % Volume in Feed reservoir [l]
        t % Time Stamp for Calculation and Plot functions [h]
        ToI        = 0; % Time of inoculation [h]
        VLflag     = 0; % Flag that activates once VLmax was reached once
        VolumeFlag = 0; % Returns 1 if VLmax is reached

        Plot1 % Plot object for left plot
        Plot2 % Plot object for right plot

        timit = 1 % Changes rate of timer to control simulation speed

        % DEBUG
        HO2
        HCO2
        cOLmax
        cCL
        CTRmax
```

```
        cCLmax
        CTR
        StC
        xCL
        CHL
        xpHkm1
        QuotpH
        mdotC
        cOL100

        Timer % Timer object needed for the simulation
    end

    methods (Access = public)

        function writeDataFile(app,line) % Creates data file entries
            dataFileID = fopen(app.CallingApp.dfn+" "+app.CallingApp.pfnx+".txt",'a');
            fprintf(dataFileID,"%s\n",line);
            fclose(dataFileID);
        end

        function writeParameterFile(app,line) % Creates parameter file entries
            parameterFileID = fopen(app.CallingApp.pfn+" "+app.CallingApp.pfnx+".txt",'a');
            fprintf(parameterFileID,"%s\n",line);
            fclose(parameterFileID);
        end

        function cycle_func(app)

            % Create timer object
            app.Timer = timer(...
                'ExecutionMode', 'fixedRate', ...                          % Run timer repeatedly
                'Period', 3600*app.CallingApp.deltat*app.timit^(-1), ...   % Period is dependent on delta t
                'BusyMode', 'queue',...                                    % Queue timer callbacks when busy
                'TimerFcn', @app.TimerFcn);                                % Specify callback function

            if app.CallingApp.SnapShot == 0

                % Set signals for stirrer, feed and aeration
                if app.CallingApp.NStwrpmEditField.Value > 0
                    app.CallingApp.motor = 1;
                else
                    app.CallingApp.motor = 0;
                end
                if app.CallingApp.FRwlhEditField.Value > 0
                    app.CallingApp.feed = 1;
                else
                    app.CallingApp.feed = 0;
                end
                if app.CallingApp.FnAIRwlminEditField.Value > 0
                    app.CallingApp.air = 1;
                else
                    app.CallingApp.air = 0;
                end
                if app.CallingApp.FnO2wlminEditField.Value > 0
                    app.CallingApp.O2 = 1;
                else
                    app.CallingApp.O2 = 0;
                end
                if app.CallingApp.FnAIRwlminEditField.Value > 0 || app.CallingApp.FnO2wlminEditField.Value > 0
                    app.CallingApp.aeration = 1;
                else
                    app.CallingApp.aeration = 0;
                end

                if app.CallingApp.InocStart == 1
                    app.inoc_occ = 1;
                    app.inoculum = 1;
                    app.InoculationSwitch.Enable = 0;
                    app.InoculationSwitch.Value  = 1;
                    app.TimeofinoculationhEditField.Value = 0.0;
                end

                % Define initial values of variables
                app.t(1)        = 0;
                if app.CallingApp.feed == 1
                    app.FR(1)   = app.CallingApp.FRw;
                else
                    app.FR(1)   = 0;
                end
                app.VL(1)       = app.CallingApp.VL0;
                app.VR(1)       = app.CallingApp.VR0;
```

```
                app.VT1(1)      = app.CallingApp.VT10;
                app.VT2(1)      = app.CallingApp.VT20;
                app.Vfeed(1)    = 0;
                app.Vacid(1)    = 0;
                app.Vbase(1)    = 0;

                if app.CallingApp.InocStart == 0
                    app.cXL(1) = 0;
                else
                    app.cXL(1) = app.CallingApp.cXL0;
                end
                app.cS1L(1)     = app.CallingApp.cS1L0;
                app.cS2L(1)     = app.CallingApp.cS2L0;
                app.cS3L(1)     = app.CallingApp.cS3L0;

                app.thetaL(1)   = app.CallingApp.thetaL0;
                app.thetaD(1)   = app.CallingApp.thetaD0;

                app.cOL(1)      = 1.1347*10^(-2);
                app.pO2(1)      = 100;
                app.pO2w(1)     = app.CallingApp.pO2w;
                app.xO2(1)      = app.CallingApp.xOAIR*100;
                app.xOL(1)      = 1.1347*10^(-2)/app.CallingApp.cOLmax;
                app.xCO2(1)     = app.CallingApp.xCAIR*100;
                app.xOG(1)      = 0.2094;
                app.xOGin(1)    = 0.2094;
                app.xCG(1)      = 0;

                app.CPLtot(1)   = 0;

                app.CB1Ltot(1) = app.CallingApp.CB1Ltot0;
                app.CB2Ltot(1) = app.CallingApp.CB2Ltot0;
                app.CAlLtot(1) = app.CallingApp.CAlLtot0;
                app.CAcLtot(1) = app.CallingApp.CAcLtot0;

                app.hF(1)       = 0;
                app.AAF(1)      = 0;

                app.pH(1)       = app.CallingApp.pH0;
                app.pHL(1)      = app.CallingApp.pH0;

                app.pG(1)       = app.CallingApp.pGcal+app.CallingApp.deltapGw*10^5;
                app.RQ(1)       = 0.1;

                app.CCLtot(1)   = (1+app.CallingApp.KC1*10^app.CallingApp.pH0+app.CallingApp.KC1*app.CallingApp.KC2*10^(2*app.
                     CallingApp.pH0))*app.CallingApp.xCAIR*app.cCLmax0/app.CallingApp.MCO2;

                if app.CallingApp.motor == 1
                    app.NSt(1) = app.CallingApp.NStw;
                else
                    app.NSt(1) = 0;
                end
                if app.CallingApp.aeration == 1
                    if app.CallingApp.air == 1
                        AIR = app.CallingApp.FnAIRw;
                    else
                        AIR = 0;
                    end

                    if app.CallingApp.O2 == 1
                        O2 = app.CallingApp.FnO2w;
                    else
                        O2 = 0;
                    end

                    if app.CallingApp.N2 == 1
                        N2 = app.CallingApp.FnN2w;
                    else
                        N2 = 0;
                    end

                    if app.CallingApp.CO2 == 1
                        CO2 = app.CallingApp.FnCO2w;
                    else
                        CO2 = 0;
                    end
                    app.FnG(1) = AIR+O2+N2+CO2;
                else
                    app.FnG(1)  = 0;
                end
                if app.CallingApp.harvest == 1
                    app.FH(1)   = app.CallingApp.FHrelw/100*app.CallingApp.FHmax;
```

```
                else
                    app.FH(1)  = 0;
                end
                app.OURm(1)    = 0;
                app.OURmax(1)  = 0;
                app.QO2max(1)  = app.FnG(1)*60*app.CallingApp.MO2/(app.VL(1)*app.CallingApp.VnM);
                app.QCO2max(1) = app.QO2max(1)*app.CallingApp.MCO2/app.CallingApp.MO2;
                app.kLa(1)     = app.CallingApp.kLamin+app.CallingApp.kLamax*((app.CallingApp.FnGw/app.CallingApp.FnGmax)^app.CallingApp.beta)*((app.CallingApp.NStw/app.CallingApp.NStmax)^(3*app.CallingApp.alpha))/((app.CallingApp.VL0/app.CallingApp.VLmin)^app.CallingApp.alpha);
                app.OTRmax(1)  = (app.CallingApp.kLamin+app.CallingApp.kLamax*((app.CallingApp.FnGw/app.CallingApp.FnGmax)^app.CallingApp.beta)*((app.CallingApp.NStw/app.CallingApp.NStmax)^(3*app.CallingApp.alpha))/((app.CallingApp.VL0/app.CallingApp.VLmin)^app.CallingApp.alpha))*app.CallingApp.cOLmax;
                app.OUR(1)     = 0;
                app.OTR(1)     = app.OTRmax(end)*(app.CallingApp.xOAIR-app.cOL(end)/app.CallingApp.cOLmax);
                app.CTR(1)     = 0;
                app.qXpX(1)    = 0;

                start(app.Timer)

            else

                file = uigetfile('*.mat',"Process File Selection");
                if file
                   load(file,'props','values','lengths','enable','val');

                    for i = 1:length(props)
                        propName = props{i};
                        if sum(values(:,i)) ~= 0 || lengths(i) > 1
                            app.(propName)(1,1:lengths(i)) = values(1:lengths(i),i);
                        end
                        if isprop(app.(propName),'Enable')
                            app.(propName).Enable = enable{i};
                        end
                        if isprop(app.(propName),'Value')
                            app.(propName).Value = val(i);
                        end
                    end

                    app.CallingApp.NStw    = app.NSt(end);
                    app.CallingApp.FRw     = app.FR(end);
                    app.CallingApp.FHrelw  = app.FH(end);
                    app.CallingApp.FnAIRw   = app.FnG(end);

                    app.PauseButton.Text = "Resume";

                    app.CallingApp.NStwrpmEditField.Value    = app.NSt(end);
                    app.CallingApp.FRwlhEditField.Value      = app.FR(end);
                    app.CallingApp.FHwEditField.Value        = app.FH(end)*100/app.CallingApp.FHmax;
                    app.CallingApp.FnAIRwlminEditField.Value = app.FnG(end);

                    app.SpeedfactorSlider.Value = 1;
                    app.timit              = 1;

                    % Display file name in control app and in figure app
                    % title
                    app.CallingApp.SelectedFileEditField.Value = file;
                    app.UIAxes.Title.String                    = erase(file,".mat");

                end

            end

        end

        function TimerFcn(app,varargin)

            % Cyclically call calculation and plot function via timer
            calculation_func(app)

            plot_func(app)

            % Set title of plot to filename if one was loaded
            if app.CallingApp.SelectedFile ~= "none" && app.CallingApp.SnapShot == 0
                app.UIAxes.Title.String   = app.CallingApp.FileName;
            end

        end

        function calculation_func(app)

            if app.t(end) <= app.CallingApp.tmax
```

```

            tic

            % Calculation of quasi stationary SPC-actual values
            if app.CallingApp.pO2_agi == 1

                % Calculate deviation from setpoint
                cEagi              = app.CallingApp.pO2w-app.pO2(end); % Error/Controller difference between measured pO2 and Setpoint e = (w-x)
                app.cE_agi         = (cEagi+0.001/app.CallingApp.deltat*app.cE_agi)/(0.001/app.CallingApp.deltat+1); % Error filtered through a time delay of first order with delta t and T = 0.001
                app.ce_agi(end+1) = app.cE_agi/(100-5); % Normalized controller difference e = (w-x)/(w_max-w_min); [-1,+1]

                % Calculate controller gains
                app.cP_agi         = app.ce_agi(end)*app.CallingApp.KP_agi; % P-part of controller with KP = 10.0
                app.cI_agi(end+1) = (app.ce_agi(end)+app.ce_agi(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_agi; % I-part of controller with KI = 1000 and with time increment deltat
                app.cD_agi         = (app.ce_agi(end)-app.ce_agi(end-1))/app.CallingApp.deltat*app.CallingApp.KD_agi; % D-part of controller with KD = 0.015 and with time increment deltat PARAMETER ANGEPASST VON 0.25, DA SONST SCHWINGUNG ZU GROß

                % Calculation of new relative setpoint for agitation
                % speed
                yNSt = app.cP_agi+app.cI_agi(end)+app.cD_agi; % PID sum

                % Define limits for new setpoint
                if yNSt < 0.3
                    yNSt = 0.3;
                elseif yNSt > 1
                    yNSt = 1;
                end

                % Calculation of new agitation speed setpoint [1/min]
                app.NSt(end+1) = yNSt*app.CallingApp.NStmax;

            elseif app.CallingApp.pO2_feed == 1

                % Calculate deviation from setpoint
                cEfeed              = app.CallingApp.pO2w-app.pO2(end); % Error/Controller difference between measured pO2 and Setpoint e = (w-x)
                app.cE_feed         = (cEfeed+0.001/app.CallingApp.deltat*app.cE_feed)/(0.001/app.CallingApp.deltat+1); % Filtered error through a PT1 delay with T = 0.001 and deltat
                app.ce_feed(end+1) = app.cE_feed/(100-0); % Normalized controller difference e = (w-x)/(w_max - w_min); [-1,+1]

                % Calculate controller outputs
                app.cP_feed = app.ce_feed(end)*app.CallingApp.KP_feed; % P-part of controller with KP = -2.0
                app.cI_feed = app.cI_feed+(app.ce_feed(end)+app.ce_feed(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_feed; % I-part of controller with KI = -15.0 and deltat
                app.cD_feed = (app.ce_feed(end)-app.ce_feed(end-1))/app.CallingApp.deltat*app.CallingApp.KD_feed; % D-part of controller with KD = -0.009

                % Calculation of new relative setpoint for feed pump
                % PID sum in percent
                app.yfeed  = ((app.cP_feed+app.cI_feed+app.cD_feed)*100+app.yfeed)/2;

                % Define limits for new setpoint
                if app.yfeed < 0
                    app.yfeed = 0;
                elseif app.yfeed > 100
                    app.yfeed = 100;
                end

                % Calculation of new agitation speed setpoint [1/h]
                app.FR(end+1) = app.yfeed/100*app.CallingApp.FRmax;

            elseif app.CallingApp.pO2_aeration == 1

                % Calculate deviation from setpoint
                cEaeration              = app.CallingApp.pO2w-app.pO2(end); % Error/Controller difference between measured pO2 and Setpoint e = (w-x)
                app.cE_aeration         = (cEaeration+0.001/app.CallingApp.deltat*app.cE_aeration)/(0.001/app.CallingApp.deltat+1); % Filtered error through a PT1 delay with T = 0.001 and deltat
                app.ce_aeration(end+1) = app.cE_aeration(end)/(100-5); % Normalized controller difference e = (w-x)/(w_max - w_min); [-1,+1]

                % Calculate controller gains
                app.cP_aeration = app.ce_aeration(end)*app.CallingApp.KP_aeration; % P-part of controller with KP = 20.0
                app.cI_aeration = app.cI_aeration+(app.ce_aeration(end)+app.ce_aeration(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_aeration; % I-part of controller with KI = 0.003 with time increment 0.005 h
```

```
                app.cD_aeration = (app.ce_aeration(end)-app.ce_aeration(end-1))/app.CallingApp.deltat*app.CallingApp.KD_aeration; % D-part of controller with KD = 0.0188

                % Calculation of new setpoint for aeration rate (add O2
                % aeration if AIR aeration is no longer sufficient)
                yaeration  = (app.cP_aeration+app.cI_aeration(end)+app.cD_aeration(end))*100; % PID sum in percent
                diff       = 0;

                % Define limits for new setpoint
                if yaeration < 30
                    yaeration = 30;
                elseif yaeration > 100
                    diff      = yaeration-100;
                    if diff > 100
                        diff = 100;
                    end
                    yaeration = 100;
                end

                % Calculation of setpoints for AIR and O2 aeration [l/h]
                AIR = yaeration/100*app.CallingApp.FnAIRmax;
                O2 = diff/100*app.CallingApp.FnO2max;

                % Calculate overall aeration rate [l/h]
                app.FnG(end+1) = AIR+O2;

            elseif app.CallingApp.pO2_gasmix == 1

                % Calculate deviation from setpoint
                cEgasmix             = app.CallingApp.pO2w-app.pO2(end); % Error/Controller difference between measured pO2 and Setpoint e = (w-x)
                app.cE_gasmix        = (cEgasmix+0.001/app.CallingApp.deltat*app.cE_gasmix)/(0.001/app.CallingApp.deltat+1); % Filtered error through a PT1 delay with T = 0.001 and deltat
                app.ce_gasmix(end+1) = app.cE_gasmix(end)/(1-app.CallingApp.xOAIR); % Normalized controller difference e = (w-x)/(w_max - w_min); [-1,+1]

                app.cP_gasmix        = app.ce_gasmix(end)*app.CallingApp.KP_gasmix; % P-part of controller with KP = 0.4
                app.cI_gasmix(end+1) = app.cI_gasmix(end)+(app.ce_gasmix(end)+app.ce_gasmix(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_gasmix; % I-part of controller with KI = 0.003 with time increment 0.005 h
                app.cD_gasmix        = (app.ce_gasmix(end)-app.ce_gasmix(end-1))/app.CallingApp.deltat*app.CallingApp.KD_gasmix; % D-part of controller with KD = 0.0005

