Digital Twin Tutorial: The Incubator Case Study

Cláudio Gomes[1][0000−0003−2692−9742], Morten Haahr Kristensen[1][0009−0008−8467−7567], Mikkel Schmidt Andersen[1][0009−0006−8678−1726], Prasad Talasila[1][10000−0002−8973−2640] Hao Feng[2][0000−0002−8272−2343], Thomas Wright[1], and Peter Gorm Larsen[1][0000−0002−4589−1500]

[1] Aarhus University, DK - 8200 Aarhus, Denmark
[2] Huawei, 518129 Shenzhen, China
{claudio.gomes,mhk,msa,prasad.talasila,thomas.wright,pgl}@ece.au.dk
https://international.au.dk/

Abstract. This paper presents a comprehensive Digital Twin (DT) tutorial using the case study of an incubator system designed for tempeh fermentation (a traditional Indonesian soy product). The study explores the development, calibration, and validation of models based on thermodynamics, offering insight into how DTs can enhance control and monitoring. Key DT services such as monitoring, decision support, and reconfiguration are implemented and evaluated. The tutorial demonstrates the potential of DTs in optimizing real-time systems while ensuring operational consistency.

Keywords: digital twin · simulation · optimization.

1 Introduction

1.1 Overview of Digital Twins (DTs)

The DT concept is gaining widespread attention due to its potential to leverage computing power for data analysis, event prediction, and scenario exploration without affecting the actual system. As interest in DTs grows across conferences, journals, and various industries, nations and businesses are increasingly investing in DTs for a wide range of applications, from microscopic systems to global environments. However, despite its promising potential, the methods and tools for developing DTs are still in the early stages, and the concept itself is not yet fully understood.

This tutorial aims to provide a comprehensive introduction to concept of DTs, focusing on the development and application of a DT in an incubator system. The tutorial synthesizes a decade of research in topics that fall under the umbrella of DTs, such as modelling and simulation, control, a

nomaly detection, and the internet of things. Some of the materials used in this tutorial have been simplified from the material presented in previous publications [19,17,18,36,45,16,5] and book [21].

1.2 Digital Twin Definition(s)

We adopt the definition from Fuller et al. [22]:

> . . . [A DT is] when the data flows between an existing physical object and a digital object [. . .] A change made to the physical object automatically leads to a change in the digital object and vice versa.

Note the contrast to an often cited definition of DT: A DT is a digital representation of a physical object or system that serves as the real-time digital counterpart of a physical object or system. As later described in section 2, this definition is closer to the definition of model. If the reader wishes to think of DT as representations of the physical system, then she/he may read this tutorial as a guide to applications of models (DT). We justify our choice of nomenclature in section 2.

The definition presented in Fuller remains too abstract to structure our presentation. We therefore propose the following breakdown of the definition, that is inline with the various surveys on the topic [29,48,32,23,6]. We focus on the critical services offered by a DT. A DT is a software support system that provides the following services, summarized in fig. 1:

Visualization Real-time visualization of system states and their history.
Monitoring Continuous monitoring of system performance, safety, or other properties
Predictive Maintenance Predicting when maintenance is required to prevent system failures.
Fault Diagnosis Identifying and diagnosing faults or unusual behavior.
Decision Making Simulating potential scenarios to predict outcomes.
Reconfiguration Enabling the Physical Twin (PT) to adapt to changes in their environment or operational conditions.

We consider these services to be building blocks of a DT. As we illustrate later they are combined to add value to a PT, but the PT should still operate without the support of a DT. This clearly distinguishes a DT from a control system. In this tutorial we will cover the underlined services above.

1.3 Our Experience

The authors are part of a research lab that have been working on the development of different services for DTs for well over a decade. In the following are some examples of Cyber-Physical System (CPS) for which one or more DT services have been produced.

Universal Robots Manipulator. In [33], the authors have built a tool that enables users to efficiently model robot dynamics by separating rigid-body dynamics from joint dynamics, allowing for rapid recalibration (an important feature in DTs to keep the model synchronized with the PT) without recompiling the entire

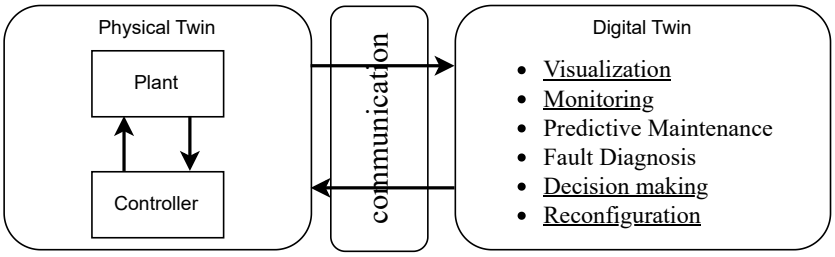

Fig. 1. Overview of services provided by a DT. The arrows represent the data flow a DT. Underlined are the services we will cover in this tutorial.

model. The tool, called Aarhus University Robotics Toolbox (AURT) supports various joint dynamics models and uses a Weighted Least Squares (WLS) method for calibration. The authors demonstrate the effectiveness of AURT through case studies involving a Universal Robots UR5e robot, showing that AURT can produce high-fidelity dynamic models with low prediction errors. In addition, 3D and 2D visualizations have been produced to visualize the robot's motion and the model's prediction, as shown in fig. 2. Although monitoring and predictive maintenance were not implemented in this work, its realization is trivial since the user can just compare the predicted current with the actual current to detect problems, such as a collision (which would show up as a sudden spike in the current).

Agricultural Robot Miniature. Woodcock et al. [45] provide a detailed case study involving an agricultural vehicle called Desktop Robotti [14], which serves as a prime example of the need for environment-aware DT in complex, uncertain environments. The case study illustrates how the model of the vehicle is used to monitor and predict its behavior under various operating conditions, accounting for factors such as noise and environmental changes. It enables the identification of safety violations by comparing real-time data with the expected performance based on statistical noise models. The case study also demonstrates how the system can perform "what-if" analyses for decision support, to determine the impact of different noise sources on the vehicle's safety and decide on the best system configuration to minimize risks.

And others. Our lab has also worked on other case studies such as the F1-tenth [1], a pancreas preservation device [7], as well as an incubator system intended for academic purposes. The incubator is the running example for this tutorial and is elaborated upon in section 4.

1.4 Purpose and Structure of the Tutorial

This tutorial aims to provide a comprehensive introduction to DT technology, with a focus on the development and application of a DT in an incubator system,

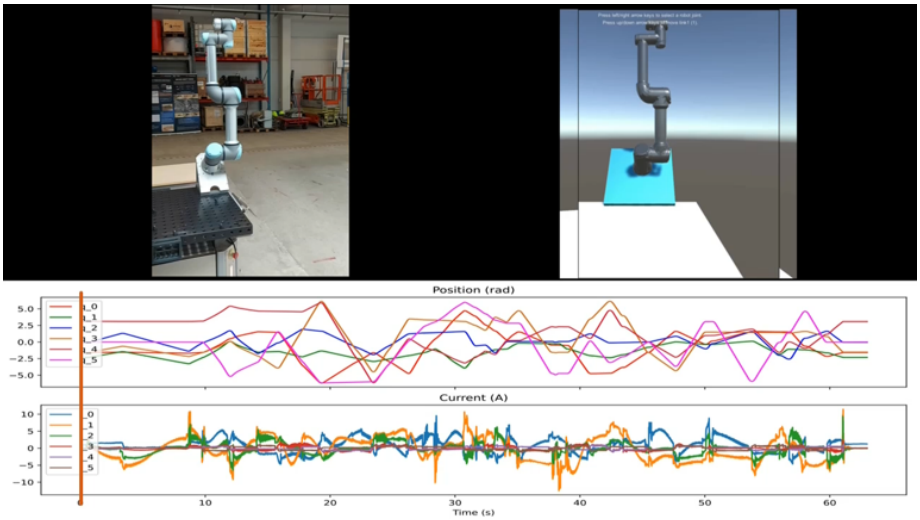

Fig. 2. Visualizations of the Universal Robots UR5e robot. On the left is a video stream of the robot. On the right is 3D visualization based on the streamed robot joint angles, and at the bottom is a plot of the positions of the joint angles as well as the predicted (by the model) current drawn at each joint overlaid on the actual current. The vertical red line indicates the time since the operation started.

and covering some of the services introduced in fig. 1. The incubator system was chosen for its simplicity and accessibility of the physics, as well as clear need for a DT. The tutorial will cover foundational concepts, the building blocks of a DT, and a detailed case study of the incubator system.

Structure. The material, most of it published elsewhere, is presented in a top-down matter, with section 2 describing the foundational concepts in digital twinning, and section 3 giving a catalog of the typical DT services. Due to their breadth, sections 2 and 3 do not delve into details, so we encourage the reader to jump to section 4 whenever a concept is not clear, as the concept is illustrated in the incubator system.

2 Building Blocks of a Digital Twin

In this section we introduce some of the foundational concepts as well as a terminology used in this tutorial. For more details about the models and their qualities we refer the reader to [36].

2.1 Cyber-Physical Systems (CPS)

A *system* is defined as "a combination of interacting elements organized to achieve one or more stated purposes" [28]. In CPSs, this may include physical

devices, natural phenomena, or information about the CPS. The *environment* refers to elements outside the system. The *system boundary* between the system and environment is determined by the system's purpose [47].

A system's *purpose* relates to its *behavior*, i.e., its interactions with the environment. The system *context* denotes all possible environment-system interactions. System *properties*, derived from functional and non-functional requirements, encode the system's purpose and should be *satisfiable*, meaning it should be possible to check if the system satisfies a property, accounting for uncertainty.

Let S represent the system, and P_S the set of properties to be satisfied in context C_S (where C_S denotes the set of all possible interactions from the environment to the system). Let $[\![S]\!]_{C_S}$ represent the system behavior under C_S as a set of traces. We write $[\![S]\!]_{C_S} \vDash p$ to indicate that the behavior satisfies property $p \in P_S$.

CPS are systems that integrate computational and physical processes. Examples of CPS include smart grids, autonomous vehicles, and industrial automation systems. The interaction between these processes is crucial for the overall system's functionality and performance. In the context of this tutorial the PT is considered a CPS.

2.2 Data Collection and Integration

Data collection forms the backbone of a DT, and it is an activity that spans multiple areas of science and engineering, relying on sensors and communication systems to capture real-time information from the PT. This data is seamlessly integrated into the digital model, where it supports analysis and informed decision-making. In the following we introduce the key areas in roughly the order of information flow, enumerating the main challenges and techniques in each.

Sensing involves gathering system and environment data through devices attached to the PT such as temperature sensors. The key areas are introduced in the following. A key aspect of sensing is sampling, where signals are measured at regular intervals. According to the Nyquist-Shannon theorem, a signal must be sampled at least twice its highest frequency to prevent information loss. However, challenges such as non-uniform sampling, noise, and quantisation — the conversion of continuous signals into discrete values — can affect signal quality. To address these, techniques like non-uniform sampling and dynamic quantisation focus on capturing significant changes in the signal, improving the efficiency of data collection.

Quantisation errors occur when continuous signals are converted into discrete signals, often resulting in minor inaccuracies. Dynamic range compression helps mitigate these errors, especially for signals with a wide amplitude range. Sensor data is also impacted by various types of noise, and flicker noise (which varies with frequency). Compensation methods are necessary to account for these disturbances and ensure accurate sensor readings. After sensing comes the transmission of information using network communication protocols.

Stable *network communication* is crucial for DTs, but it can be disrupted by delays and data loss. Communication protocols like TCP ensure reliable data

transmission by prioritizing accuracy, while UDP sacrifices some reliability for faster performance. To counteract network degradation and connection drops, techniques such as synchronizing the clocks of the PT and DT and handling outdated data can maintain system integrity.

Data compression is essential for managing network traffic and storage requirements. Lossless compression preserves the integrity of the data, while lossy compression trades some accuracy for a higher compression ratio. In time-series data, effective compression methods are critical for reducing traffic without compromising key information.

Several *messaging protocols* support communication in DT ecosystems, including ZeroMQ, ROS, Apache Kafka, and RabbitMQ. These protocols use patterns like publish-subscribe and request-reply to ensure real-time data flow across distributed systems.

Finally, *Time-Series Databases* like InfluxDB[3] and ClickHouse[4] store the vast amounts of data generated by sensors in the PT. These databases are optimized for real-time processing and long-term storage, supporting critical DT services like monitoring and predictive modeling. Built-in compression techniques further reduce memory usage, enabling scalable and efficient data management.

Setting up a data collection is first step towards obtaining a valid model of the PT system, discussed next.

2.3 Modelling Systems

DTs use models of PTs that they represent to provide some of the services identified in subsection 1.2, covering aspects like energy consumption or functional performance. The choice of models is based on trade-offs between precision, accuracy, and computational costs.

What to Model A *model*, following [31,41], is based on a system, reflects a subset of its properties, and serves a predefined purpose in a specific context. Experiments on models should faithfully reflect property satisfaction on the real system, as discussed in [13], and summarized in the commuting diagram of fig. 3.

The following details the main qualities of the model, in an informal manner, to help the reader see the nuances faced by modelers. The scope of modelling is too broad for an attempt at a more rigorous formalization without focusing on a specific modelling formalism such as state machines, differential equations, Markov models, etc. The reader is encouraged to read [36] for more details of these qualities.

A model is *relevant* if its context encompasses the system's context, ensuring that the model can represent the system in that specific context. Relevance ensures that the model operates effectively within the intended scope and environment of the real system.