                % Calculation of new relative setpoint for xOGin
                ygasmix  = (app.cP_gasmix+app.cI_gasmix(end)+app.cD_gasmix(end))*100; % PID sum in percent

                % Define limits for new setpoint
                if ygasmix < app.CallingApp.xOAIR*100
                    ygasmix = app.CallingApp.xOAIR*100;
                elseif ygasmix > 100
                    ygasmix = 100;
                end

                % Calculation of setpoint fir xOGin [-]
                xOGinw = ygasmix/100*1;
                app.CallingApp.xOGinwEditField.Value = xOGinw;

                % Calculation of AIR and O2 aeration rates [l/min]
                AIR    = (app.CallingApp.FnGw*(xOGinw-1))/(app.CallingApp.xOAIR-1);
                O2     = app.CallingApp.FnGw-AIR;

                % Calculation of overall aeration rate [l/min]
                app.FnG(end+1) = AIR+O2;

            end

            % Set stirrer speed if it is not pO2-controlled [1/min]
            if app.CallingApp.pO2_agi ~= 1
                if app.CallingApp.motor == 1
                    app.NSt(end+1) = app.CallingApp.NStw;
                else
                    app.NSt(end+1) = 0;
                end
            end

            % Calculation of feed rate with feed pump ON/OFF if it is not pO2-controlled [l/h]
            if app.CallingApp.pO2_feed ~= 1
                if app.CallingApp.feed == 1
                    app.FR(end+1) = app.CallingApp.FRw;
                else
                    app.FR(end+1) = 0;
                end
            end
```

```

% Display current values for feed rate and stirrer speed
app.CallingApp.FRlhEditField.Value   = app.FR(end);
app.CallingApp.NStrpmEditField.Value = app.NSt(end);

% Calculate aeration rate [l/min]
if app.CallingApp.pO2_aeration ~= 1 && app.CallingApp.pO2_gasmix ~= 1
    if app.CallingApp.aeration == 1
        if app.CallingApp.air == 1
            AIR = app.CallingApp.FnAIRw;
        else
            AIR = 0;
        end

        if app.CallingApp.O2 == 1
            O2 = app.CallingApp.FnO2w;
        else
            O2 = 0;
        end

        % Total unfiltered aeration rate
        app.FnG(end+1) = AIR+O2;
    else
        app.FnG(end+1)  = 0;
    end
end

% Display current value for aeration rate [l/min]
app.CallingApp.FnGlminEditField.Value = app.FnG(end);

if app.CallingApp.LW_harvest  == 1

    % Calculate difference to setpoint
    cELW             = app.VL(end)*app.CallingApp.rhoL-app.CallingApp.LWw; % Error/Controller difference between measured liquid weight and Setpoint e = (w-x)
    app.cE_LW        = (cELW+0.001/app.CallingApp.deltat*app.cE_LW)/(0.001/app.CallingApp.deltat+1); % Error filtered through a time delay of first order with deltat and T = 0.001 h
    app.ce_LW(end+1) = app.cE_LW/(app.CallingApp.VLmax*app.CallingApp.rhoL-app.CallingApp.VLmin*app.CallingApp.rhoL); % Normalized controller difference e = (w-x)/(w_max - w_min); [-1,+1]

    % Calculate controller gains
    cP_LW            = app.ce_LW(end)*app.CallingApp.KP_LW; % P-part of controller with KP = 10.0
    app.cI_LW(end+1) = app.cI_LW(end)+(app.ce_LW(end)+app.ce_LW(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_LW; % I-part of controller with KI = 0.3 and with time increment deltat
    cD_LW            = (app.ce_LW(end)-app.ce_LW(end-1))/app.CallingApp.deltat*app.CallingApp.KD_LW; % D-part of controller with KD = 1.0

    % Calculation of new relative setpoint for harvest pump
    yLW  = (cP_LW+app.cI_LW(end)+cD_LW)*100; % PID sum in percent

    % Define limits for new setpoint
    if yLW < 0
        yLW = 0;
    elseif yLW > 100
        yLW = 100;
    end

    % Calculation of new setpoint for harvest pump [l/h]
    app.FH(end+1) = yLW/100*app.CallingApp.FHmax;
else
    % Set harvest rate with harvest pump ON/OFF [l/h]
    if app.CallingApp.harvest == 1
        app.FH(end+1) = app.CallingApp.FHrelw/100*app.CallingApp.FHmax;
    else
        app.FH(end+1) = 0;
    end
end

% Display current value for harvest rate
app.CallingApp.FHEditField.Value = (app.FH(end)/app.CallingApp.FHmax)*100;

% Calculate molefraction at reactor inlet [-]
if app.FnG(end) > 0
    app.xOGin(end+1) = (app.CallingApp.xOAIR*AIR+O2)/app.FnG(end);
    xCGin            = (app.CallingApp.xCAIR*AIR)/app.FnG(end); % CO2 term was deleted as no aeration with CO2 will take place in this simulation
else
    app.xOGin(end+1) = 0;
    xCGin            = 0;
end

% Display current value for xOGin
```

```
                app.CallingApp.xOGinEditField.Value = app.xOGin(end);

                % Set tONi to time of inoculation, if it has already taken
                % place [h]
                if app.ToI ~= 0
                    tONi = app.ToI;
                end

                % Calculate anti foam addition activity
                if app.CallingApp.antifoam == 1 && app.ToI ~= 0
                    TON    = app.t(end)-tONi;
                    AAFin = app.CallingApp.AAFtast*app.CallingApp.VAF^(TON/app.CallingApp.Ttast);
                else
                    tONi  = app.t(end);
                    AAFin = 0;
                    TON    = 0;
                end

                if app.CallingApp.pH_mode == 1
                    % Calculation of titration rate with alkali pump ON/OFF [l/h]
                    if app.CallingApp.alkali == 1
                        FT2 = app.CallingApp.FT2max;
                    else
                        FT2 = 0;
                    end

                    % Calculation of titration rate with acid pump ON/OFF [l/h]
                    if app.CallingApp.acid == 1
                        FT1 = app.CallingApp.FT1max;
                    else
                        FT1 = 0;
                    end
                end

                if app.CallingApp.pH_mode == 0
                    % Calculation of pH difference to Setpoint and
                    % corresponding controller output as setpoint for
                    % acid/alkali pumps (Cascade Control)
                    if abs(app.CallingApp.pHw-app.pHL(end)) < 0.1
                        FT1 = 0;
                        FT2 = 0;
                    else

                        % Calculate difference to setpoint
                        epH       = app.CallingApp.pHw-app.pHL(end); % Error/Controller difference between pH in Liquid and
                             Setpoint e = (w-x)
                        app.cEpH  = (epH+0.001/app.CallingApp.deltat*app.cEpH)/(0.001/app.CallingApp.deltat+1); % Filtered
                             error through a PT1 delay with T = 0.001 h and deltat
                        cepH      = app.cEpH/(app.CallingApp.pHLmaxgr-app.CallingApp.pHLmingr); % Normalized controller
                             difference e = (w-x) / (w_max - w_min) ; [-1,+1]

                        % Calculate controller gains
                        cP_pH   = cepH*app.CallingApp.KP_pH; % P-part of controller with KP = 10000

                        % Calculation of new relative difference setpoint
                        % by master controller
                        % in percent
                        ypH     = cP_pH*100;
                        if ypH > 100
                            ypH = 100;
                        elseif ypH < -100
                            ypH = -100;
                        end

                        % Calculation of deviation from difference setpoint
                        cEypH   = app.CallingApp.ypH_SET-(ypH/100); % Error/Controller difference between measured pH
                             difference to setpoint and wanted difference
                        ceypH   = cEypH/(1-(-1)); % Normalized controller difference if the maximum ypH is 1 and the minimum
                             ypH is -1

                        % Calculation of new setpoint for acid/alkali pump
                        % by slave controller
                        % [l/h]
                        cP_ypH = ceypH*100; % P-part of controller in percent

                        % Calculation of T1 or T2 flow rate
                        % Define limits for new setpoint
                        if cP_ypH > 0
                            yT1 = cP_ypH*app.CallingApp.KP_pH2a;
                            if yT1 > 100
                                yT1 = 100;
                            end
```

```
            FT1 = yT1/100*app.CallingApp.FT2max;
            FT2 = 0;
        elseif cP_ypH < 0
            yT2 = cP_ypH*app.CallingApp.KP_pH2b;
            if yT2 > 100
                yT2 = 100;
            end
            FT1 = 0;
            FT2 = yT2/100*app.CallingApp.FT1max;
        else
            FT1 = 0;
            FT2 = 0;
        end
    end
end

% Display current values for FT1 and FT2
app.CallingApp.FT1acidlhEditField.Value = FT1;
app.CallingApp.FT2baselhEditField.Value = FT2;

if app.CallingApp.temp_mode == 1
    % Calculation of cooling power with cooling flux ON/OFF
    % [kg/h]
    if app.CallingApp.cooling == 1
        app.mdotC = app.CallingApp.mdotCmax;
    else
        app.mdotC = 0;
    end

    % Calculation of heating power with heating rod ON/OFF [W]
    if app.CallingApp.heating == 1
        PH = app.CallingApp.PHmax;
    else
        PH = 0;
    end
end

if app.CallingApp.temp_mode == 0
    % Calculation of temperature difference to Setpoint and corresponding controller parameters for temperature control at split range
    % A PT1 controller is used to filter the error signal and dampen its fluctuations
    e               = app.CallingApp.thetaLw-app.thetaL(end); % Error/Controller difference between Temperature in Liquid and Setpoint e = (w-x)
    app.cE          = (e+0.001/app.CallingApp.deltat*app.cE)/(0.001/app.CallingApp.deltat+1); % Filtered error through a PT1 delay with T = 0.001 h and deltat
    app.ce(end+1)   = app.cE/(app.CallingApp.thetaLmaxgr-app.CallingApp.thetaLmingr); % Normalized controller difference e = (w-x) / (w_max - w_min) ; [-1,+1]

    app.cP_Part         = app.ce(end)*app.CallingApp.KP_temp1; % P-part of controller with KP = 0.1
    app.cI_Part(end+1) = app.cI_Part(end)+(app.ce(end)+app.ce(end-1))/2*app.CallingApp.deltat*app.CallingApp.KI_temp1; % I-part of controller with KI = 0.01 with time increment deltat

    % Calculation of steam mass flux and cooling flux in case of steam heating at split-range
    wDJ  = app.cP_Part+app.cI_Part(end)+app.CallingApp.thetaDJ_WP; % PI sum + Workingpoint (ThetaDJw = ThetaLw)

    cDJ  = wDJ-app.thetaDJ; % Error/controller difference between temperature in double jacket and the calculated setpoint cDJ
    CDJ  = cDJ/(100-0); % Normalized controller difference assuming the double jacket should not be cooler than 0°C or hotter than 100°C

    yDJ = CDJ*100; % Master controller output in percent

    % Define limits for new setpoint
    if yDJ > 100
        yDJ = 100;
    elseif yDJ < -100
        yDJ = -100;
    end

    if yDJ > 0
        yH    = yDJ*app.CallingApp.KP_temp2h; % Slave controller output with KP = 10
        if yH > 100
            yH = 100;
        end
        mdotH = (yH/100)*app.CallingApp.mdotHmax;
        app.mdotC = 0;
    else
        yC    = -yDJ*app.CallingApp.KP_temp2c; % Slave controller output with KP = -10
        if yC < -100
            yC = -100;
        end
```

```
                        mdotH = 0;
                        app.mdotC = (yC/100)*app.CallingApp.mdotCmax*10;
                    end
                end

                % Eigenvalues of foam and anti foam deq. [1/h]
                lamdaF  = -app.AAF(end)/app.CallingApp.tauF0/(1+app.CallingApp.KFpX*app.cXL(end));
                lamdaAF = -app.cXL(end)/app.CallingApp.KAF;

                if app.VL(end) > 0
                    DR  = app.FR(end)./app.VL(end); % Refered feeding rate [1/h]
                    DT1 = FT1/app.VL(end); % Refered acid titration rate [1/h]
                    DT2 = FT2/app.VL(end); % Refered alkali titration rate [1/h]
                else
                    DR  = 0;
                    DT1 = 0;
                    DT2 = 0;
                end

                % Define ODE function for volume calculation
                ODE_Vol       = @(t,yV) ODE_Volume(app.FR(end),app.FH(end),FT1,FT2,t,yV);

                % Liquid volume balance
                [~,yV] = ode45(ODE_Vol,[0 app.CallingApp.deltat],[app.VL(end) app.VR(end) app.VT2(end) app.VT1(end)]);
                app.VL(end+1)  = yV(end,1);
                app.CallingApp.mLkgEditField.Value = app.VL(end)*app.CallingApp.rhoL;
                app.VR(end+1)  = yV(end,2);
                app.VT2(end+1) = yV(end,3);
                app.VT1(end+1) = yV(end,4);

                app.Vfeed(end+1) = app.CallingApp.VR0-app.VR(end);
                app.Vbase(end+1) = app.CallingApp.VT20-app.VT2(end);
                app.Vacid(end+1) = app.CallingApp.VT10-app.VT1(end);

                app.CallingApp.addedfeedlEditField.Value = app.Vfeed(end);
                app.CallingApp.addedbaselEditField.Value = app.Vbase(end);
                app.CallingApp.addedacidlEditField.Value = app.Vacid(end);
                app.CallingApp.VLlEditField.Value        = app.VL(end);

                % Refered dilution rate [1/h]
                Din = DR+DT1+DT2;

                % Temperature liquid phase [K]
                TL  = app.thetaL(end)+app.CallingApp.TnG;

                % Pressure in reactor measured in gas phase [N/m^2]
                if app.thetaL(end) < 100
                    app.pG(end+1) = app.pGw;
                    ExppDL        = 0;
                    pDL           = 0;
                else
                    ExppDL        = 10.9-2461/TL-2.065*log10(app.thetaL(end)/app.CallingApp.TnG);
                    % Steam pressure in liquid phase
                    pDL           = 9.8067*10^ExppDL;
                    app.pG(end+1) = pDL;
                end

                % Over pressure indication [bar]
                app.deltapG = (app.pG(end)-app.CallingApp.pnG)/10^5;


                % Quasistationary molar respiration quotient (offgas)
                RQ_Z = app.xCO2(end)/100*(1-app.xOGin(end))-xCGin*(1-app.xO2(end)/100);
                RQ_N = app.xOGin(end)*(1-app.xCO2(end)/100)-app.xO2(end)/100*(1-xCGin);
                if RQ_N ~= 0
                    app.RQ(end+1) = RQ_Z/RQ_N;
                else
                    app.RQ(end+1) = 1;
                end
                if app.RQ(end) <= 0
                    app.RQ(end) = 0.0001; % Avoid NaN error for C balance as it requires division by RQ
                end

                % to compare - RQ over metabolism
%                 RQ_int = app.CallingApp.yCpO*app.CallingApp.MO2/app.CallingApp.MCO2;

                % Iterative calculation of pH in liquid phase
                % Cations of the bufffer
                y1 = (app.CB1Ltot(end)+2*app.CB2Ltot(end))/app.CallingApp.CH0;

                % Anions of the buffer = total phosphoric acid
                y2 = (app.CB1Ltot(end)+app.CB2Ltot(end))/app.CallingApp.CH0;
```

```

% Dissolved-CO2
y3 = app.CCLtot(end)/app.CallingApp.CH0;

% Product acetate
y4 = app.CPLtot(end)/app.CallingApp.CH0;

% Titrated base
y5 = app.CAlLtot(end)/app.CallingApp.CH0;

% Titrated acid
y6 = app.CAcLtot(end)/app.CallingApp.CH0;

% Iterative solution
xpH    = 10^(7-app.pH(end));
app.QuotpH = 0.9;