[3] `https://www.influxdata.com/`
[4] `https://clickhouse.com//`

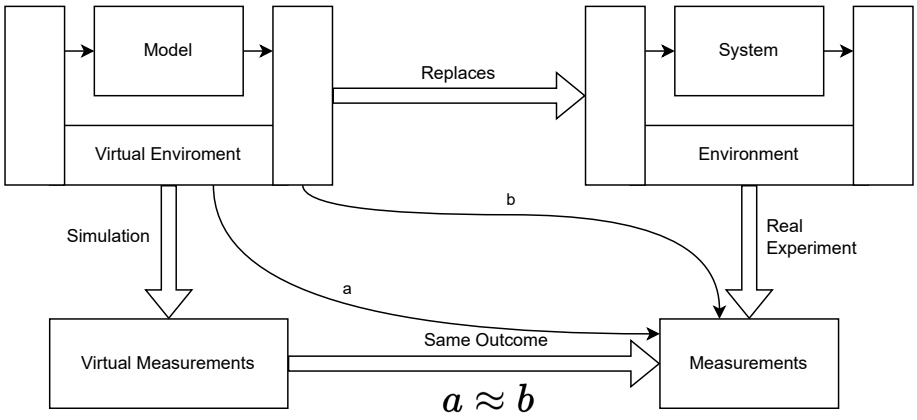

Fig. 3. Commuting diagram illustrating the relationship between a real system and its corresponding valid model.

A model is *verifiable* if it is possible to determine whether the model satisfies each property of interest. Verifiability is critical for ensuring that the model behaves as intended and adheres to its design requirements.

Substitutability holds when the behavior of the model aligns with that of the system for all relevant properties. This quality ensures that the model can replace the system in experiments or analyses, producing equivalent outcomes.

Fidelity measures how closely the model's behavior matches that of the system under comparable conditions. High fidelity indicates that the model accurately reflects the system's properties and behavior, which is particularly important in safety-critical or precision-oriented applications.

Note that *verifiability* is essential; if models cannot be checked for properties, they cannot provide *substitutable* results. Next, *substitutability* is important to verify the satisfaction of properties, though quick or approximate checks may be necessary for large possibility spaces. *Relevance* follows, ensuring models remain applicable in a dynamic system. Non-relevant models increase the risk of errors, especially in safety-critical environments. Finally, *fidelity* should be considered once other qualities are satisfied, ensuring the system's properties are reflected with sufficient accuracy. Low fidelity may be acceptable in non-critical applications to reduce resource use.

If the above properties are satisfied, the model is considered valid. The relationship between the model and the system can be summarized by a conceptual framework. This framework illustrates the relationship between a real system and its corresponding valid model, highlighting how virtual experiments can replace real-world experiments, provided the model is valid. It shows a system interacting with its environment, where measurements are taken and then replicated in a virtual environment using a model. The model and virtual environment are

used for simulations, with the goal that the outcomes of these simulations should match those of the real-world experiments, allowing the virtual experiment to serve as an accurate substitute.

In terms of object of study, models can focus on all aspects of CPS, summarized in fig. 4, and detailed as follows:

Network Represents the communication infrastructure that allows different components of the system to exchange data and information.

Platform Refers to the underlying hardware or software environment that supports the operation of CPS components.

Actuators Physical devices that interact with the environment by converting electrical signals into mechanical actions, such as motors, valves, etc.

Sensors Devices that monitor physical properties like temperature, pressure, or motion and provide data to the system.

Controller The computational unit responsible for decision-making and sending commands to actuators based on the data received from sensors.

Plant Represents the physical process or system being controlled and monitored, such as a manufacturing machine or an energy grid.

Cyber Refers to the computational components of the system, encompassing the software, algorithms, and networking aspects that process data and control actions.

Physical Refers to the real-world physical components that interact with the environment, such as sensors, actuators, and the plant.

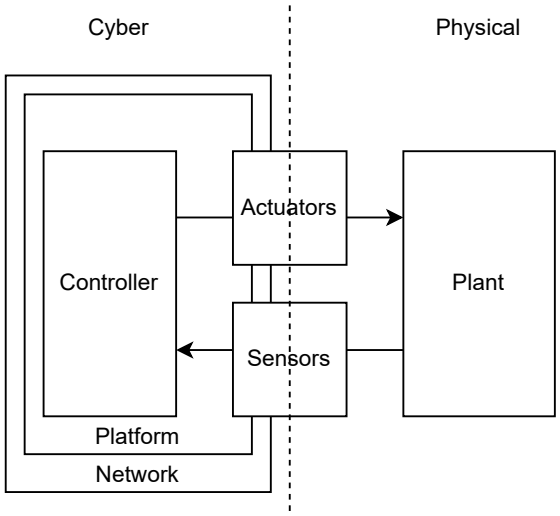

Fig. 4. Aspects of CPSs that can be the focus of models.

How to Model The DT model set is heterogeneous, often developed by specialists from different disciplines, using diverse formalisms, including:

Physics-based models: Derived from first principles, these models use algebraic and differential equations to describe physical processes like heat transfer. They are commonly represented using ordinary or partial differential equations. Example models such as Ordinary Differential Equation (ODE)s, describe physical conservation laws. These models can be simulated using numerical methods like Euler's method. Partial Differential Equations (PDEs) provide more complex models for spatial heat distribution, often requiring spatial discretization methods like finite element or finite volume methods.

Data-driven models: These models predict system behavior based on input-output data, using techniques like machine learning, deep learning, and physics-informed machine learning. They are versatile but rely on high-quality data. Machine learning, especially deep learning, has become a popular approach. These models are trained using methods like supervised, unsupervised, or reinforcement, learning.

Models for computer-based systems: These describe hardware and software components using formalisms like finite state machines or hybrid automata. Finite State Machines (FSM) can represent the systems states and transitions with conditions attached to represent when the current state of the system changes. Hybrid automata extend FSMs to include continuous dynamics, allowing for more complex system behavior. These models can be used to describe the behavior of embedded systems, control systems, and software systems.

Models are characterized by state form (numerical or symbolic), state evolution (continuous or discrete time), and model behavior (deterministic or stochastic). These criteria help in selecting appropriate formalisms for DTs.

Simulation is the process of producing the behavior of a model. Simulations can be classified into different types such as continuous, discrete-event, or hybrid, depending on the nature of the system and the way time and events are modeled. We revisit this topic in subsection 3.3.

We refer the reader to [21,9,44] for excellent introductions to the topic of modelling, and to [10] for a general treatment of the topic of simulation.

2.4 Calibration

Calibration involves finding parameter values that align model behavior with real-world data. For instance, in a simple linear model $y = ax + b$, parameters a and b are estimated to match experimental observations. In complex non-linear models, optimization techniques like gradient descent are necessary to minimize errors and potentially avoid local minima.

For linear algebraic models, the least squares method is used. It minimizes the sum of squared residuals between the observed and predicted values. For example, given a system $y = ax_1 + bx_2$, parameters a and b can be estimated from experimental data through matrix operations.

Non-linear models require iterative methods like gradient descent, which minimizes the residual error between the predicted and observed data. In complex surfaces with multiple minima, heuristic methods like genetic algorithms or Gauss-Newton methods can be employed to avoid local minima and find optimal solutions.

In practice, measurements are often noisy, which complicates the calibration process. Least squares can be adapted to handle noise, but more advanced methods like gradient descent become essential for non-linear models. The challenge lies in finding parameters that balance fitting noisy data without overfitting. Underfitting occurs when models are too simple, while overfitting happens when models are too complex and fail to generalize. Techniques like regularization and careful experiment design can help mitigate these issues.

2.5 Validation of Models

Validation is the process of ensuring that the model is valid by comparing its predictions with the real world data. Figure 5 puts together the concepts of modelling, simulation, and calibration. In the figure we show calibration as a synonym with validation because it is after the choice of suitable parameters that the model qualities can be evaluated.

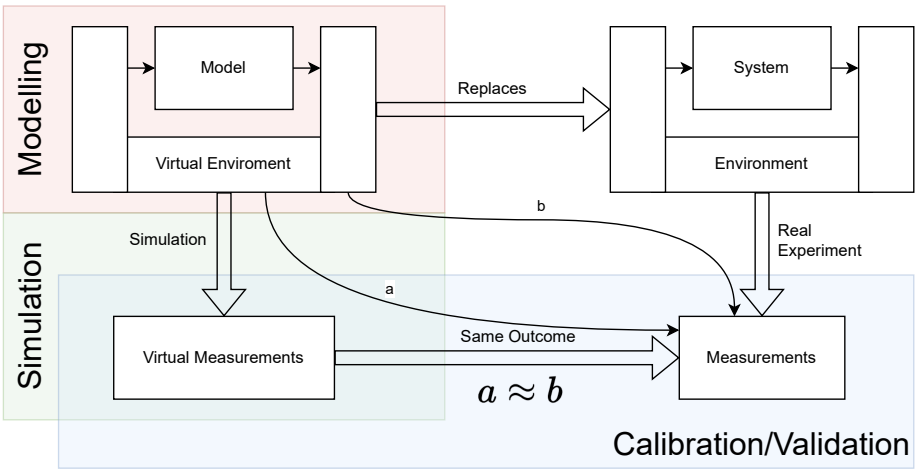

Fig. 5. Commuting diagram illustrating the relationship between a real system and its corresponding valid model.

2.6 DT vs. Digital Shadow

A DT involves bi-directional data flow between the PT and its digital counterpart, allowing for real-time feedback and control. In contrast, a Digital Shadow

(DS) only allows data to flow from the PT to the digital model, without direct feedback capabilities. We like making this distinction because the set of challenges tackled by a DT are different from those faced by a DS. Note that a DT can start as a digital shadow where it can bring the most value and insight, and then evolve to a full DT. Figure 6 illustrates the different services provided by a DS and a DT. Note how the DT can leverage the services provided by the DS.

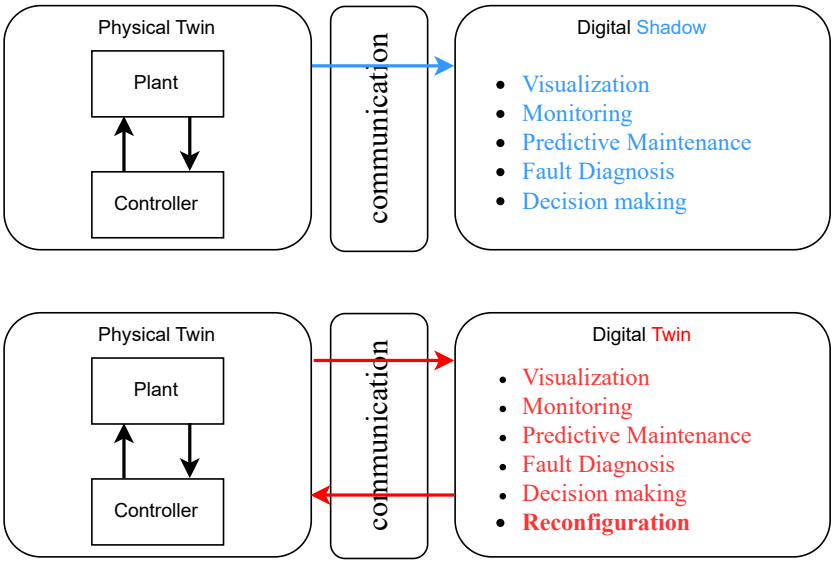

Fig. 6. Communications inside a DS and DT. The blue and red arrows represent the data flow of a DS and a DT respectively. A DT includes the features of a DS and has unique features such as reconfiguration.

3 Digital Twin Services

We now provide an overview of the services that were summarized in fig. 1.

3.1 Visualization

Visualization translates complex information into figures, graphs, 3D models, or even 4D interactive interfaces, enabling users to grasp the current state of the PT. It enhances system monitoring, supports decision-making, and helps identify patterns or risks.

Five key visualization types are commonly used:

1. **Data graphs and charts:** Display real-time data trends, such as system metrics.

2. **Schematic diagrams:** Show system structure and component relationships, often as 2D floor plans.
3. **3D models:** Visualize physical characteristics of the PT, useful for systems requiring detailed examination.
4. **Augmented Reality (AR):** Overlay digital data onto the physical world, enhancing user interaction with the PT.
5. **Mixed and Virtual Reality:** Combine AR/Virtual Reality with interactive controls for a full 4D experience, ideal for precision tasks.

Dashboards are the preferred user interface to offer high-level views of system performance using graphs and charts. Popular frameworks include:

InfluxDB: A cloud-based time-series database with customizable dashboards for real-time analytics.
JMobile: A framework for IoT systems, supporting process management and data visualization.
Grafana: A multi-platform visualizer that integrates with various data sources, offering interactive web-based interfaces.

For 4D visualization, the following tools are commonly used:

Unreal Engine: A 4D graphics engine for resource-intensive applications in industries like simulation and architecture.
Unity: A 4D engine widely used in DTs for mobile apps, automated vehicles, and robotic systems.
Godot: A lightweight, open-source game engine for basic 4D visualizations.
iTwin: A platform for infrastructure DTs, supporting 4D visualisations and IoT data streams.

3.2 Monitoring

A DT enhances a PT by integrating models of expected behavior with real-time data from the PT, identifying anomalies or deviations from normal operations [2,3]. This monitoring function supports stakeholders in making informed decisions [2]. Monitoring can be done *offline*, analyzing stored data for debugging, or *online*, tracking properties during operation and triggering prevention or recovery actions. Here we focus on online monitoring.