% Iteration
for i = 1:100
    if (app.QuotpH <= 0.999 || app.QuotpH >= 1.001) && xpH >= 0 % If xpH < 0 NaN error occurs SET FLAG THIS WOULD HAVE HAPPENED
        app.xpHkm1 = xpH;
        fpH0     = xpH^2-1;
        fpH1     = xpH*y1;
        fpH2     = -(app.ApH*xpH^2+2*app.BpH*xpH+3*app.CpH)*xpH/(xpH^3+app.ApH*xpH^2+app.BpH*xpH+app.CpH)*y2;
        fpH3     = -(app.DpH+2*app.EpH/xpH)*y3;
        fpH4     = -app.FpH*xpH/(app.FpH+xpH)*y4;
        fpH5     = app.GpH*xpH^2/(1+app.GpH*xpH)*y5;
        fpH6     = -(app.HpH*xpH+2*app.IpH)*xpH/(xpH^2+app.HpH*xpH+app.IpH)*y6;
        fpH      = fpH0+fpH1+fpH2+fpH3+fpH4+fpH5+fpH6;
        fstrpH0 = 2*xpH;
        fstrpH1 = y1;
        fstrpH2 = -((app.ApH^2-2*app.BpH)*xpH^4+ ...
            (2*app.ApH*app.BpH-6*app.CpH)*xpH^3+ ...
            2*app.BpH^2*xpH^2+4*app.BpH*app.CpH*xpH+ ...
            3*app.CpH^2)/((xpH^3+app.ApH*xpH^2+app.BpH*xpH+app.CpH)^2)*y2;
        fstrpH3 = 2*app.EpH/(xpH*xpH)*y3;
        fstrpH4 = -app.FpH^2/((xpH+app.FpH)^2)*y4;
        fstrpH5 = app.GpH*xpH*(2+app.GpH*xpH)/((1+app.GpH*xpH)^2)*y5;
        fstrpH6 = -((app.HpH^2-2*app.IpH)*xpH^2+2*app.HpH*app.IpH*xpH+2*app.IpH^2)/((xpH^2+app.HpH*xpH+app.IpH)^2)*y6;
        fstrpH  = fstrpH0+fstrpH1+fstrpH2+fstrpH3+fstrpH4+fstrpH5+fstrpH6;
        xpH      = xpH-fpH/fstrpH;
        app.QuotpH  = app.xpHkm1/xpH;
    else
        break
    end
end

if xpH < 0
    xpH = -xpH; % Set flag that this occured
end

% pH value in liquid phase
app.pHL(end+1) = 7-log10(xpH);
app.CallingApp.pHLEditField.Value = app.pHL(end);

% Molar concentration of the H+ ions in the liquid phase
app.CHL = xpH*app.CallingApp.CH0;

% Influence of temperature and pH of the growth
if app.pHL(end) >= app.CallingApp.thetaLmingr && app.pHL(end) <= app.CallingApp.pHLmaxgr
    fpH    = (app.pHL(end)-app.CallingApp.pHLmingr)*(app.pHL(end)-app.CallingApp.pHLmaxgr)/((app.pHL(end)-app.CallingApp.pHLmingr)*(app.pHL(end)-app.CallingApp.pHLmaxgr)-(app.pHL(end)-app.CallingApp.pHLoptgr)^2);
else
    fpH    = 0;
end
if app.thetaL(end) >= app.CallingApp.thetaLmingr && app.thetaL(end) <= app.CallingApp.thetaLmaxgr
    ftheta = ((app.thetaL(end)-app.CallingApp.thetaLmaxgr)*(app.thetaL(end)-app.CallingApp.thetaLmingr)^2)/((app.CallingApp.thetaLoptgr-app.CallingApp.thetaLmingr)*((app.CallingApp.thetaLoptgr-app.CallingApp.thetaLmingr)*(app.thetaL(end)-app.CallingApp.thetaLoptgr)-(app.CallingApp.thetaLoptgr-app.CallingApp.thetaLmaxgr)*(app.CallingApp.thetaLoptgr+app.CallingApp.thetaLmingr-2*app.thetaL(end))));
else
    ftheta = 0;
end

% Maximum specific growth rate glucose [1/h]
my1max = app.CallingApp.my1opt*fpH*ftheta;

% Maximum specific growth rate glycerol [1/h]
my2max = app.CallingApp.my2opt*fpH*ftheta;
```

```
                % Maximum specific growth rate acetate [1/h]
                my3max = app.CallingApp.my3opt*fpH*ftheta;

                % Maximum specific glucose uptake rate [1/h]
                app.qS1pXmax = (my1max+app.mySm)/app.CallingApp.yXpS1gr;

                % Maximum specific glycerol uptake rate [1/h]
                app.qS2pXmax = (my2max+app.mySm)/app.CallingApp.yXpS2gr;

                % Maximum specific acetate uptake rate [1/h]
                app.qS3pXmax = (my3max+app.mySm)/app.CallingApp.yXpS3gr;

                % Maximum specific oxygen uptake rate [1/h]
                app.qOpXmax  = (my1max+app.mySm)/app.CallingApp.yXpOgr+app.CallingApp.qOpXm;

                % Calculation of O2 quantities
                % Maintenance O2 uptake rate [1/h]
                app.OURm(end+1)    = app.CallingApp.qOpXm*app.cXL(end);

                % Maximum O2 uptake rate [1/h]
                app.OURmax(end+1)  = app.qOpXmax*app.cXL(end);

                % Calculation of O2 Henry constant [Nm/kg]
                app.HO2 = app.CallingApp.HnO2/(1+app.CallingApp.K1HO2*app.thetaL(end)+app.CallingApp.K2HO2*app.thetaL(end)^2+app.CallingApp.K3HO2*app.thetaL(end)^3+app.CallingApp.K4HO2*app.thetaL(end)^4);

                % O2-concentration in liquid phase at 100 % pO2-indication
                app.cOL100 = app.CallingApp.pGcal*app.CallingApp.xOGcal/app.HO2;

                % Maximum potential O2 concentration in liquid phase [g/l]
                app.cOLmax = app.pG(end)/app.HO2;

                % Maximum oxygen supply rate [g/(l*h)]
                if app.VL(end) > 0
                    app.QO2max(end+1) = app.FnG(end)*60*app.CallingApp.MO2/(app.CallingApp.VnM*app.VL(end));
                else
                    app.QO2max(end+1) = 0;
                end

                % Maximum CO2 supply rate [g/(l*h)]
                app.QCO2max(end+1) = app.QO2max(end)*app.CallingApp.MCO2/app.CallingApp.MO2;

                % Calculation of viscosity [Ns/m^2]
                eta = app.CallingApp.eta0*(app.CallingApp.eta1/app.CallingApp.eta0)^(app.cXL(end)/app.CallingApp.cXLeta);

                % Volume refered O2-transition coefficient kLa [1/h]
                VLwert         = (app.VL(end)/app.CallingApp.VLmin)^app.CallingApp.alpha;
                NStwert        = (app.NSt(end)/app.CallingApp.NStmax)^(3*app.CallingApp.alpha);
                FGwert         = (app.FnG(end)/app.CallingApp.FnGmax)^app.CallingApp.beta;
                etawert        = (eta/app.CallingApp.eta0)^app.CallingApp.gamma;

                app.kLa(end+1) = app.CallingApp.kLamin+app.CallingApp.kLamax*FGwert*NStwert/VLwert*etawert*(1-app.AAF(end));

                % Theoretical maximum O2 transfer rate [g/(l*h)]
                app.OTRmax(end+1)  = app.kLa(end)*app.cOLmax;

                % Calculation of Stanton coefficient of oxygen
                if app.QO2max(end) > 0
                    StO = app.OTRmax(end)/app.QO2max(end);
                else
                    StO = 10^20;
                end

                % Optimum O2 limiting transport rate
%                 OTRopt = app.OTRmax(end)/(1+StO);

                % O2-pulp quantity in reactor liquid phase
                app.xOL(end+1) = app.cOL(end)/app.cOLmax;

                % OTR [g/(l*h)]
                app.OTR(end+1) = app.OTRmax(end)*(app.xOGin(end)-app.xOL(end))*2/((1+StO-(1-app.RQ(end))*StO*app.xOL(end))+sqrt((1+StO-(1-app.RQ(end))*StO*app.xOL(end))^2-4*(1-app.RQ(end))*StO*(app.xOGin(end)-app.xOL(end))));
                if app.OTR(end) < 0
                    app.OTR(end) = 0;
                end

                % Calculation of quasistationary O2 gas phase mole fraction [-]
                if app.OTR(end) ~= 0
                    app.xOG(end+1) = (app.QO2max(end)*app.xOGin(end)-app.OTR(end))/(app.QO2max(end)-(1-app.RQ(end))*app.OTR(end));
                else
```

```
        app.xOG(end+1) = app.xOGin(end);
    end

    % Calculation of CO2 Herny constant [Nm/kg]
    app.HCO2 = app.CallingApp.HnCO2/(1+app.CallingApp.K1HCO2*app.thetaL(end)+app.CallingApp.K2HCO2*app.thetaL(end)^2+app.CallingApp.K3HCO2*app.thetaL(end)^3+app.CallingApp.K4HCO2*app.thetaL(end)^4);

    % Maximum CO2 concentration [g/l]
    app.cCLmax = app.pG(end)/app.HCO2;

    % Dissolved CO2-concentration in liquid phase [g/l]
    app.cCL = (app.CHL^2*app.CallingApp.MCO2*app.CCLtot(end))/(app.CHL^2+app.CallingApp.KC1*app.CHL+app.CallingApp.KC1*app.CallingApp.KC2);

    % CO2-pulp quantity in reactor liquid phase [-]
    app.xCL = app.cCL/app.cCLmax;

    % Theoretical maximum CO2 transfer rate [g/(l*h)]
    app.CTRmax = app.CallingApp.deltaCpO*app.kLa(end)*app.cCLmax;

    % Stanton coefficient of CO2
    if app.QCO2max(end) > 0
        app.StC = app.CTRmax/app.QCO2max(end);
    else
        app.StC = 10^20;
    end

    % CO2 transfer rate
    app.CTR(end+1) = app.CTRmax*(xCGin-app.xCL)*2/((1+app.StC-(1-1/app.RQ(end))*app.StC*app.xCL)+sqrt((1+app.StC-(1-1/app.RQ(end))*app.StC*app.xCL)^2-4*(1-1/app.RQ(end))*app.StC*(xCGin-app.xCL)));

    % Calculation of the quasistationary CO2 gas phase mole fraction
    app.xCG(end+1) = (app.QCO2max(end)*xCGin-app.CTR(end))/(app.QCO2max(end)-(1-1/app.RQ(end))*app.CTR(end));

    % Optimum specific substrate uptake rate (no O2 limitation) [1/h]
    if app.cS1L(end) <= 0
        qS1pXopt = 0;
    else
        qS1pXopt = app.qS1pXmax*app.cS1L(end)/(app.cS1L(end)+app.CallingApp.kS1);
    end

    % Optimum specific substrate uptake rate 2 [1/h]
    if app.cS2L(end) <= 0
        qS2pXopt = 0;
    else
        qS2pXopt = app.qS2pXmax*app.cS2L(end)/(app.cS2L(end)+app.CallingApp.kS2)*app.CallingApp.kI21/(app.cS1L(end)+app.CallingApp.kI21);
    end

    % Share of substrate 3 of the product
    if app.CPLtot(end) <= 0
        app.cS3L(end+1) = 0;
    else
        app.cS3L(end+1) = app.CPLtot(end)*app.CallingApp.MP;
    end

    % Optimum specific substrate uptake rate 3 [1/h]
    if app.cS3L(end) <= 0
        qS3pXopt = 0;
    else
        qS3pXopt = app.qS3pXmax*app.cS3L(end)/(app.cS3L(end)+app.CallingApp.kS3)*app.CallingApp.kI31/(app.cS1L(end)+app.CallingApp.kI31)*app.CallingApp.kI32/(app.cS2L(end)+app.CallingApp.kI32);
    end

    % Specific cell growth rate at substrate limitation
    qXpXSgr = app.CallingApp.yXpS1gr*qS1pXopt+app.CallingApp.yXpS2gr*qS2pXopt+app.CallingApp.yXpS3gr*qS3pXopt;

    % Specific cell growth rate at O2 limitation
    if app.cOL(end) <= 0
        qXpXOgr = 0;
    else
        qXpXOgr = app.CallingApp.yXpOgr*(app.qOpXmax-app.CallingApp.qOpXm)*app.cOL(end)/(app.CallingApp.kO+app.cOL(end));
    end

    % Specific cell growth rate
    if qXpXSgr < qXpXOgr
        qXpXgr = qXpXSgr;
        if any(app.inoculum == 1)
            app.LimitationTypeLabel.Text = "Substrate Limitation";
        end
    else
```

```
                        qXpXgr = qXpXOgr;
                        if any(app.inoculum == 1)
                            app.LimitationTypeLabel.Text = "Oxygen Limitation";
                        end
                    end

                    % Calculation of cell specific reaction rates
                    % Specific cell reaction rate
                    if any(app.inoculum == 1)
                        app.qXpX(end+1) = qXpXgr-app.mySm;
                    else
                        app.qXpX(end+1) = 0;
                    end
                    app.qXpX1hEditField.Value = app.qXpX(end);

                    % Specific substrate uptake rate glucose [1/h]
                    SSG = qXpXgr/app.CallingApp.yXpS1gr;
                    if SSG < qS1pXopt
                        qS1pX = SSG;
                    else
                        qS1pX = qS1pXopt;
                    end

                    % Specific substrate uptake rate glycerine [1/h]
                    SSGl = (qXpXgr-app.CallingApp.yXpS1gr*qS1pX)/app.CallingApp.yXpS2gr;
                    if SSGl < qS2pXopt
                        qS2pX = SSGl;
                    else
                        qS2pX = qS2pXopt;
                    end

                    % Specific substrate uptake rate acetate [1/h]
                    SSA = (qXpXgr-app.CallingApp.yXpS1gr*qS1pX-app.CallingApp.yXpS2gr*qS2pX)/app.CallingApp.yXpS3gr;
                    if SSA < qS3pXopt
                        qS3pX = SSA;
                    else
                        qS3pX = qS3pXopt;
                    end

                    % Specific ammonia uptake rate [1/h]
                    qAlpX = app.CallingApp.yAlpXgr*qXpXgr;

                    % Specific uptake rate of titrated acid [1/h]
                    qAcpX = app.CallingApp.yAcpXgr*qXpXgr;

                    % Specific oxygen uptake rate [1/h]
                    qOpX = qXpXgr/app.CallingApp.yXpOgr+app.CallingApp.qOpXm;

                    % Resulting O2 yield coeficient
%                     yXpO = app.CallingApp.yXpOgr*app.qXpX(end)/(app.qXpX(end)+app.myOm);

                    % Volumetric oxygen uptake rate [g/(l*h)] and kLa [1/h]
                    if any(app.inoculum == 1)
                        app.OUR(end+1) = qOpX*app.cXL(end);
                    else
                        app.OUR(end+1) = 0;
                    end

                    % Volumetric oCO2 production rate [g/(l*h)]
                    CER = app.CallingApp.yCpO*app.OUR(end);

                    % Specific production rate, S-limited, O2-inhibited [1/h]
                    qPpX = (app.CallingApp.yPpS1*qS1pXopt+app.CallingApp.yPpS2*qS2pXopt)*app.CallingApp.kIPO/(app.CallingApp.kIPO+
                          app.cOL(end))-app.CallingApp.yPpS3*qS3pX;

                    % Calculation of ammonia transfer rate (volatile) [g/(l*h)]
                    AlTR = -app.CallingApp.KAlvol*app.FnG(end)*app.CAlLtot(end)*app.CallingApp.MAl/app.VL(end);

                    % Calculation of variables in the temperature systems
                    % Quasitationary calculation of the heat exchanger
                    % temperature
                    % Dilution rate of the steam flux [1/h]
                    if app.CallingApp.temp_mode == 0
                        DH = mdotH/app.mHmax;
                    end

                    % Time constant of the primary heating cycle [h]
                    tauHTh = app.mHmax*app.CallingApp.cH2O/(app.kHTh*app.CallingApp.AHTh);

                    % Dimensionless quantities
                    if app.CallingApp.temp_mode == 0
                        phiHTh = DH*tauHTh;
```

```
    phiHT   = mdotH/app.CallingApp.mdotT;
end

% Temperature of heating outlet
% Electrical heating [°C]
if app.CallingApp.temp_mode == 1
    if app.CallingApp.heating == 1
        app.thetaTh = app.thetaD(end)+app.RT*PH;
    else
        app.thetaTh = app.thetaL(end); % Added this line to account for thetaTh when the heating power is turned off
    end
end