Monitoring methods can be broadly categorized as *model-based* and *data-driven*. Model-based monitoring checks system behavior against defined properties, using temporal logics such as Linear Temporal Logic (LTL [37]) or Signal Temporal Logic (STL [34]). Temporal logics enable the expression of properties over system traces. LTL is useful for expressing safety and liveness properties in discrete time, while STL adds time constraints and continuous-time support, making it suitable for real-time systems. STL also provides quantitative outputs, useful for predicting how close the system is to violating a property. Data-driven monitors, by contrast, use machine learning models trained on historical data to detect anomalies in the PT.

Model-based monitoring techniques, such as Runtime Verification (RV), allow DTs to monitor system properties in real-time. These monitors can trigger compensating actions if a violation of safety properties is detected. Various tools are available for implementing RV, including Java PathExplorer, MonPoly, and RTAMT [27,4,35].

Data-driven anomaly detection relies on real-time sensor data, analyzed using machine learning models trained on historical data. This approach is often framed as a one-class classification problem, learning what constitutes normal operation and identifying deviations as anomalies. Various models can be employed, such as autoencoders or Recurrent Neural Networks (RNNs), which are trained on normal data and predict or reconstruct time-series measurements.

State Estimation An important technique in monitoring is state estimation, which effectively combines real-time sensor data with a model of the PT to estimate the system's internal state. This is particularly useful when some states are not directly observable. Just like RV, state estimation can be done offline (e.g., in batches of data), or online. The technique is illustrated in fig. 7. In the figure, the plant represents the PT being monitored or controlled that is modelled by the model. The "plant" could be any PT like a machine, vehicle, or environmental process. The environment represents external factors or conditions that influence the plant. These could include external inputs like temperature, pressure, or other environmental variables affecting the plant's behavior. The model is the mathematical or computational model used to simulate or represent the behavior of the plant. The model relates the states of the plant with its output via equations, typically ordinary differential equations in the state space representation. The input refers to the control inputs provided to the plant. These inputs could be in the form of actuators controlling variables like speed, temperature, or position, and note that they are sent in parallel to the plant and the model. Sample refers to the data collected from the plant at discrete intervals, usually using sensors. These samples are taken into account by the state estimation algorithm. Note that the model does not have access to the samples directly. The samples are the observable or measurable outputs of the plant and are used by the state estimation algorithm to correct the model's predictions. The "states" represent the internal variables of the plant that are being estimated. Note that in the model, the states are fed back in each new timestep. The outputs of the state estimator refer to the final output after processing the model with the sample. In addition, the state estimator can often produce a measure of how good the model is performing in terms of predicting accurately the plant's behavior.

For offline state estimation, where samples can be accessed in batch, the problem being solved can be formulated as follows: find states $x_0, x_1, \ldots, x_i$ such that

$$\left\| [\bar{y}_0, \bar{y}_1, \ldots, \bar{y}_i]^{\mathrm{T}} - [y_0, y_1, \ldots, y_i]^{\mathrm{T}} \right\| \approx 0$$

where the outputs are the ones produced by the state estimation algorithm in fig. 7.

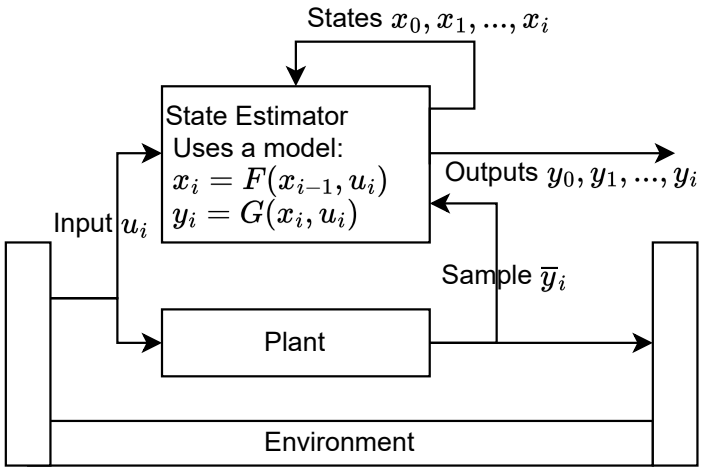

Fig. 7. State estimation illustration.

In the online case, the problem is more challenging as the samples are not available in batch. The state estimation algorithm must estimate the states based on the model and the samples available at each time step i. The problem can be formulated as follows: at each new sample, find state x_i such that

$$\|\bar{y}_i - y_i\| \approx 0$$

The Kalman filter is a widely used state estimator that combines noisy sensor data with a dynamic model of the system to estimate the true state. Extended Kalman Filter (EKF) and Unscented Kalman filters (UKF) are used when the system is nonlinear. Particle filters are another option for nonlinear systems, using a set of particles to represent the system's state distribution. We refer the reader to [40] for a comprehensive introduction to state estimation techniques, and to [18] for a detailed derivation of the Kalman filter.

3.3 Decision Support

In the context of this tutorial, decision support represents the DT service that allows the user to conduct various simulation experiments on the historical data, on future data, and solve optimization problems based on simulations.

We start with a primer in simulation of ordinary differential equations (as these are used later in section 4).

Simulation of Ordinary Differential Equations To simulate the ODE, values for the parameters and its initial states must be found. The ODE and initial values form an Initial Value Problem (IVP), solved using *numerical time*

integration. A general ODE of this type is:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{p}) \ , \ \mathbf{x}(0) = \mathbf{x}_0 \tag{1}$$

where $\mathbf{x}$ represents the state vector, $\mathbf{u}$ the input vector, and $\mathbf{p}$ the model parameters.

Numerical integration schemes estimate the state vector at discrete intervals, with the Euler method being the simplest:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p})\Delta t \tag{2}$$

More advanced methods use multiple steps or evaluations within each time step to improve accuracy. The simulation output tracks the system's state over time. Libraries like *SciPy* in Python offer various numerical integration schemes for solving ODEs. Interested readers can refer to [8,26] for more on numerical time integration.

What-if Simulation A what-if simulation is a data- and computation-intensive approach used to evaluate the behavior of a PT under different scenarios and hypotheses [38]. If unwanted behaviors are detected in the PT, a what-if simulation enables operators or the DT itself to simulate potential interventions, compare outcomes, and make better-informed decisions. For effective use, the simulations must run faster than real-time, and combining multiple simulation components through "co-simulation" may be required when multiple models are involved [25]. In a DT context, what-if simulations can leverage historical data from the other DT services, such as the state estimation, and feed results into other services for further optimization.

Design Space Exploration Design Space Exploration (DSE) is a form of optimization, used to evaluate different design alternatives for a system [15], aiming to meet performance goals. Each alternative consists of specific parameter values, and (co-)simulations are performed for each one. Genetic algorithms can help choose which simulations to run when the design space is too large [39]. DSE often involves balancing conflicting objectives, such as speed, accuracy, and energy consumption. A ranking function or the Pareto Optimal front [12] may be used to select the best design.

While DSE and what-if simulations both explore alternatives, they differ in context: DSE is used during design stages to optimize the system, whereas what-if simulations are applied during operational stages to explore the effects of different conditions on the deployed system.