% Steam heating as alternative for electrical heating
if app.CallingApp.temp_mode == 0
    app.thetaTh = ((1+phiHTh)*app.CHmax*app.thetaD(end)+phiHT*(app.CHmax*app.CallingApp.thetaHin+app.Qny(end)))/(app.CHmax*(1+phiHTh+phiHT));
end

% Dilution rate of cooling flux [1/h]
DC     = app.mdotC/app.CallingApp.mC;
phiCTc = DC*app.tauCTc;

% Cooling outlet temperature [°C]
app.thetaTc = (phiCTc*app.CallingApp.thetaCin+(1+phiCTc)*app.phiTcC*app.thetaTh)/((1+app.phiTcC)*(1+phiCTc)-1);

% Cooling water outlet temperature [°C]
app.thetaC  = (phiCTc*app.CallingApp.thetaCin+app.thetaTc)/(1+phiCTc);

% Cascade control quantity
app.thetaDJ = app.thetaTc;

% Volumetric heat capacity of the liquid ohase and share of
% the wall [Wh/K]
CL = app.CallingApp.rhoL*app.VL(end)*app.CallingApp.cH2O+app.CallingApp.mWL*app.CallingApp.cW;

% Time constant liquid phase - double jacket [h]
tauLD = CL/(app.kDL*app.CallingApp.ADL);

% Time constant liquid phase - environment [h]
tauLU = CL/(app.kLU*app.CallingApp.ALU);

% Time constnat reactor liquid phase [h]
tauL = 1/(1/tauLD+1/tauLU);

% Microbiological heat generation [W]
QdotM = app.CallingApp.KHM*app.VL(end)*app.OUR(end);

% Thermal efficiency of the stirrer
QdotSt = app.CallingApp.KHSt*app.VL(end)*app.NSt(end)^3;

% DEFINITION OF DYNAMIC MODEL EQUATIONS
% Cell mass balances and inoculation
ODE_concbal = @(t,y) ODE_Luttmann_complete(app.qXpX(end),Din,app.CallingApp.cS1R,qS1pX,app.CallingApp.cS2R,qS2pX,app.CallingApp.CPRtot,qPpX,app.CallingApp.MP,app.CallingApp.cOT1,app.CallingApp.cOT2,app.CallingApp.cOR,app.OTR(end),app.OUR(end),app.cOL100,app.CallingApp.TMpO2,DT1,app.CallingApp.CAcT1tot,qAcpX,app.CallingApp.MAc,DT2,app.CallingApp.CAlT2tot,qAlpX,AlTR,app.CallingApp.MAl,DR,app.CallingApp.CCRtot,app.CallingApp.CCT1tot,app.CallingApp.CCT2tot,app.CTR(end),CER,app.CallingApp.MCO2,app.pHL(end),app.CallingApp.TMpH,app.CallingApp.TMxO2,app.xOG(end),app.CallingApp.TMxCO2,app.xCG(end),lamdaF,app.CallingApp.qhpX,lamdaAF,AAFin,app.tauD,app.DD,app.thetaTc,app.tauDL,app.CallingApp.thetaU,app.tauDU,tauL,tauLD,tauLU,QdotM,QdotSt,CL,t,y);

if all(app.inoculum == 0) || any(app.inoc_occ == 1)
    [~,y] = ode45(ODE_concbal,[0 app.CallingApp.deltat],[app.cXL(end) app.cS1L(end) app.cS2L(end) app.CPLtot(end) app.cOL(end) app.pO2(end) app.CB1Ltot(end) app.CB2Ltot(end) app.CAcLtot(end) app.CAlLtot(end) app.CCLtot(end) app.pH(end) app.xO2(end) app.xCO2(end) app.hF(end) app.AAF(end) app.thetaD(end) app.thetaL(end)]);
    for i = 1:18
        if y(end,i) < 0
            y(end,i) = 0;
        end
    end
    app.cXL(end+1)     = y(end,1);
    app.cS1L(end+1)    = y(end,2);
    app.cS2L(end+1)    = y(end,3);
    app.CPLtot(end+1) = y(end,4);
    app.cOL(end+1)     = y(end,5);
    app.pO2(end+1)     = y(end,6);
    app.CB1Ltot(end+1) = y(end,7);
    app.CB2Ltot(end+1) = y(end,8);
```

```
            app.CAcLtot(end+1) = y(end,9);
            app.CAlLtot(end+1) = y(end,10);
            app.CCLtot(end+1)  = y(end,11);
            app.pH(end+1)      = y(end,12);
            app.xO2(end+1)     = y(end,13);
            app.xCO2(end+1)    = y(end,14);
            app.hF(end+1)      = y(end,15);
            app.AAF(end+1)     = y(end,16);
            app.thetaD(end+1)  = y(end,17);
            app.thetaL(end+1)  = y(end,18);
            if app.thetaL(end) > 100.0
                app.thetaL(end) = 100.0; % Set flag that this happened!
            end
        end

        if any(app.inoculum == 1) && all(app.inoc_occ == 0)
            app.cXL(end+1)                        = app.CallingApp.cXL0; % [g/h]
            app.ToI                               = app.t(end);
            app.TimeofinoculationhEditField.Value = app.ToI;

            [~,y] = ode45(ODE_concbal,[0 app.CallingApp.deltat],[app.cXL(end-1) app.cS1L(end) app.cS2L(end) app.CPLtot
                 (end) app.cOL(end) app.pO2(end) app.CB1Ltot(end) app.CB2Ltot(end) app.CAcLtot(end) app.CAlLtot(end)
                 app.CCLtot(end) app.pH(end) app.xO2(end) app.xCO2(end) app.hF(end) app.AAF(end) app.thetaD(end) app.
                 thetaL(end)]);
            for i = 2:18
                if y(end,i) < 0
                    y(end,i) = 0;
                end
            end
            app.cS1L(end+1)    = y(end,2);
            app.cS2L(end+1)    = y(end,3);
            app.CPLtot(end+1) = y(end,4);
            app.cOL(end+1)     = y(end,5);
            app.pO2(end+1)     = y(end,6);
            app.CB1Ltot(end+1) = y(end,7);
            app.CB2Ltot(end+1) = y(end,8);
            app.CAcLtot(end+1) = y(end,9);
            app.CAlLtot(end+1) = y(end,10);
            app.CCLtot(end+1)  = y(end,11);
            app.pH(end+1)      = y(end,12);
            app.xO2(end+1)     = y(end,13);
            app.xCO2(end+1)    = y(end,14);
            app.hF(end+1)      = y(end,15);
            app.AAF(end+1)     = y(end,16);
            app.thetaD(end+1)  = y(end,17);
            app.thetaL(end+1)  = y(end,18);
            if app.thetaL(end) > 100.0
                app.thetaL(end) = 100.0; % Set flag that this happened!
            end

            app.inoc_occ = 1;
        end

        app.CallingApp.thetaLCEditField.Value  = app.thetaL(end);
        app.CallingApp.pO2EditField.Value      = app.pO2(end);
        app.CallingApp.addedAFlEditField.Value = app.AAF(end);

        app.pO2w = zeros(size(app.pO2))+app.CallingApp.pO2w;

        % Set Volume Flag if VLmax was reached
        if app.VL(end) >= app.CallingApp.VLmax
            app.VolumeFlag = 1;
        end

        % Display error message if VLmax was reached and set VLFlag
        % to avoid displaying the message multiple times
        if any(app.VolumeFlag == 1) && any(app.VLflag == 0)

            app.VLflag = 1;
            stop(app.Timer);
            errordlg('VLmax was reached');
            app.PauseButton.Text = "Resume";

        end

        % Write calculated data into data file if the
        % command is set to on
        if app.CallingApp.mds == 1
            msg=sprintf("%s; %7.2f; %6.2f; %6.2f; %6.2f; %6.10f; %6.2f; %6.2f; %6.2f; %6.2f; %6.2f; %6.2f; %6.2f; %6.2
                 f; %6.2f; %6.2f; %6.2f; %6.2f;",datetime,app.t(end),app.cXL(end),app.cS1L(end),app.cS2L(end),app.
                 cS3L(end),app.pO2(end),app.thetaL(end),app.pHL(end),app.xOG(end),app.xCG(end),app.VL(end),app.NSt(
                 end),app.FnG(end),app.FR(end),app.deltapG(end),app.qXpX(end),app.CAlLtot(end));
```

```
                    writeDataFile(app,msg);
                end

                app.t(end+1) = app.t(end)+app.CallingApp.deltat;
                app.CallingApp.ProcessTimehEditField.Value = app.t(end);

                if ~isempty(app.CallingApp.Tree.CheckedNodes)
                    UITablefunc(app.CallingApp)
                end

                toc
                app.TimeforOperationsEditField.Value = toc;

            else

                stop(app.Timer);

            end
        end

        function plot_func(app)

                Mstr   = {'cXL' 'cS1L' 'cS2L' 'cS3L' 'FR' 'FH' 'NSt' 'FnG' 'pHL' 'thetaL' 'pO2' 'xOG' 'xOL' 'qXpX' 'CAlLtot' 'cOL' 'Vacid' 'Vbase' 'AAF' 'Vfeed' 'VL' 'xOGin' 'pO2w'};
                tabstr = {'cXL [g/l]' 'cS1L [g/l]' 'cS2L [g/l]' 'cS3L [g/l]' 'FR [1/h]' 'FH [1/h]' 'NSt [rpm]' 'FnG [1/min]' 'pHL [-]' 'thetaL [°C]' 'pO2 [%]' 'xOG [-]' 'xOL [-]' 'µ [1/h]' 'CAlLtot [mol/l]' 'cOL [g/l]' 'added acid [l]' 'added base [l]' 'added antifoam [l]' 'added feed [l]' 'VL [l]' 'xOGin [-]' 'pO2w [%]'};

                a = 1;

                % Eliminate all plottable variables that are currently not
                % loaded in the workspace (avoids error if a node referring
                % to an empty variable is selected)
                for i = 1:length(tabstr)
                    if isempty(app.(Mstr{i})) ~= 1
                        str{a} = tabstr{i};
                        M(a,:) = app.(Mstr{i});
                        a = a+1;
                    end
                end

                h = 1;

                % Assess the checked nodes and compare them to the actually
                % loaded variables as filtered previously. Save the
                % positions of the variables that are to be plotted
                CN    = app.Tree.CheckedNodes;
                for j = 1:length(CN)
                    data = CN(j).Text;
                    p     = find(strcmp(data,str));
                    if isempty(p) ~=1
                        index(h) = p;
                        h = h+1;
                    end
                end

                % If variables were selected, split them into two
                % categories (one for each y axis) and generate
                % corresponding axis labels
                if exist("data","var")
                    n = 1;
                    m = 1;

                        for i = 1:length(index)
                                if i < length(index)/2+1
                                    M1(n,:) = M(index(i),:);
                                    h1(n)    = str(index(i));
                                    n = n+1;
                                else
                                    M2(m,:) = M(index(i),:);
                                    h2(m)    = str(index(i));
                                    m = m+1;
                                end
                                headers(n+m-2) = str(index(i));
                        end

                        % Plot selected data against t
                        if exist("M2","var")
                            yyaxis(app.UIAxes,'right')
                            app.Plot1 = plot(app.UIAxes,app.t,M2,'Color','red');
                            lim = app.UIAxes.YLim;
                            app.variablelimrightminEditField.Value = lim(1);
```

```
                            app.variablelimrightEditField.Value = lim(2);
                            ylabel(app.UIAxes,h2)
                        end

                        if exist("M1","var")
                            yyaxis(app.UIAxes,'left')
                            app.Plot2 = plot(app.UIAxes,app.t,M1,'Color','black');
                            lim = app.UIAxes.YLim;
                            app.variablelimleftminEditField.Value = lim(1);
                            app.variablelimleftEditField.Value = lim(2);
                            ylabel(app.UIAxes,h1')
                            legend(app.UIAxes,headers,"Location","best")
                        end

                        lim = app.UIAxes.XLim;
                        app.tlimminEditField.Value = lim(1);
                        app.tlimEditField.Value = lim(2);
                end

        end

        function prev_plot(app)

            SelectedFile = uigetfile(app.CallingApp.filter);
            ReadData     = readcell(SelectedFile);

            app.UIAxes.Title.String = erase(convertCharsToStrings(SelectedFile),app.CallingApp.erasestr);
            app.CallingApp.SelectedFileEditField.Value = convertCharsToStrings(SelectedFile);

            % Separate Header from Data
            Header = ReadData(1,:);
            Data   = ReadData(2:length(ReadData),:);

            % Convert data type to string and then to numbers
            Data = string(Data);
            Data = str2double(Data);

            % Assign values given in loaded file
            for i = 1:length(Header)
                app.(Header{i})(1,:) = Data(:,i);
            end

            app.pG = app.deltapG+app.CallingApp.pnG;
            app.InoculationSwitch.Enable = "off";
            app.Lamp.Visible             = "off";

            plot_func(app)

        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Define Control App as mainapp
            app.CallingApp = mainapp;

            if app.CallingApp.ExpertMode == 0
                app.LimitationTypeLabel.Visible = "off";
            end

            app.InoculationSwitch.ItemsData = [0 1];

            % Calculate growth rates
            app.mySm = app.CallingApp.yXpS1gr*app.CallingApp.qS1pXm; % Specific substrate-maintenance-growth rate [1/h]
            app.myOm = app.CallingApp.yXpOgr*app.CallingApp.qOpXm+app.mySm; % Specific oxygen-maintenance-growth rate [1/h]

            app.qS1pXmax = (app.CallingApp.my1opt+app.mySm)/app.CallingApp.yXpS1gr; % Maximum specific glucose uptake rate [1/
                h]
            app.qS2pXmax = (app.CallingApp.my2opt+app.mySm)/app.CallingApp.yXpS2gr; % Maximum specific glycerol uptake rate
                [1/h]
            app.qS3pXmax = (app.CallingApp.my3opt+app.mySm)/app.CallingApp.yXpS3gr; % Maximum specific acetate uptake rate [1/
                h]

            app.qOpXmax  = (app.CallingApp.my1opt+app.mySm)/app.CallingApp.yXpOgr+app.CallingApp.qOpXm; % Maximum specific
                oxygen uptake rate [1/h] INFO: INSTEAD OF MY1MAX WHICH IS NOT DEFINED YET I USED MY1OPT (change later maybe)

            % Calculate parameters of temperature system
```

```
            app.kCT      = 1/(1/app.CallingApp.alphaC+app.CallingApp.deltaCT/app.CallingApp.lamdaCT+1/app.CallingApp.alphaT); % Heat passing coefficient cooling system - exchange system [W/(m^2*K)]
            app.tauCTc   = app.CallingApp.mC*app.CallingApp.cH2O/(app.kCT*app.CallingApp.ACT); % Time constant cooling system - cooling down system [h]
            app.RT       = 1/(app.CallingApp.mdotT*app.CallingApp.cH2O); % Heat resistance [K/W]
            app.DTc      = app.CallingApp.mdotT/app.CallingApp.mTc; % Flow through rate of temperature stream in cooling down system [1/h]
            app.tauTcC   = app.CallingApp.mTc*app.CallingApp.cH2O/(app.kCT*app.CallingApp.ACT); % Time constant cooling system [h]
            app.phiTcC   = app.DTc*app.tauTcC; % Time constant of cooling down system
            app.DD       = app.CallingApp.mdotT/app.CallingApp.mD; % Flow through rate of temperature stream in double jacket system [1/h]
            app.CD       = app.CallingApp.mD*app.CallingApp.cH2O+app.CallingApp.mWD*app.CallingApp.cW; % Volumetric heat capacity double jacket [Wh/K]
            app.kDU      = 1/(1/app.CallingApp.alphaD+app.CallingApp.deltaDU/app.CallingApp.lamdaDU+1/app.CallingApp.alphaU); % Heat transmission coefficient double jacket - environment [W/(m^2*K)]
            app.tauDU    = app.CD/(app.kDU*app.CallingApp.ADU); % Time constant double jacket - environment [h]
            app.kDL      = 1/(1/app.CallingApp.alphaD+app.CallingApp.deltaDL/app.CallingApp.lamdaDL+1/app.CallingApp.alphaL); % Heat transmissio  coefficient double jacket - liquid phase [W/(m^2*K)]
            app.tauDL    = app.CD/(app.kDL*app.CallingApp.ADL); % Time constant double jacket - liquid phase [h]
            app.tauD     = 1/(app.DD+1/app.tauDU+1/app.tauDL); % Time constant of double jacket system [h]
            app.kLU      = 1/(1/app.CallingApp.alphaL+app.CallingApp.deltaLU/app.CallingApp.lamdaLU+1/app.CallingApp.alphaU); % Heat transmission coefficient liquid - environment [W/(m^2*K)]
            app.mHmax    = app.CallingApp.VH*app.CallingApp.rhoH2O; % Maximum possible mass in heating system H [kg]
            app.CHmax    = app.mHmax*app.CallingApp.cH2O; % Maximum volumetricheat capacity of the heating system [Ws/K]
            app.kHTh     = 1/(1/app.CallingApp.alphaH+app.CallingApp.deltaHTh/app.CallingApp.lamdaHTh+1/app.CallingApp.alphaTh); % Coefficient of heat transmission [kJ/m^2]
            app.Qny      = app.CallingApp.deltahv*app.mHmax; % Maximum amount of heat of evaporation [kJ]

            % Absolute pressure set-point calculation [N/m^2]
            app.pGw = app.CallingApp.deltapGw*10^5+app.CallingApp.pnG;

            % Calculate parameters of pH system
            app.cCLmax0  = app.CallingApp.pGcal/app.CallingApp.HCO20;

            % Dimensionless dissociation constants
            app.ApH = app.CallingApp.KB1/app.CallingApp.CH0;
            app.BpH = app.ApH*app.CallingApp.KB2/app.CallingApp.CH0;
            app.CpH = app.BpH*app.CallingApp.KB3/app.CallingApp.CH0;
            app.DpH = app.CallingApp.KC1/app.CallingApp.CH0;
            app.EpH = app.DpH*app.CallingApp.KC2/app.CallingApp.CH0;
            app.FpH = app.CallingApp.KP/app.CallingApp.CH0;
            app.GpH = app.CallingApp.KAl/app.CallingApp.CH0;
            app.HpH = app.CallingApp.KAc1/app.CallingApp.CH0;
            app.IpH = app.HpH*app.CallingApp.KAc2/app.CallingApp.CH0;

            % Set inoculation lamp color to red if inoculation has not yet
            % occured
            if app.CallingApp.InocStart ~= 0
                app.Lamp.Color = "green";
            else
                app.Lamp.Color = "red";
            end

            xlim(app.UIAxes,[0 app.CallingApp.tmax])
            app.tlimEditField.Value = app.CallingApp.tmax;

            yyaxis(app.UIAxes,'left')
            lim = app.UIAxes.YLim;
            app.UIAxes.YColor = 'k';
            app.variablelimleftEditField.Value = lim(2);

            yyaxis(app.UIAxes,'right')
            lim = app.UIAxes.YLim;
            app.UIAxes.YColor = 'r';
            app.variablelimrightEditField.Value = lim(2);

            if app.CallingApp.mds == 1
                  % Define file names including the date and time of creation
                  app.CallingApp.dfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of data file
                  app.CallingApp.pfnx= strrep(datestr(datetime),':','-'); % Add timestamp to name of parameter file

                  % Write the chosen parameters to the parameter file
                  writeParameterFile(app,sprintf("%s",app.CallingApp.Version));
                  writeParameterFile(app," ");
                  writeParameterFile(app,"Chosen Parameters");
                  writeParameterFile(app,sprintf("cXL0          : %3.2f g",app.CallingApp.cXL0));
                  writeParameterFile(app,sprintf("cS1L0          : %3.2f g",app.CallingApp.cS1L0));
                  writeParameterFile(app,sprintf("FRw          : %3.2f 1/h",app.CallingApp.FRw));
                  writeParameterFile(app,sprintf("NStw          : %3.2f g/g",app.CallingApp.NStw));
                  writeParameterFile(app,sprintf("FnGw          : %3.2f 1/h",app.CallingApp.FnGw));
                  writeParameterFile(app,sprintf("VLmax          : %3.2f 1/h",app.CallingApp.VLmax));
```

```
                writeParameterFile(app,sprintf("cS1R          : %3.2f 1/h",app.CallingApp.cS1R));
                writeParameterFile(app,sprintf("VR0           : %3.2f 1/h",app.CallingApp.VR0));
                writeParameterFile(app,sprintf("tmax          : %3.2f 1/h",app.CallingApp.tmax));

                % Define headers for the file columns
                writeDataFile(app,"Time        ;           t [s];        cXL [g/l];        cS1L [g/l];        cS2L [g/l];        cS3L [g/l];        pO2 [%];        thetaL [°C];        pHL [-];        xOG [-];        xCG [-];        VL [l];        NSt [rpm];        FnG [l/min];        FR [l/h];        deltapG [bar];        my [1/h];        CAltot [mol/l]");

                % Change text on push button
                app.CallingApp.DataStorageStartButton.Text = "Data Storage Stop";
            end

            app.UIFigure.Name = "Simulation App";
        end

        % Callback function: ExitButton, UIFigure
        function UIFigureCloseRequest(app, event)
            % Check if timer is running and stop it if it is
            R = get(app.Timer, 'Running');
            if isequal(R ,'on')
                stop(app.Timer);
            end

            app.PauseButton.Text = "Resume";
            YN = uiconfirm(app.UIFigure,'Do you want to stop the simulation?', 'Close request');
            if strcmpi(YN,'OK')
                if app.CallingApp.StartSimulationButton.Text == "Stop Simulation"
                 app.CallingApp.EditStartingVariablesButton.Enable = "on";

                 app.CallingApp.ProcessTimehEditField.Value = 0.0;

                 app.CallingApp.ProcessRunningLamp.Color = "red";

                 app.CallingApp.StartSimulationButton.Text = "Start Simulation";

                 app.CallingApp.mds = 0;

                 app.CallingApp.SnapShot = 0;

                 app.CallingApp.PrevPlot = 0;

                 app.CallingApp.DataStorageRunningLamp.Color = "red";
                 app.CallingApp.DataStorageStartButton.Text = "Data Storage Start";
                end

                 delete(app)
            else
                start(app.Timer);
                app.PauseButton.Text = "Pause";
            end
        end

        % Button pushed function: ExportPlotButton
        function ExportPlotButtonPushed(app, event)
            filter = {'*.jpg';'*.png';'*.tif';'*.pdf'};
            if app.CallingApp.SelectedFile ~= "none"
                [filename,filepath] = uiputfile(filter,'Save plot of cultivation',strrep(datestr(datetime),':','-')+" "+app.CallingApp.FileName);
            else
                [filename,filepath] = uiputfile(filter);
            end
            if filename ~= 0
                exportgraphics(app.UIAxes,[filepath filename],"Resolution",300,'BackgroundColor','none','ContentType','vector');
                msgbox(sprintf('File was saved as %s',filename));
                app.ExportPlotButton.Text = "Export again";
            end
        end

        % Button pushed function: PauseButton
        function PauseButtonPushed(app, event)
             if app.PauseButton.Text == "Pause"
                 stop(app.Timer)
                 app.PauseButton.Text = "Resume";
             else
                 start(app.Timer)
                 app.PauseButton.Text = "Pause";
             end
        end
```

```
        % Value changed function: SpeedfactorSlider
        function SpeedfactorSliderValueChanged(app, event)
            app.timit = event.Value;
            stop(app.Timer); % CHECK IF TIMER WAS ALREADY STOPPED
            app.Timer.Period = 3600*app.CallingApp.deltat*app.timit^(-1);
            start(app.Timer);
        end

        % Value changed function: InoculationSwitch
        function InoculationSwitchValueChanged(app, event)
            app.inoculum = 1;

            app.InoculationSwitch.Enable = 0;

            app.Lamp.Color = "green";
        end

        % Value changed function: tlimEditField
        function tlimEditFieldValueChanged(app, event)
            if app.tlimminEditField.Value < app.tlimEditField.Value
                xlim(app.UIAxes,[app.tlimminEditField.Value app.tlimEditField.Value]);
            else
                xlim(app.UIAxes,[0 app.tlimEditField.Value]);
            end
        end

        % Value changed function: variablelimleftEditField
        function variablelimleftEditFieldValueChanged(app, event)
            if app.variablelimleftminEditField.Value < app.variablelimleftEditField.Value
                yyaxis(app.UIAxes,'left')
                ylim(app.UIAxes,[app.variablelimleftminEditField.Value app.variablelimleftEditField.Value]);
            else
                yyaxis(app.UIAxes,'left')
                ylim(app.UIAxes,[0 app.variablelimleftEditField.Value]);
            end
        end

        % Value changed function: variablelimrightEditField
        function variablelimrightEditFieldValueChanged(app, event)
            if app.variablelimrightminEditField.Value < app.variablelimrightEditField.Value
                yyaxis(app.UIAxes,'right')
                ylim(app.UIAxes,[app.variablelimrightminEditField.Value app.variablelimrightEditField.Value]);
            else
                yyaxis(app.UIAxes,'right')
                ylim(app.UIAxes,[0 app.variablelimrightEditField.Value]);
            end
        end

        % Button pushed function: SaveProcessButton
        function SaveProcessButtonPushed(app, event)
            app.PauseButtonPushed(app)

            props        = properties(app);
            values       = double.empty(0,length(props));
            lengths      = ones(1,length(props));
            enable       = cell(1,length(props));
            val          = double.empty(0,length(props));

            for i = 1:length(props)
                propName = props{i};
                if isnumeric(app.(propName))
                    values(1:length(app.(propName)),i) = app.(propName);
                    lengths(i)                         = length(app.(propName));
                end
                if isprop(app.(propName),'Enable')
                    enable{i} = app.(propName).Enable;
                end
                if isprop(app.(propName),'Value')
                    val(i) = app.(propName).Value;
                end
            end

            file = uiputfile('*.mat',"Save Message");
            if file
                save(file,'props','values','lengths','enable','val');
            end

            app.PauseButtonPushed(app)
        end

        % Value changed function: tlimminEditField
        function tlimminEditFieldValueChanged(app, event)
```

```
            if app.tlimminEditField.Value < app.tlimEditField.Value
                xlim(app.UIAxes,[app.tlimminEditField.Value app.tlimEditField.Value]);
            else
                xlim(app.UIAxes,[0 app.tlimEditField.Value]);
            end
        end

        % Value changed function: variablelimleftminEditField
        function variablelimleftminEditFieldValueChanged(app, event)
            if app.variablelimleftminEditField.Value < app.variablelimleftEditField.Value
                yyaxis(app.UIAxes,'left')
                ylim(app.UIAxes,[app.variablelimleftminEditField.Value app.variablelimleftEditField.Value]);
            else
                yyaxis(app.UIAxes,'left')
                ylim(app.UIAxes,[0 app.variablelimleftEditField.Value]);
            end
        end

        % Value changed function: variablelimrightminEditField
        function variablelimrightminEditFieldValueChanged(app, event)
            if app.variablelimrightminEditField.Value < app.variablelimrightEditField.Value
                yyaxis(app.UIAxes,'right')
                ylim(app.UIAxes,[app.variablelimrightminEditField.Value app.variablelimrightEditField.Value]);
            else
                yyaxis(app.UIAxes,'right')
                ylim(app.UIAxes,[0 app.variablelimrightEditField.Value]);
            end
        end

        % Callback function: Tree
        function TreeCheckedNodesChanged(app, event)
            if app.CallingApp.PrevPlot == 1
                plot_func(app)
            end
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 1068 517];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Cultivation Simulation')
            xlabel(app.UIAxes, 't [h]')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.Position = [28 111 653 390];

            % Create ExitButton
            app.ExitButton = uibutton(app.UIFigure, 'push');
            app.ExitButton.ButtonPushedFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);
            app.ExitButton.FontSize = 14;
            app.ExitButton.FontWeight = 'bold';
            app.ExitButton.Position = [716 15 100 36];
            app.ExitButton.Text = 'Exit';

            % Create PauseButton
            app.PauseButton = uibutton(app.UIFigure, 'push');
            app.PauseButton.ButtonPushedFcn = createCallbackFcn(app, @PauseButtonPushed, true);
            app.PauseButton.Position = [432 75 100 22];
            app.PauseButton.Text = 'Pause';

            % Create SpeedfactorSliderLabel
            app.SpeedfactorSliderLabel = uilabel(app.UIFigure);
            app.SpeedfactorSliderLabel.HorizontalAlignment = 'right';
            app.SpeedfactorSliderLabel.Position = [21 85 74 22];
            app.SpeedfactorSliderLabel.Text = 'Speed factor';

            % Create SpeedfactorSlider
            app.SpeedfactorSlider = uislider(app.UIFigure);
            app.SpeedfactorSlider.Limits = [1 5];
            app.SpeedfactorSlider.MajorTicks = [1 3 5];
            app.SpeedfactorSlider.MajorTickLabels = {'Real-Time', '3X faster', '5X faster'};
```

```
            app.SpeedfactorSlider.ValueChangedFcn = createCallbackFcn(app, @SpeedfactorSliderValueChanged, true);
            app.SpeedfactorSlider.MinorTicks = [];
            app.SpeedfactorSlider.Position = [116 94 200 3];
            app.SpeedfactorSlider.Value = 1;

            % Create TimeforOperationsEditFieldLabel
            app.TimeforOperationsEditFieldLabel = uilabel(app.UIFigure);
            app.TimeforOperationsEditFieldLabel.HorizontalAlignment = 'right';
            app.TimeforOperationsEditFieldLabel.Tooltip = {'This '};
            app.TimeforOperationsEditFieldLabel.Position = [316 22 121 22];
            app.TimeforOperationsEditFieldLabel.Text = 'Time for Operation [s]';

            % Create TimeforOperationsEditField
            app.TimeforOperationsEditField = uieditfield(app.UIFigure, 'numeric');
            app.TimeforOperationsEditField.ValueDisplayFormat = '%.5f';
            app.TimeforOperationsEditField.Editable = 'off';
            app.TimeforOperationsEditField.Tooltip = {'This edit field displays the time that is needed to calculate the
                  variable values for the next point in time. It can be used to assess the lower limit of delta t.'};
            app.TimeforOperationsEditField.Position = [451 22 66 22];

            % Create InoculationSwitchLabel
            app.InoculationSwitchLabel = uilabel(app.UIFigure);
            app.InoculationSwitchLabel.HorizontalAlignment = 'center';
            app.InoculationSwitchLabel.Position = [720 291 63 22];
            app.InoculationSwitchLabel.Text = 'Inoculation';

            % Create InoculationSwitch
            app.InoculationSwitch = uiswitch(app.UIFigure, 'slider');
            app.InoculationSwitch.ItemsData = {'0', '1'};
            app.InoculationSwitch.ValueChangedFcn = createCallbackFcn(app, @InoculationSwitchValueChanged, true);
            app.InoculationSwitch.Position = [729 262 45 20];
            app.InoculationSwitch.Value = '1';

            % Create Lamp
            app.Lamp = uilamp(app.UIFigure);
            app.Lamp.Position = [808 262 20 20];

            % Create qXpX1hEditFieldLabel
            app.qXpX1hEditFieldLabel = uilabel(app.UIFigure);
            app.qXpX1hEditFieldLabel.HorizontalAlignment = 'center';
            app.qXpX1hEditFieldLabel.Position = [716 387 62 22];
            app.qXpX1hEditFieldLabel.Text = 'qXpX [1/h]';

            % Create qXpX1hEditField
            app.qXpX1hEditField = uieditfield(app.UIFigure, 'numeric');
            app.qXpX1hEditField.ValueDisplayFormat = '%.5f';
            app.qXpX1hEditField.Position = [712 366 70 22];

            % Create LimitationTypeLabel
            app.LimitationTypeLabel = uilabel(app.UIFigure);
            app.LimitationTypeLabel.HorizontalAlignment = 'center';
            app.LimitationTypeLabel.FontSize = 14;
            app.LimitationTypeLabel.FontWeight = 'bold';
            app.LimitationTypeLabel.Position = [682 420 130 22];
            app.LimitationTypeLabel.Text = 'Limitation Type';