Decision support services are intimately related to Reconfiguration, since the outcome of a decision support service may lead to a reconfiguration of the system, discussed next.

3.4 Reconfiguration

A critical feature of a DT is its ability to autonomously manage the PT, such as addressing anomalies, ensuring safety, or reconfiguring the PT to adapt to changes in its environment. This process may involve re-optimizing parameters and updating models, and may use data from monitoring and decision support services to inform the reconfiguration. It can combine the results from the other services, e.g., anomaly detection, what-if, or DSE in order to optimize the system. A popular architecture to represent reconfiguration is the MAPE-K loop.

MAPE-K Loop The Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) loop is a framework for self-adaptive systems that enables dynamic reconfiguration in response to changes or anomalies in the environment or the system itself [30]. It comprises four main phases—Monitor, Analyze, Plan, and Execute—that operate over a shared Knowledge base, as illustrated in fig. 8.

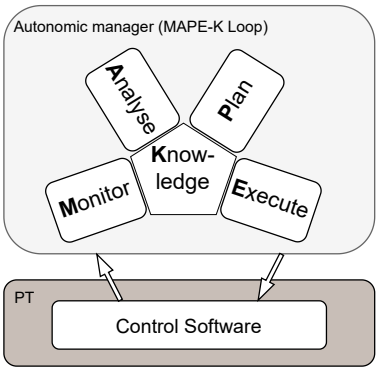

Fig. 8. The MAPE-K loop architecture [30].

Monitor In the Monitor phase, the system collects data from various sources, such as sensors or logs, to observe the current state of the PT. This data includes metrics and events that are relevant for detecting significant changes or anomalies in the system's operation.

Analyze The Analyze phase processes the collected data to identify patterns, trends, or anomalies. Techniques such as anomaly detection algorithms, statistical analysis, or machine learning models may be employed. The goal is to determine whether the current state deviates from expected behavior and requires adaptation.

Plan During the Plan phase, the system devises a strategy to address any identified issues or to optimize performance. This involves generating a set of actions or changes to the system's configuration that can mitigate anomalies or improve operation. The planning process may consider multiple alternatives and evaluate them based on predefined criteria or objectives.

Execute In the Execute phase, the planned actions are implemented to modify the system's behavior. This may involve reconfiguring components, updating parameters, or deploying new resources. The execution should be carried out in a controlled manner to ensure system stability and minimize disruption.

Knowledge The shared Knowledge component is a repository that stores information used by all phases of the loop. This includes models of the system, historical data, policies, and any other contextual information necessary for decision-making. The knowledge base enables consistency and coordination among the different phases.

4 Case Study: The Incubator System

4.1 Overview

The incubator system, illustrated in fig. 9, serves as a practical example of a CPS-based DT. It consists of a styrofoam box with a heatbed, fan, and temperature sensors, controlled by a Raspberry Pi, made to incubate tempeh. The reader is invited to see an online video presenting the setup[5].

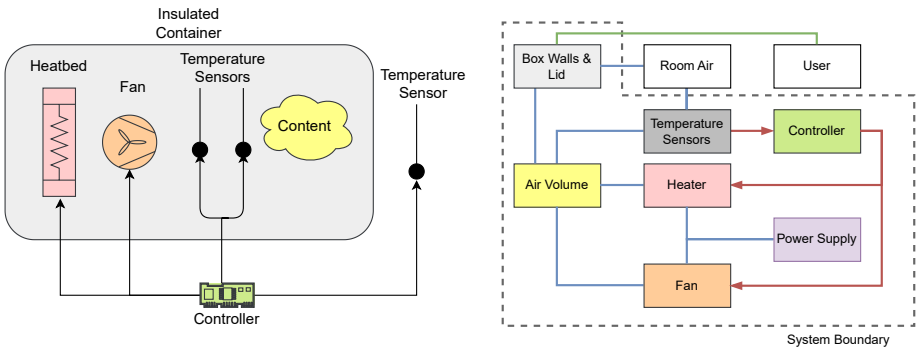

Fig. 9. Schematic overview of the incubator system and the role of the DT in tempeh production. The left diagram shows the physical setup, while the right diagram represents the interactions between various components, including digital information flows and user interactions. Reproduced from [21].

[5] `https://youtu.be/gG4za7iPY0I`

Tempeh: Tempeh is a traditional fermented food originating from Southeast Asia, especially known in Indonesia for its nutty flavor and firm texture. While traditionally made from soybeans, it can also be produced using various legumes such as black beans, chickpeas, and lentils. Each variation offers different flavors and nutritional profiles, making tempeh a versatile plant-based protein option [42].

The process of making tempeh, as shown in fig. 10, involves soaking and cooking the soybeans before inoculating them with the fungus *Rhizopus oligosporus*. The mixture is then fermented at a controlled temperature, usually between 24 to 48 hours, depending on the desired final product. During the fermentation, the fungus binds the beans into a cake-like structure, developing the firm texture and the nutty flavor associated with tempeh.

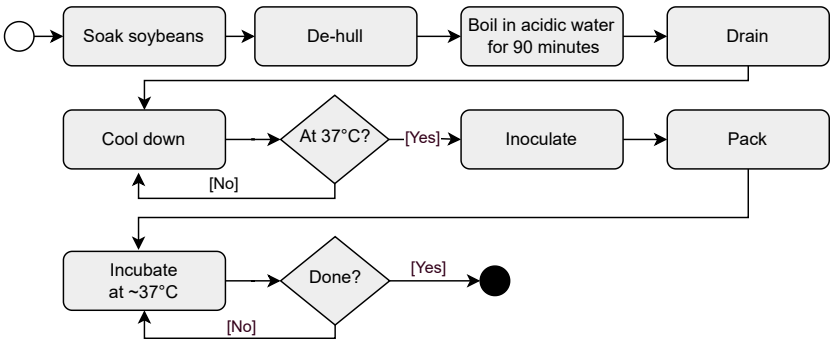

Fig. 10. Overview of the tempeh-making process. The incubation phase is critical to maintaining a temperature of 37.5°C for optimal fermentation. Reproduced from [21].

During the fermentation, the fungus initially absorbs heat from the environment. After around 20 hours, the fungi rapidly grow, producing their own heat, which can cause the internal temperature of the tempeh to rise above the surroundings by as much as 7°C. The tempeh is typically ready for consumption once it passes its temperature peak, as seen in fig. 11.

Role of DT: The DT serves as an extension to the physical incubator, offering these enhanced services through a connected device such as a smartphone. Although the incubator can function without the DT, users who opt to use the DT-enabled features will benefit from improved control, ensuring higher consistency and better quality of the final tempeh product.

Hardware: The incubator hardware includes a styrofoam box for insulation, a heat bed (RepRap PCB Heat Bed MK2a), a fan to distribute heat, and three DS18S20 temperature sensors (two inside and one outside). A Raspberry Pi

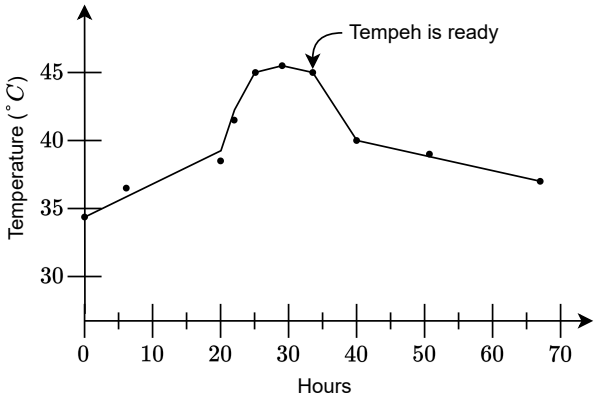

Fig. 11. Temperature changes during tempeh fermentation. Adapted from [42]. Variations in temperature and timings are common and depend on several factors such as environmental conditions and bean type.

manages the sensors and controls the heat bed and fan, connecting them to a higher voltage power supply using a set of relays on a printed circuit board. We refer the reader to the online repository with up-to-date documentation about all aspects of the hardware of the incubator [24].

Software: The incubator's control system, summarized in fig. 12 and implemented in Python, includes a communication server, a low-level driver, and a controller that manages the heatbed and fan, with a basic on-off control strategy.

Communication Server – RabbitMQ implementing the AMQP protocol is employed for managing communication and synchronisation, facilitating message exchange between the controller and low-level driver. It ensures seamless communication between components, which is essential for the DT process.

Low-level driver – Manages low-level communication with sensors and actuators. The low-level driver interacts with the PT, collecting data periodically and ensuring safe protocol execution. It reads the temperature data and checks for commands from the controller via the communication server.

Controller – The controller operates based on bang-bang control with a slight variation. It turns on the heatbed until the temperature reaches the setpoint, then waits to prevent overshoot. The controller parameters are: LL – Lower temperature limit; UL – Upper temperature limit; H – Heating duration; C – Waiting duration.

4.2 Digital Twin Architecture

The software architecture of the DT is an instance of the service oriented architecture pattern. Service-Oriented Architecture (SOA) is a design pattern where

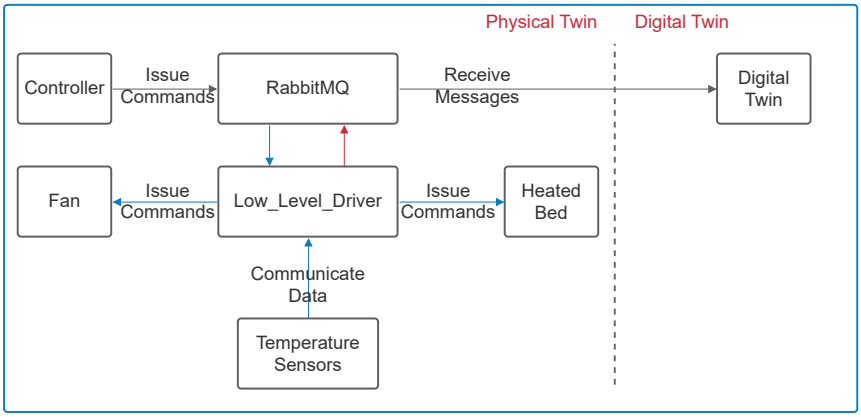

Fig. 12. Overview of communication in the Incubator. Arrows indicate the information flow. Reproduced from [20].

software components provide services to other components over a network. In SOA, these services are loosely coupled, reusable, and can be accessed independently, promoting flexibility and scalability. Each service performs a specific business function and can be accessed via standard communication protocols like RabbitMQ.

In this architecture, where even the controller is seen as a service, each DT service translates directly to a corresponding software process that is communicating with other services to accomplish a given task. The communication between services is accomplished via RabbitMQ messages where services publish and subscribe to messages with certain topics. This communication is very flexible enabling the following scenarios:

Point-to-Point One producer sends a message to a specific queue, and a single consumer retrieves it, ensuring one-to-one communication. It's useful for tasks like job processing.

Topic-Based Publish/Subscribe Messages are sent to an exchange with a routing key, and only consumers with matching routing keys (using wildcard patterns, e.g., `*.log`) receive the messages. This enables flexible one-to-many communication based on message topics.

Load Balancing This is particularly useful in scenarios where there are multiple consumers working on tasks from a shared queue. Multiple instances of the same DT service can started, and connected to the same queue. When multiple consumers are connected to a queue, RabbitMQ distributes messages evenly in a round-robin fashion to avoid overloading any single consumer. In the incubator DT this is used for services that provide decision support.

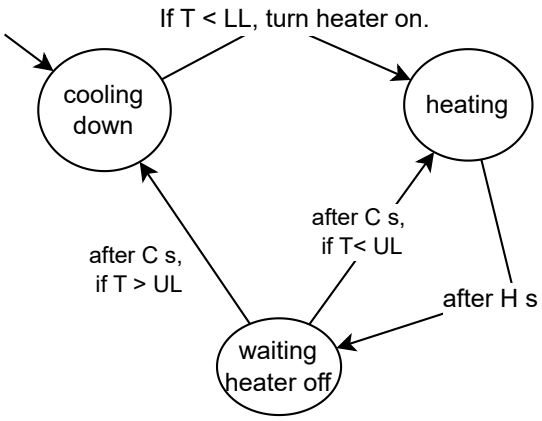

Fig. 13. Controller Statechart. Reproduced from [20].

Request-Reply This pattern involves sending a request message and awaiting a reply, enabling synchronous communication. A reply queue is used to send the response back to the original requester.

4.3 Data Collection

The data is collected by a Python process (a service) that subscribes to most messages exchanged between the controller and low level driver, and stores them in InfluxDB, a time-series database. The data stored includes all sensor and actuator data, as well as control parameters.

4.4 Modeling the Physical Twin

We first introduce the thermodynamics equations that were used to model the temperature evolution inside the incubator box.

Lumped Element Heat Transfer Modelling Here we introduce the fundamental concepts of the lumped element heat transfer model, providing enough information to ensure the manuscript is self-contained. For a comprehensive treatment of thermodynamics and calculus, readers are directed to sources such as [43,11].

Consider the warm air (e.g., 80°C) insulated inside a Styrofoam box, in a room that's at 20°C. Over time, the air cools down as heat transfers from the air to the surrounding room. The rate of heat transfer depends on two factors: the temperature difference between the air and the room and how good the box is as an insulator. The thicker and more insulating the box walls, the slower the rate of heat transfer. The heat transfer rate is governed by Fourier's Law of heat conduction:

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = -G_{br}(T_b - T_r), \tag{3}$$

where Q is the heat energy of the air volume, G_{br} is the thermal conductivity of the box, T_b and T_r are the air and room temperatures, respectively. The negative sign indicates heat flow from the air inside the box to the room. To keep the model simple, we assume the material properties, thickness, and area remain constant, consolidating them into the constant G_{br}.