            % Create TimeofinoculationhEditFieldLabel
            app.TimeofinoculationhEditFieldLabel = uilabel(app.UIFigure);
            app.TimeofinoculationhEditFieldLabel.HorizontalAlignment = 'center';
            app.TimeofinoculationhEditFieldLabel.Position = [685 193 123 22];
            app.TimeofinoculationhEditFieldLabel.Text = 'Time of inoculation [h]';

            % Create TimeofinoculationhEditField
            app.TimeofinoculationhEditField = uieditfield(app.UIFigure, 'numeric');
            app.TimeofinoculationhEditField.ValueDisplayFormat = '%.5f';
            app.TimeofinoculationhEditField.Editable = 'off';
            app.TimeofinoculationhEditField.Position = [715 171 64 22];

            % Create PlotConfigurationPanel
            app.PlotConfigurationPanel = uipanel(app.UIFigure);
            app.PlotConfigurationPanel.Title = 'Plot Configuration';
            app.PlotConfigurationPanel.BackgroundColor = [1 1 1];
            app.PlotConfigurationPanel.Position = [838 1 274 517];

            % Create Tree
            app.Tree = uitree(app.PlotConfigurationPanel, 'checkbox');
            app.Tree.Position = [29 132 218 357];

            % Create PlotOptionsNode
            app.PlotOptionsNode = uitreenode(app.Tree);
            app.PlotOptionsNode.Text = 'Plot Options';

```

```
            % Create cXLglNode
            app.cXLglNode = uitreenode(app.PlotOptionsNode);
            app.cXLglNode.NodeData = 1;
            app.cXLglNode.Text = 'cXL [g/l]';

            % Create cS1LglNode
            app.cS1LglNode = uitreenode(app.PlotOptionsNode);
            app.cS1LglNode.NodeData = 2;
            app.cS1LglNode.Text = 'cS1L [g/l]';

            % Create cS2LglNode
            app.cS2LglNode = uitreenode(app.PlotOptionsNode);
            app.cS2LglNode.NodeData = 3;
            app.cS2LglNode.Text = 'cS2L [g/l]';

            % Create cS3LglNode
            app.cS3LglNode = uitreenode(app.PlotOptionsNode);
            app.cS3LglNode.NodeData = 4;
            app.cS3LglNode.Text = 'cS3L [g/l]';

            % Create FRlhNode
            app.FRlhNode = uitreenode(app.PlotOptionsNode);
            app.FRlhNode.NodeData = 5;
            app.FRlhNode.Text = 'FR [l/h]';

            % Create FHlhNode
            app.FHlhNode = uitreenode(app.PlotOptionsNode);
            app.FHlhNode.NodeData = 6;
            app.FHlhNode.Text = 'FH [l/h]';

            % Create NStrpmNode
            app.NStrpmNode = uitreenode(app.PlotOptionsNode);
            app.NStrpmNode.NodeData = 7;
            app.NStrpmNode.Text = 'NSt [rpm]';

            % Create FnGlminNode
            app.FnGlminNode = uitreenode(app.PlotOptionsNode);
            app.FnGlminNode.NodeData = 8;
            app.FnGlminNode.Text = 'FnG [l/min]';

            % Create pHLNode
            app.pHLNode = uitreenode(app.PlotOptionsNode);
            app.pHLNode.NodeData = 9;
            app.pHLNode.Text = 'pHL [-]';

            % Create thetaLCNode
            app.thetaLCNode = uitreenode(app.PlotOptionsNode);
            app.thetaLCNode.NodeData = 10;
            app.thetaLCNode.Text = 'thetaL [°C]';

            % Create pO2Node
            app.pO2Node = uitreenode(app.PlotOptionsNode);
            app.pO2Node.NodeData = 11;
            app.pO2Node.Text = 'pO2 [%]';

            % Create xOGNode
            app.xOGNode = uitreenode(app.PlotOptionsNode);
            app.xOGNode.NodeData = 12;
            app.xOGNode.Text = 'xOG [-]';

            % Create cOLglNode
            app.cOLglNode = uitreenode(app.PlotOptionsNode);
            app.cOLglNode.NodeData = 16;
            app.cOLglNode.Text = 'cOL [g/l]';

            % Create xOLNode
            app.xOLNode = uitreenode(app.PlotOptionsNode);
            app.xOLNode.NodeData = 13;
            app.xOLNode.Text = 'xOL [-]';

            % Create hNode
            app.hNode = uitreenode(app.PlotOptionsNode);
            app.hNode.NodeData = 14;
            app.hNode.Text = 'μ [1/h]';

            % Create CAltotmollNode
            app.CAltotmollNode = uitreenode(app.PlotOptionsNode);
            app.CAltotmollNode.NodeData = 15;
            app.CAltotmollNode.Text = 'CAltot [mol/l]';

            % Create addedacidlNode
            app.addedacidlNode = uitreenode(app.PlotOptionsNode);
```

```
            app.addedacidlNode.NodeData = 17;
            app.addedacidlNode.Text = 'added acid [l]';

            % Create addedbaselNode
            app.addedbaselNode = uitreenode(app.PlotOptionsNode);
            app.addedbaselNode.NodeData = 18;
            app.addedbaselNode.Text = 'added base [l]';

            % Create addedantifoamlNode
            app.addedantifoamlNode = uitreenode(app.PlotOptionsNode);
            app.addedantifoamlNode.NodeData = 19;
            app.addedantifoamlNode.Text = 'added antifoam [l]';

            % Create addedfeedlNode
            app.addedfeedlNode = uitreenode(app.PlotOptionsNode);
            app.addedfeedlNode.NodeData = 20;
            app.addedfeedlNode.Text = 'added feed [l]';

            % Create VLlNode
            app.VLlNode = uitreenode(app.PlotOptionsNode);
            app.VLlNode.NodeData = 21;
            app.VLlNode.Text = 'VL [l]';

            % Create xOGinNode
            app.xOGinNode = uitreenode(app.PlotOptionsNode);
            app.xOGinNode.NodeData = 22;
            app.xOGinNode.Text = 'xOGin [-]';

            % Create pO2wNode
            app.pO2wNode = uitreenode(app.PlotOptionsNode);
            app.pO2wNode.NodeData = 23;
            app.pO2wNode.Text = 'pO2w [%]';

            % Assign Checked Nodes
            app.Tree.CheckedNodes = [app.cXLglNode, app.cS1LglNode, app.pO2Node];
            % Assign Checked Nodes
            app.Tree.CheckedNodesChangedFcn = createCallbackFcn(app, @TreeCheckedNodesChanged, true);

            % Create tlimEditFieldLabel
            app.tlimEditFieldLabel = uilabel(app.PlotConfigurationPanel);
            app.tlimEditFieldLabel.HorizontalAlignment = 'right';
            app.tlimEditFieldLabel.Position = [10 69 27 22];
            app.tlimEditFieldLabel.Text = 't lim';

            % Create tlimEditField
            app.tlimEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.tlimEditField.ValueChangedFcn = createCallbackFcn(app, @tlimEditFieldValueChanged, true);
            app.tlimEditField.Position = [194 69 48 22];

            % Create variablelimleftEditFieldLabel
            app.variablelimleftEditFieldLabel = uilabel(app.PlotConfigurationPanel);
            app.variablelimleftEditFieldLabel.HorizontalAlignment = 'right';
            app.variablelimleftEditFieldLabel.Position = [8 42 94 22];
            app.variablelimleftEditFieldLabel.Text = 'variable lim (left)';

            % Create variablelimleftEditField
            app.variablelimleftEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.variablelimleftEditField.ValueChangedFcn = createCallbackFcn(app, @variablelimleftEditFieldValueChanged, true)
                ;
            app.variablelimleftEditField.Position = [194 42 48 22];

            % Create variablelimrightEditFieldLabel
            app.variablelimrightEditFieldLabel = uilabel(app.PlotConfigurationPanel);
            app.variablelimrightEditFieldLabel.HorizontalAlignment = 'right';
            app.variablelimrightEditFieldLabel.Position = [8 16 101 22];
            app.variablelimrightEditFieldLabel.Text = 'variable lim (right)';

            % Create variablelimrightEditField
            app.variablelimrightEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.variablelimrightEditField.ValueChangedFcn = createCallbackFcn(app, @variablelimrightEditFieldValueChanged,
                true);
            app.variablelimrightEditField.Position = [194 16 48 22];

            % Create MaxLabel
            app.MaxLabel = uilabel(app.PlotConfigurationPanel);
            app.MaxLabel.HorizontalAlignment = 'center';
            app.MaxLabel.Position = [204 95 28 22];
            app.MaxLabel.Text = 'Max';

            % Create MinLabel
            app.MinLabel = uilabel(app.PlotConfigurationPanel);
            app.MinLabel.HorizontalAlignment = 'center';
```

```
            app.MinLabel.Position = [139 95 25 22];
            app.MinLabel.Text = 'Min';

            % Create tlimminEditField
            app.tlimminEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.tlimminEditField.ValueChangedFcn = createCallbackFcn(app, @tlimminEditFieldValueChanged, true);
            app.tlimminEditField.Position = [128 69 48 22];

            % Create variablelimleftminEditField
            app.variablelimleftminEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.variablelimleftminEditField.ValueChangedFcn = createCallbackFcn(app, @variablelimleftminEditFieldValueChanged,
                true);
            app.variablelimleftminEditField.Position = [128 42 48 22];

            % Create variablelimrightminEditField
            app.variablelimrightminEditField = uieditfield(app.PlotConfigurationPanel, 'numeric');
            app.variablelimrightminEditField.ValueChangedFcn = createCallbackFcn(app, @
                variablelimrightminEditFieldValueChanged, true);
            app.variablelimrightminEditField.Position = [128 16 48 22];

            % Create SaveProcessButton
            app.SaveProcessButton = uibutton(app.UIFigure, 'push');
            app.SaveProcessButton.ButtonPushedFcn = createCallbackFcn(app, @SaveProcessButtonPushed, true);
            app.SaveProcessButton.Position = [716 74 100 23];
            app.SaveProcessButton.Text = 'Save Process';

            % Create ExportPlotButton
            app.ExportPlotButton = uibutton(app.UIFigure, 'push');
            app.ExportPlotButton.ButtonPushedFcn = createCallbackFcn(app, @ExportPlotButtonPushed, true);
            app.ExportPlotButton.Position = [55 22 100 22];
            app.ExportPlotButton.Text = 'Export Plot';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = FigureAppODESystemLuttmannpHTEMPexported(varargin)

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The starting variables dialogue box

```
classdef DialogBox1exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                matlab.ui.Figure
        Label                   matlab.ui.control.Label
        StartingVariablesLabel  matlab.ui.control.Label
        tmaxhEditField          matlab.ui.control.NumericEditField
        tmaxhEditFieldLabel     matlab.ui.control.Label
        VR0lEditField           matlab.ui.control.NumericEditField
        VR0lEditFieldLabel      matlab.ui.control.Label
        ConfirmButton           matlab.ui.control.Button
        cS1RglEditField         matlab.ui.control.NumericEditField
        cS1RglEditFieldLabel    matlab.ui.control.Label
        VLmaxlEditField         matlab.ui.control.NumericEditField
        VLmaxlEditFieldLabel    matlab.ui.control.Label
        cS1L0glEditField        matlab.ui.control.NumericEditField
        cS1L0glEditFieldLabel   matlab.ui.control.Label
        cXL0glEditField         matlab.ui.control.NumericEditField
        cXL0glEditFieldLabel    matlab.ui.control.Label
    end


    properties (Access = public)
        CallingApp   % Main app object
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Store main app in property for CloseRequestFcn to use
            app.CallingApp = mainapp;

            % Update UI with input values
            app.cXL0glEditField.Value     = app.CallingApp.cXL0;
            app.cS1L0glEditField.Value    = app.CallingApp.cS1L0;
            app.VLmaxlEditField.Value     = app.CallingApp.VLmax;
            app.cS1RglEditField.Value     = app.CallingApp.cS1R;
            app.VR0lEditField.Value       = app.CallingApp.VR0;
            app.tmaxhEditField.Value      = app.CallingApp.tmax;

            if app.CallingApp.FileName ~= "none"
                app.Label.Text = app.CallingApp.FileName;
            else
                app.Label.Text = "";
            end

            % Define window name
            app.UIFigure.Name = "Starting Variables Edit Window";
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            % Enable the Plot Opions button in main app
            app.CallingApp.EditStartingVariablesButton.Enable = 'on';

            % Delete the dialog box
            delete(app)
        end

        % Button pushed function: ConfirmButton
        function ConfirmButtonPushed(app, event)
            % Store inputs as properties of mainapp
            app.CallingApp.cXL0    = app.cXL0glEditField.Value;
            app.CallingApp.cS1L0   = app.cS1L0glEditField.Value;
            app.CallingApp.VLmax   = app.VLmaxlEditField.Value;
            app.CallingApp.cS1R    = app.cS1RglEditField.Value;
            app.CallingApp.VR0     = app.VR0lEditField.Value;
            app.CallingApp.tmax    = app.tmaxhEditField.Value;

            % Call close request function
            UIFigureCloseRequest(app)
        end
    end

    % Component initialization
```

```
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 256 342];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create cXL0glEditFieldLabel
            app.cXL0glEditFieldLabel = uilabel(app.UIFigure);
            app.cXL0glEditFieldLabel.HorizontalAlignment = 'right';
            app.cXL0glEditFieldLabel.Position = [23 238 86 22];
            app.cXL0glEditFieldLabel.Text = 'cXL0 [g/l]';

            % Create cXL0glEditField
            app.cXL0glEditField = uieditfield(app.UIFigure, 'numeric');
            app.cXL0glEditField.Limits = [0.05 10];
            app.cXL0glEditField.Position = [127 238 108 22];
            app.cXL0glEditField.Value = 0.5;

            % Create cS1L0glEditFieldLabel
            app.cS1L0glEditFieldLabel = uilabel(app.UIFigure);
            app.cS1L0glEditFieldLabel.HorizontalAlignment = 'right';
            app.cS1L0glEditFieldLabel.Position = [35 209 74 22];
            app.cS1L0glEditFieldLabel.Text = 'cS1L0 [g/l]';

            % Create cS1L0glEditField
            app.cS1L0glEditField = uieditfield(app.UIFigure, 'numeric');
            app.cS1L0glEditField.Limits = [0 10];
            app.cS1L0glEditField.Position = [127 209 108 22];
            app.cS1L0glEditField.Value = 0.5;

            % Create VLmaxlEditFieldLabel
            app.VLmaxlEditFieldLabel = uilabel(app.UIFigure);
            app.VLmaxlEditFieldLabel.HorizontalAlignment = 'right';
            app.VLmaxlEditFieldLabel.Position = [53 149 55 22];
            app.VLmaxlEditFieldLabel.Text = 'VLmax [l]';

            % Create VLmaxlEditField
            app.VLmaxlEditField = uieditfield(app.UIFigure, 'numeric');
            app.VLmaxlEditField.Limits = [1 50];
            app.VLmaxlEditField.Position = [126 149 108 22];
            app.VLmaxlEditField.Value = 6;

            % Create cS1RglEditFieldLabel
            app.cS1RglEditFieldLabel = uilabel(app.UIFigure);
            app.cS1RglEditFieldLabel.HorizontalAlignment = 'right';
            app.cS1RglEditFieldLabel.Position = [50 120 58 22];
            app.cS1RglEditFieldLabel.Text = 'cS1R [g/l]';

            % Create cS1RglEditField
            app.cS1RglEditField = uieditfield(app.UIFigure, 'numeric');
            app.cS1RglEditField.Limits = [20 700];
            app.cS1RglEditField.Position = [126 120 108 22];
            app.cS1RglEditField.Value = 150;

            % Create ConfirmButton
            app.ConfirmButton = uibutton(app.UIFigure, 'push');
            app.ConfirmButton.ButtonPushedFcn = createCallbackFcn(app, @ConfirmButtonPushed, true);
            app.ConfirmButton.Position = [131 16 100 23];
            app.ConfirmButton.Text = 'Confirm';

            % Create VR0lEditFieldLabel
            app.VR0lEditFieldLabel = uilabel(app.UIFigure);
            app.VR0lEditFieldLabel.HorizontalAlignment = 'right';
            app.VR0lEditFieldLabel.Position = [66 90 42 22];
            app.VR0lEditFieldLabel.Text = 'VR0 [l]';

            % Create VR0lEditField
            app.VR0lEditField = uieditfield(app.UIFigure, 'numeric');
            app.VR0lEditField.Limits = [0.1 10];
            app.VR0lEditField.Position = [126 90 108 22];
            app.VR0lEditField.Value = 5;

            % Create tmaxhEditFieldLabel
            app.tmaxhEditFieldLabel = uilabel(app.UIFigure);
            app.tmaxhEditFieldLabel.HorizontalAlignment = 'right';
            app.tmaxhEditFieldLabel.Position = [60 60 48 22];
```

```
            app.tmaxhEditFieldLabel.Text = 'tmax [h]';

            % Create tmaxhEditField
            app.tmaxhEditField = uieditfield(app.UIFigure, 'numeric');
            app.tmaxhEditField.Limits = [0.1 50];
            app.tmaxhEditField.Position = [126 60 108 22];
            app.tmaxhEditField.Value = 1;

            % Create StartingVariablesLabel
            app.StartingVariablesLabel = uilabel(app.UIFigure);
            app.StartingVariablesLabel.HorizontalAlignment = 'center';
            app.StartingVariablesLabel.FontSize = 16;
            app.StartingVariablesLabel.FontWeight = 'bold';
            app.StartingVariablesLabel.Position = [54 308 149 22];
            app.StartingVariablesLabel.Text = 'Starting Variables';

            % Create Label
            app.Label = uilabel(app.UIFigure);
            app.Label.HorizontalAlignment = 'center';
            app.Label.VerticalAlignment = 'top';
            app.Label.WordWrap = 'on';
            app.Label.Position = [23 268 212 35];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = DialogBox1exported(varargin)

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The pH controller parameters dialogue box

```
classdef DialogBox2exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        KP_slavebaseEditField       matlab.ui.control.NumericEditField
        KP_slavebaseEditFieldLabel  matlab.ui.control.Label
        ConfirmButton               matlab.ui.control.Button
        KP_slaveacidEditField       matlab.ui.control.NumericEditField
        KP_slaveacidEditFieldLabel  matlab.ui.control.Label
        KP_masterEditField          matlab.ui.control.NumericEditField
        KP_masterEditFieldLabel     matlab.ui.control.Label
        SlaveControllerLabel        matlab.ui.control.Label
        MasterControllerLabel       matlab.ui.control.Label
        ParametersofthepHControllerLabel  matlab.ui.control.Label
    end


    properties (Access = public)
        CallingApp   % Main app object
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Store main app in property for CloseRequestFcn to use
            app.CallingApp = mainapp;

            % Update UI with input values
            app.KP_masterEditField.Value     = app.CallingApp.KP_pH;
            app.KP_slaveacidEditField.Value  = app.CallingApp.KP_pH2a;
            app.KP_slavebaseEditField.Value  = app.CallingApp.KP_pH2b;

            % Define window name
            app.UIFigure.Name = "pH Controller Parameters";
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            % Enable the Plot Opions button in main app
            app.CallingApp.pHParametersButton.Enable = 'on';

            % Delete the dialog box
            delete(app)
        end

        % Button pushed function: ConfirmButton
        function ConfirmButtonPushed(app, event)
            % Store inputs as properties of mainapp
            app.CallingApp.KP_pH   = app.KP_masterEditField.Value;
            app.CallingApp.KP_pH2a = app.KP_slaveacidEditField.Value;
            app.CallingApp.KP_pH2b = app.KP_slavebaseEditField.Value;

            % Call close request function
            UIFigureCloseRequest(app)
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 216 301];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create ParametersofthepHControllerLabel
            app.ParametersofthepHControllerLabel = uilabel(app.UIFigure);
            app.ParametersofthepHControllerLabel.HorizontalAlignment = 'center';
            app.ParametersofthepHControllerLabel.WordWrap = 'on';
            app.ParametersofthepHControllerLabel.FontSize = 14;
```

```
            app.ParametersofthepHControllerLabel.FontWeight = 'bold';
            app.ParametersofthepHControllerLabel.Position = [0 249 218 34];
            app.ParametersofthepHControllerLabel.Text = {'Parameters of the'; 'pH Controller'};

            % Create MasterControllerLabel
            app.MasterControllerLabel = uilabel(app.UIFigure);
            app.MasterControllerLabel.Position = [60 209 98 22];
            app.MasterControllerLabel.Text = 'Master Controller';

            % Create SlaveControllerLabel
            app.SlaveControllerLabel = uilabel(app.UIFigure);
            app.SlaveControllerLabel.Position = [63 136 91 22];
            app.SlaveControllerLabel.Text = 'Slave Controller';

            % Create KP_masterEditFieldLabel
            app.KP_masterEditFieldLabel = uilabel(app.UIFigure);
            app.KP_masterEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_masterEditFieldLabel.Position = [39 179 65 22];
            app.KP_masterEditFieldLabel.Text = 'KP_master';

            % Create KP_masterEditField
            app.KP_masterEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_masterEditField.ValueDisplayFormat = '%.4f';
            app.KP_masterEditField.Position = [119 179 60 22];

            % Create KP_slaveacidEditFieldLabel
            app.KP_slaveacidEditFieldLabel = uilabel(app.UIFigure);
            app.KP_slaveacidEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_slaveacidEditFieldLabel.Position = [23 103 82 22];
            app.KP_slaveacidEditFieldLabel.Text = 'KP_slave acid';

            % Create KP_slaveacidEditField
            app.KP_slaveacidEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_slaveacidEditField.ValueDisplayFormat = '%.4f';
            app.KP_slaveacidEditField.Position = [119 103 60 22];

            % Create ConfirmButton
            app.ConfirmButton = uibutton(app.UIFigure, 'push');
            app.ConfirmButton.ButtonPushedFcn = createCallbackFcn(app, @ConfirmButtonPushed, true);
            app.ConfirmButton.Position = [59 19 100 23];
            app.ConfirmButton.Text = 'Confirm';

            % Create KP_slavebaseEditFieldLabel
            app.KP_slavebaseEditFieldLabel = uilabel(app.UIFigure);
            app.KP_slavebaseEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_slavebaseEditFieldLabel.Position = [23 69 86 22];
            app.KP_slavebaseEditFieldLabel.Text = 'KP_slave base';

            % Create KP_slavebaseEditField
            app.KP_slavebaseEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_slavebaseEditField.ValueDisplayFormat = '%.4f';
            app.KP_slavebaseEditField.Position = [119 69 60 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = DialogBox2exported(varargin)

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
```

```
            delete(app.UIFigure)
        end
    end
end
```

The temperature controller parameters dialogue box

```
classdef DialogBox3exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                       matlab.ui.Figure
        KP_slavecoolingEditField       matlab.ui.control.NumericEditField
        KP_slavecoolingEditFieldLabel  matlab.ui.control.Label
        KI_masterEditField             matlab.ui.control.NumericEditField
        KI_masterEditFieldLabel        matlab.ui.control.Label
        ConfirmButton                  matlab.ui.control.Button
        KP_slaveheatingEditField       matlab.ui.control.NumericEditField
        KP_slaveheatingEditFieldLabel  matlab.ui.control.Label
        KP_masterEditField             matlab.ui.control.NumericEditField
        KP_masterEditFieldLabel        matlab.ui.control.Label
        SlaveControllerLabel           matlab.ui.control.Label
        MasterControllerLabel          matlab.ui.control.Label
        ParametersoftheTemperatureControllerLabel  matlab.ui.control.Label
    end


    properties (Access = public)
        CallingApp   % Main app object
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Store main app in property for CloseRequestFcn to use
            app.CallingApp = mainapp;

            % Update UI with input values
            app.KP_masterEditField.Value        = app.CallingApp.KP_temp1;
            app.KI_masterEditField.Value        = app.CallingApp.KI_temp1;
            app.KP_slaveheatingEditField.Value  = app.CallingApp.KP_temp2h;
            app.KP_slavecoolingEditField.Value  = app.CallingApp.KP_temp2c;

            % Define window name
            app.UIFigure.Name = "Temperature Controller Parameters";
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            % Enable the Plot Opions button in main app
            app.CallingApp.tempParametersButton.Enable = 'on';

            % Delete the dialog box
            delete(app)
        end

        % Button pushed function: ConfirmButton
        function ConfirmButtonPushed(app, event)
            % Store inputs as properties of mainapp
            app.CallingApp.KP_temp1 = app.KP_masterEditField.Value;
            app.CallingApp.KI_temp1   = app.KI_masterEditField.Value;
            app.CallingApp.KP_temp2h = app.KP_slaveheatingEditField.Value;
            app.CallingApp.KP_temp2c = app.KP_slavecoolingEditField.Value;

            % Call close request function
            UIFigureCloseRequest(app)
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 251 312];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create ParametersoftheTemperatureControllerLabel
```

```
            app.ParametersoftheTemperatureControllerLabel = uilabel(app.UIFigure);
            app.ParametersoftheTemperatureControllerLabel.HorizontalAlignment = 'center';
            app.ParametersoftheTemperatureControllerLabel.WordWrap = 'on';
            app.ParametersoftheTemperatureControllerLabel.FontSize = 14;
            app.ParametersoftheTemperatureControllerLabel.FontWeight = 'bold';
            app.ParametersoftheTemperatureControllerLabel.Position = [20 260 218 34];
            app.ParametersoftheTemperatureControllerLabel.Text = {'Parameters of the'; 'Temperature Controller'};

            % Create MasterControllerLabel
            app.MasterControllerLabel = uilabel(app.UIFigure);
            app.MasterControllerLabel.Position = [80 227 98 22];
            app.MasterControllerLabel.Text = 'Master Controller';

            % Create SlaveControllerLabel
            app.SlaveControllerLabel = uilabel(app.UIFigure);
            app.SlaveControllerLabel.Position = [83 133 91 22];
            app.SlaveControllerLabel.Text = 'Slave Controller';

            % Create KP_masterEditFieldLabel
            app.KP_masterEditFieldLabel = uilabel(app.UIFigure);
            app.KP_masterEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_masterEditFieldLabel.Position = [59 197 65 22];
            app.KP_masterEditFieldLabel.Text = 'KP_master';

            % Create KP_masterEditField
            app.KP_masterEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_masterEditField.ValueDisplayFormat = '%.4f';
            app.KP_masterEditField.Position = [139 197 60 22];

            % Create KP_slaveheatingEditFieldLabel
            app.KP_slaveheatingEditFieldLabel = uilabel(app.UIFigure);
            app.KP_slaveheatingEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_slaveheatingEditFieldLabel.Position = [29 101 103 22];
            app.KP_slaveheatingEditFieldLabel.Text = 'KP_slave (heating';

            % Create KP_slaveheatingEditField
            app.KP_slaveheatingEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_slaveheatingEditField.ValueDisplayFormat = '%.4f';
            app.KP_slaveheatingEditField.Position = [139 101 60 22];

            % Create ConfirmButton
            app.ConfirmButton = uibutton(app.UIFigure, 'push');
            app.ConfirmButton.ButtonPushedFcn = createCallbackFcn(app, @ConfirmButtonPushed, true);
            app.ConfirmButton.Position = [79 19 100 23];
            app.ConfirmButton.Text = 'Confirm';

            % Create KI_masterEditFieldLabel
            app.KI_masterEditFieldLabel = uilabel(app.UIFigure);
            app.KI_masterEditFieldLabel.HorizontalAlignment = 'right';
            app.KI_masterEditFieldLabel.Position = [59 166 60 22];
            app.KI_masterEditFieldLabel.Text = 'KI_master';

            % Create KI_masterEditField
            app.KI_masterEditField = uieditfield(app.UIFigure, 'numeric');
            app.KI_masterEditField.ValueDisplayFormat = '%.4f';
            app.KI_masterEditField.Position = [139 166 60 22];

            % Create KP_slavecoolingEditFieldLabel
            app.KP_slavecoolingEditFieldLabel = uilabel(app.UIFigure);
            app.KP_slavecoolingEditFieldLabel.HorizontalAlignment = 'right';
            app.KP_slavecoolingEditFieldLabel.Position = [28 67 106 22];
            app.KP_slavecoolingEditFieldLabel.Text = 'KP_slave (cooling)';

            % Create KP_slavecoolingEditField
            app.KP_slavecoolingEditField = uieditfield(app.UIFigure, 'numeric');
            app.KP_slavecoolingEditField.ValueDisplayFormat = '%.4f';
            app.KP_slavecoolingEditField.Position = [139 67 60 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = DialogBox3exported(varargin)

            % Create UIFigure and components
            createComponents(app)

```

```
            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The p_{O2} controller parameters dialogue box

```
classdef DialogBox4exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure             matlab.ui.Figure
        KDEditField_4        matlab.ui.control.NumericEditField
        KDEditField_4Label   matlab.ui.control.Label
        KIEditField_4        matlab.ui.control.NumericEditField
        KIEditField_4Label   matlab.ui.control.Label
        KPEditField_4        matlab.ui.control.NumericEditField
        KPEditField_4Label   matlab.ui.control.Label
        KDEditField_3        matlab.ui.control.NumericEditField
        KDEditField_3Label   matlab.ui.control.Label
        KIEditField_3        matlab.ui.control.NumericEditField
        KIEditField_3Label   matlab.ui.control.Label
        KPEditField_3        matlab.ui.control.NumericEditField
        KPEditField_3Label   matlab.ui.control.Label
        KDEditField_2        matlab.ui.control.NumericEditField
        KDEditField_2Label   matlab.ui.control.Label
        KIEditField_2        matlab.ui.control.NumericEditField
        KIEditField_2Label   matlab.ui.control.Label
        KPEditField_2        matlab.ui.control.NumericEditField
        KPEditField_2Label   matlab.ui.control.Label
        KDEditField          matlab.ui.control.NumericEditField
        KDEditFieldLabel     matlab.ui.control.Label
        FeedLabel            matlab.ui.control.Label
        GasmixLabel          matlab.ui.control.Label
        KIEditField          matlab.ui.control.NumericEditField
        KIEditFieldLabel     matlab.ui.control.Label
        ParametersofthepO2ControllerLabel  matlab.ui.control.Label
        KPEditField          matlab.ui.control.NumericEditField
        KPEditFieldLabel     matlab.ui.control.Label
        ConfirmButton        matlab.ui.control.Button
        AerationLabel        matlab.ui.control.Label
        AgitationLabel       matlab.ui.control.Label
    end


    properties (Access = public)
        CallingApp   % Main app object
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Store main app in property for CloseRequestFcn to use
            app.CallingApp = mainapp;

            % Update UI with input values
            app.KPEditField.Value = app.CallingApp.KP_agi;
            app.KIEditField.Value = app.CallingApp.KI_agi;
            app.KDEditField.Value = app.CallingApp.KD_agi;

            app.KPEditField_2.Value = app.CallingApp.KP_gasmix;
            app.KIEditField_2.Value = app.CallingApp.KI_gasmix;
            app.KDEditField_2.Value = app.CallingApp.KD_gasmix;

            app.KPEditField_3.Value = app.CallingApp.KP_aeration;
            app.KIEditField_3.Value = app.CallingApp.KI_aeration;
            app.KDEditField_3.Value = app.CallingApp.KD_aeration;

            app.KPEditField_4.Value = app.CallingApp.KP_feed;
            app.KIEditField_4.Value = app.CallingApp.KI_feed;
            app.KDEditField_4.Value = app.CallingApp.KD_feed;

            % Define window name
            app.UIFigure.Name = "pO2 Controller Parameters";
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            % Enable the Plot Opions button in main app
            app.CallingApp.pO2ParametersButton.Enable = 'on';

            % Delete the dialog box
            delete(app)
```

```
        end

        % Button pushed function: ConfirmButton
        function ConfirmButtonPushed(app, event)
            % Store inputs as properties of mainapp
            app.CallingApp.KP_agi = app.KPEditField.Value;
            app.CallingApp.KI_agi = app.KIEditField.Value;
            app.CallingApp.KD_agi = app.KDEditField.Value;

            app.CallingApp.KP_gasmix = app.KPEditField_2.Value;
            app.CallingApp.KI_gasmix = app.KIEditField_2.Value;
            app.CallingApp.KD_gasmix = app.KDEditField_2.Value;

            app.CallingApp.KP_aeration = app.KPEditField_3.Value;
            app.CallingApp.KI_aeration = app.KIEditField_3.Value;
            app.CallingApp.KD_aeration = app.KDEditField_3.Value;

            app.CallingApp.KP_feed = app.KPEditField_4.Value;
            app.CallingApp.KI_feed = app.KIEditField_4.Value;
            app.CallingApp.KD_feed = app.KDEditField_4.Value;

            % Call close request function
            UIFigureCloseRequest(app)
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 353 384];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create AgitationLabel
            app.AgitationLabel = uilabel(app.UIFigure);
            app.AgitationLabel.HorizontalAlignment = 'center';
            app.AgitationLabel.Position = [47 307 52 22];
            app.AgitationLabel.Text = 'Agitation';

            % Create AerationLabel
            app.AerationLabel = uilabel(app.UIFigure);
            app.AerationLabel.HorizontalAlignment = 'center';
            app.AerationLabel.Position = [48 165 50 22];
            app.AerationLabel.Text = 'Aeration';

            % Create ConfirmButton
            app.ConfirmButton = uibutton(app.UIFigure, 'push');
            app.ConfirmButton.ButtonPushedFcn = createCallbackFcn(app, @ConfirmButtonPushed, true);
            app.ConfirmButton.Position = [128 14 100 23];
            app.ConfirmButton.Text = 'Confirm';

            % Create KPEditFieldLabel
            app.KPEditFieldLabel = uilabel(app.UIFigure);
            app.KPEditFieldLabel.Position = [43 278 25 22];
            app.KPEditFieldLabel.Text = 'KP';

            % Create KPEditField
            app.KPEditField = uieditfield(app.UIFigure, 'numeric');
            app.KPEditField.ValueDisplayFormat = '%.4f';
            app.KPEditField.Position = [83 278 60 22];

            % Create ParametersofthepO2ControllerLabel
            app.ParametersofthepO2ControllerLabel = uilabel(app.UIFigure);
            app.ParametersofthepO2ControllerLabel.HorizontalAlignment = 'center';
            app.ParametersofthepO2ControllerLabel.WordWrap = 'on';
            app.ParametersofthepO2ControllerLabel.FontSize = 14;
            app.ParametersofthepO2ControllerLabel.FontWeight = 'bold';
            app.ParametersofthepO2ControllerLabel.Position = [-18 344 392 22];
            app.ParametersofthepO2ControllerLabel.Text = 'Parameters of the pO2 Controller';

            % Create KIEditFieldLabel
            app.KIEditFieldLabel = uilabel(app.UIFigure);
            app.KIEditFieldLabel.Position = [43 244 25 22];
            app.KIEditFieldLabel.Text = 'KI';

            % Create KIEditField
```

```
            app.KIEditField = uieditfield(app.UIFigure, 'numeric');
            app.KIEditField.ValueDisplayFormat = '%.4f';
            app.KIEditField.Position = [83 244 60 22];

            % Create GasmixLabel
            app.GasmixLabel = uilabel(app.UIFigure);
            app.GasmixLabel.HorizontalAlignment = 'center';
            app.GasmixLabel.Position = [243 307 46 22];
            app.GasmixLabel.Text = 'Gasmix';

            % Create FeedLabel
            app.FeedLabel = uilabel(app.UIFigure);
            app.FeedLabel.HorizontalAlignment = 'center';
            app.FeedLabel.Position = [251 165 33 22];
            app.FeedLabel.Text = 'Feed';

            % Create KDEditFieldLabel
            app.KDEditFieldLabel = uilabel(app.UIFigure);
            app.KDEditFieldLabel.Position = [43 211 25 22];
            app.KDEditFieldLabel.Text = 'KD';

            % Create KDEditField
            app.KDEditField = uieditfield(app.UIFigure, 'numeric');
            app.KDEditField.ValueDisplayFormat = '%.4f';
            app.KDEditField.Position = [83 211 60 22];

            % Create KPEditField_2Label
            app.KPEditField_2Label = uilabel(app.UIFigure);
            app.KPEditField_2Label.Position = [216 278 25 22];
            app.KPEditField_2Label.Text = 'KP';

            % Create KPEditField_2
            app.KPEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.KPEditField_2.ValueDisplayFormat = '%.4f';
            app.KPEditField_2.Position = [256 278 60 22];

            % Create KIEditField_2Label
            app.KIEditField_2Label = uilabel(app.UIFigure);
            app.KIEditField_2Label.Position = [216 244 25 22];
            app.KIEditField_2Label.Text = 'KI';

            % Create KIEditField_2
            app.KIEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.KIEditField_2.ValueDisplayFormat = '%.4f';
            app.KIEditField_2.Position = [256 244 60 22];

            % Create KDEditField_2Label
            app.KDEditField_2Label = uilabel(app.UIFigure);
            app.KDEditField_2Label.Position = [216 211 25 22];
            app.KDEditField_2Label.Text = 'KD';

            % Create KDEditField_2
            app.KDEditField_2 = uieditfield(app.UIFigure, 'numeric');
            app.KDEditField_2.ValueDisplayFormat = '%.4f';
            app.KDEditField_2.Position = [256 211 60 22];

            % Create KPEditField_3Label
            app.KPEditField_3Label = uilabel(app.UIFigure);
            app.KPEditField_3Label.Position = [43 134 25 22];
            app.KPEditField_3Label.Text = 'KP';

            % Create KPEditField_3
            app.KPEditField_3 = uieditfield(app.UIFigure, 'numeric');
            app.KPEditField_3.ValueDisplayFormat = '%.4f';
            app.KPEditField_3.Position = [83 134 60 22];

            % Create KIEditField_3Label
            app.KIEditField_3Label = uilabel(app.UIFigure);
            app.KIEditField_3Label.Position = [43 100 25 22];
            app.KIEditField_3Label.Text = 'KI';

            % Create KIEditField_3
            app.KIEditField_3 = uieditfield(app.UIFigure, 'numeric');
            app.KIEditField_3.ValueDisplayFormat = '%.4f';
            app.KIEditField_3.Position = [83 100 60 22];

            % Create KDEditField_3Label
            app.KDEditField_3Label = uilabel(app.UIFigure);
            app.KDEditField_3Label.Position = [43 67 25 22];
            app.KDEditField_3Label.Text = 'KD';

            % Create KDEditField_3
```

```
            app.KDEditField_3 = uieditfield(app.UIFigure, 'numeric');
            app.KDEditField_3.ValueDisplayFormat = '%.4f';
            app.KDEditField_3.Position = [83 67 60 22];

            % Create KPEditField_4Label
            app.KPEditField_4Label = uilabel(app.UIFigure);
            app.KPEditField_4Label.Position = [216 134 25 22];
            app.KPEditField_4Label.Text = 'KP';

            % Create KPEditField_4
            app.KPEditField_4 = uieditfield(app.UIFigure, 'numeric');
            app.KPEditField_4.ValueDisplayFormat = '%.4f';
            app.KPEditField_4.Position = [256 134 60 22];

            % Create KIEditField_4Label
            app.KIEditField_4Label = uilabel(app.UIFigure);
            app.KIEditField_4Label.Position = [216 100 25 22];
            app.KIEditField_4Label.Text = 'KI';

            % Create KIEditField_4
            app.KIEditField_4 = uieditfield(app.UIFigure, 'numeric');
            app.KIEditField_4.ValueDisplayFormat = '%.4f';
            app.KIEditField_4.Position = [256 100 60 22];

            % Create KDEditField_4Label
            app.KDEditField_4Label = uilabel(app.UIFigure);
            app.KDEditField_4Label.Position = [216 67 25 22];
            app.KDEditField_4Label.Text = 'KD';

            % Create KDEditField_4
            app.KDEditField_4 = uieditfield(app.UIFigure, 'numeric');
            app.KDEditField_4.ValueDisplayFormat = '%.4f';
            app.KDEditField_4.Position = [256 67 60 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = DialogBox4exported(varargin)

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The liquid weight controller parameters dialogue box

```
classdef DialogBox5exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure          matlab.ui.Figure
        KDEditField       matlab.ui.control.NumericEditField
        KDEditFieldLabel  matlab.ui.control.Label
        KIEditField       matlab.ui.control.NumericEditField
        KIEditFieldLabel  matlab.ui.control.Label
        ParametersoftheLiquidWeightControllerLabel  matlab.ui.control.Label
        KPEditField       matlab.ui.control.NumericEditField
        KPEditFieldLabel  matlab.ui.control.Label
        ConfirmButton     matlab.ui.control.Button
    end


    properties (Access = public)
        CallingApp   % Main app object
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app, mainapp)
            % Store main app in property for CloseRequestFcn to use
            app.CallingApp = mainapp;

            % Update UI with input values
            app.KPEditField.Value = app.CallingApp.KP_feed;
            app.KIEditField.Value = app.CallingApp.KI_feed;
            app.KDEditField.Value = app.CallingApp.KD_feed;

            % Define window name
            app.UIFigure.Name = "Liquid Weight Controller Parameters";
        end

        % Close request function: UIFigure
        function UIFigureCloseRequest(app, event)
            % Enable the Plot Opions button in main app
            app.CallingApp.LWParametersButton.Enable = 'on';

            % Delete the dialog box
            delete(app)
        end

        % Button pushed function: ConfirmButton
        function ConfirmButtonPushed(app, event)
            % Store inputs as properties of mainapp
            app.CallingApp.KP_feed = app.KPEditField.Value;
            app.CallingApp.KI_feed = app.KIEditField.Value;
            app.CallingApp.KD_feed = app.KDEditField.Value;

            % Call close request function
            UIFigureCloseRequest(app)
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [1 1 1];
            app.UIFigure.Position = [100 100 217 246];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.CloseRequestFcn = createCallbackFcn(app, @UIFigureCloseRequest, true);

            % Create ConfirmButton
            app.ConfirmButton = uibutton(app.UIFigure, 'push');
            app.ConfirmButton.ButtonPushedFcn = createCallbackFcn(app, @ConfirmButtonPushed, true);
            app.ConfirmButton.Position = [60 20 100 23];
            app.ConfirmButton.Text = 'Confirm';

            % Create KPEditFieldLabel
```

```
            app.KPEditFieldLabel = uilabel(app.UIFigure);
            app.KPEditFieldLabel.Position = [60 137 25 22];
            app.KPEditFieldLabel.Text = 'KP';

            % Create KPEditField
            app.KPEditField = uieditfield(app.UIFigure, 'numeric');
            app.KPEditField.ValueDisplayFormat = '%.4f';
            app.KPEditField.Position = [100 137 60 22];

            % Create ParametersoftheLiquidWeightControllerLabel
            app.ParametersoftheLiquidWeightControllerLabel = uilabel(app.UIFigure);
            app.ParametersoftheLiquidWeightControllerLabel.HorizontalAlignment = 'center';
            app.ParametersoftheLiquidWeightControllerLabel.WordWrap = 'on';
            app.ParametersoftheLiquidWeightControllerLabel.FontSize = 14;
            app.ParametersoftheLiquidWeightControllerLabel.FontWeight = 'bold';
            app.ParametersoftheLiquidWeightControllerLabel.Position = [15 190 190 34];
            app.ParametersoftheLiquidWeightControllerLabel.Text = {'Parameters of the'; 'Liquid Weight Controller'};

            % Create KIEditFieldLabel
            app.KIEditFieldLabel = uilabel(app.UIFigure);
            app.KIEditFieldLabel.Position = [60 103 25 22];
            app.KIEditFieldLabel.Text = 'KI';

            % Create KIEditField
            app.KIEditField = uieditfield(app.UIFigure, 'numeric');
            app.KIEditField.ValueDisplayFormat = '%.4f';
            app.KIEditField.Position = [100 103 60 22];

            % Create KDEditFieldLabel
            app.KDEditFieldLabel = uilabel(app.UIFigure);
            app.KDEditFieldLabel.Position = [60 70 25 22];
            app.KDEditFieldLabel.Text = 'KD';

            % Create KDEditField
            app.KDEditField = uieditfield(app.UIFigure, 'numeric');
            app.KDEditField.ValueDisplayFormat = '%.4f';
            app.KDEditField.Position = [100 70 60 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = DialogBox5exported(varargin)

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @(app)startupFcn(app, varargin{:}))

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

The liquid volume ODE system

```
function [dyV] = ODE_Volume(FR,FH,FT1,FT2,t,yV)

% Liquid volume balance [l/h]
dyV(1) = FR+FT1+FT2-FH;

% Substrate reservoir balance [l/h]
dyV(2) = -FR;

% pH-base reservoir balance [l/h]
dyV(3) = -FT2;

% pH-acid reservoir balance [l/h]
dyV(4) = -FT1;

dyV = dyV';
end
```

The mass balance and temperature ODE system

```
function [dy] = ODE_Luttmann_complete(qXpX,Din,cS1R,qS1pX,cS2R,qS2pX,CPRtot,qPpX,MP,cOT1,cOT2,cOR,OTR,OUR,cOL100,TMpO2,DT1,
      CAcT1tot,qAcpX,MAc,DT2,CAlT2tot,qAlpX,AlTR,MAl,DR,CCRtot,CCT1tot,CCT2tot,CTR,CER,MCO2,pHL,TMpH,TMxO2,xOG,TMxCO2,xCG,
      lamdaF,qhpX,lamdaAF,AAFin,tauD,DD,thetaTc,tauDL,thetaU,tauDU,tauL,tauLD,tauLU,QdotM,QdotSt,CL,t,y)
% ADD HARVEST TO BALANCE LATER

% Cell concentration balance
dy(1) = (qXpX-Din)*y(1);

% Glucose concentration balance
dy(2) = DR*cS1R-Din*y(2)-qS1pX*y(1);

% Glycerol concentration balance
dy(3) = DR*cS2R-Din*y(3)-qS2pX*y(1);

% Product concentration balance
dy(4) = DR*CPRtot-Din*y(4)+qPpX*y(1)/MP;

% O2 concentration balance
dy(5) = DR*cOR+DT1*cOT1+DT2*cOT2-Din*y(5)+OTR-OUR;

% pO2 balance
dy(6) = ((y(5)/cOL100)*100-y(6))/TMpO2;

% Buffer acid balance [mol/(l*h)]
dy(7) = -Din*y(7);

% Buffer base balance [mol/(l*h)]
dy(8) = -Din*y(8);

% Titration acid balance [mol/(l*h)]
dy(9) = DT1*CAcT1tot-Din*y(9)-qAcpX*y(1)/MAc;

% Ammonia balance [mol/(l*h)]
dy(10) = DT2*CAlT2tot-Din*y(10)-(qAlpX*y(1)+AlTR)/MAl;

% Dissolved CO2 balance [mol/(l*h)]
dy(11) = DR*CCRtot+DT1*CCT1tot+DT2*CCT2tot-Din*y(11)+(CTR+CER)/MCO2;

% pH-calculation with measurement dynamic [-]
dy(12) = (pHL-y(12))/TMpH;

% O2-mole fraction in offgas with measurement dynamic [-]
dy(13) = (xOG*100-y(13))/TMxO2;

% CO2-mole fraction in offgas with measurement dynamic [-]
dy(14) = (xCG*100-y(14))/TMxCO2;

% Relative foam height [1/h]
dy(15) = lamdaF*y(15)+qhpX*y(1);

% Anti foam activity [1/h]
dy(16) = lamdaAF*y(16)+AAFin;

% Temperature in double jacket [°C/h]
dy(17) = -y(17)/tauD+DD*thetaTc+y(18)/tauDL+thetaU/tauDU;

% Temperature in liquid phase of reactor [°C/h]
dy(18) = -y(18)/tauL+y(17)/tauLD+thetaU/tauLU+(QdotM+QdotSt)/CL;

dy = dy';
end
```

Declaration of Originality

I, Lena Sophia Kaletsch, hereby confirm that I have written the accompanying thesis titled “Development of an interactive *Escherichia coli* fed-batch fermentation simulation” myself, without contributions from any sources other than those cited in the text and acknowledgments. This applies also to all graphics and images included in the thesis.

I have not submitted or published this thesis elsewhere.

. . . .　　　　. [redacted]

Place, Date　　　　Signature