Next, we relate the heat transfer to the box air's temperature change. The heat capacity C_b quantifies how much heat is needed to raise the box air's temperature. The heat transfer rate can be expressed as:

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = C_b \frac{\mathrm{d}T_b}{\mathrm{d}t}. \tag{4}$$

By combining this with Fourier's Law, we derive the following differential equation:

$$\frac{\mathrm{d}T_b}{\mathrm{d}t} = \frac{-G_{br}(T_b - T_{\mathrm{room}})}{C_b}. \tag{5}$$

Additional Factors: Electrical Heating. Now, suppose we heat the rod by passing an electrical current through it. The total heat transfer now includes both the heat loss to the room and the electrical power contribution, given by the product of voltage and current $V \cdot I$. This modifies the heat transfer equation to:

$$\frac{\mathrm{d}T_b}{\mathrm{d}t} = \frac{-G_{br}(T_b - T_{\mathrm{room}}) + V \cdot I}{C_b}. \tag{6}$$

Incubator Temperature Model To get the incubator temperature model, we apply the previous derivation steps twice: to the heat exchange between the heatbed resistance (piece of metal that's a heater, with it own volume and heat transfer rate) and the box air volume, and to the heat exchange between the box air volume and the room. We therefore obtain the following ODE:

$$C_h \frac{\mathrm{d}T_h}{\mathrm{d}t} = H_h \cdot P_h \qquad - G_{hb} \cdot (T_h - T_b) \tag{7a}$$

$$C_b \frac{\mathrm{d}T_b}{\mathrm{d}t} = G_{hb} \cdot (T_h - T_b) - G_{br} \cdot (T_b - T_r) \tag{7b}$$

where T_h is the temperature of the heater (measured in °C), T_b is the temperature of the air inside the box (°C), T_r is the known input room temperature (°C); C_h is the heat capacity of the heater (J/(Kg · °C)) while C_b is the heat capacity of the box including contained air (J/(Kg·°C)). The electric power supplied to the heatbed P_h is a constant given by voltage times current $P_h = V_h \cdot I_h$ (W), while H_h is a Boolean variable utilised to switch electric power on and off. Finally, G_{br} is a coefficient that models heat transfer between the box and the room (W/°C) while G_{hb} is a coefficient that models heat transfer between the heater and the box (W/°C).

To see the general form of the ODE in eq. (1), we set:

$$\mathbf{x} = [T_h, T_b]^{\mathrm{T}}, \quad \mathbf{u} = [H_h]^{\mathrm{T}}, \quad \mathbf{p} = [C_h, C_b, G_{br}, G_{hb}]^{\mathrm{T}}, \quad \mathbf{x}(0) = [24, 24]^{\mathrm{T}} \tag{8}$$

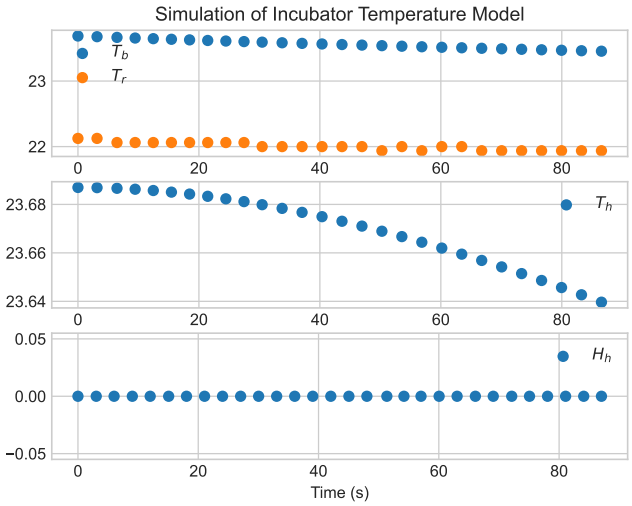

Fig. 14. Example behavior of the incubator temperature model introduced in eq. (7), obtained by running a time integration method for 30 steps from the given initial values. The vertical axis of the two topmost plots measure Celsius and the bottom plot shows the heater state which is turned off for the whole duration of the simulation.

Figure 14 illustrates the simulation of eq. (7) for 30 time steps, with a timestep of 3 seconds.

Model Calibration and Validation. The model of the incubator in eq. (7) contains a few parameters and variables that we estimate from a physical experiment, from vendor part's datasheets, or are set by the experiment:

- C_h and C_b are the heat capacities of the heater and the box, estimated from data.
- G_{br} and G_{hb} are the heat transfer coefficients between the box and the room, and between the heater and the box, respectively, also estimated from data.
- V_h and I_h are the voltage and current supplied to the heatbed, respectively, and are obtained from the data sheet of the power supply, fan, and heater.
- H_h is a Boolean variable that represents the state of the heatbed, and is obtained from the controller signal defined for the experiment.
- T_r is measured by the sensor inside the room.
- T_b is measured by the sensors inside the box. We take the average of the two sensors.
- T_h *cannot be measured by the sensors*, but this value will still be estimated since eq. (7) relates it to T_b.
- The initial values for the heatbed and box air are known as well as the room temperature, because the incubator starts out "cold".

The physical experiment consists of hard coding the controller so that it outputs a particular profile of on and off while all data is being recorded. The

on-off signal can be seen in fig. 15. The calibration then consists of solving an optimization problem (a non linear least squares) that will try and estimate the values for the parameters above for which T_b obtained from the simulation best matches the T_b obtained from the data. For the incubator this process takes a few seconds and the results can be seen in fig. 15. As can be seen the model has an almost perfect agreement with the experiment carried out.

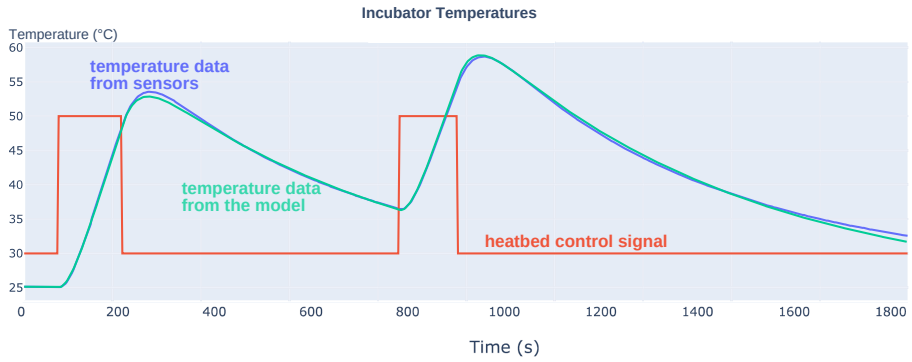

Fig. 15. Calibration results of eq. (7). The blue line is the average temperature representing the air temperature inside the incubator, while the green line represents the results from our calibrated model. The red line shows the scaled states of the heatbed: on (= 1) and off (= 0).

Revisiting the discussion in subsection 2.3, it is worth noting that several assumptions are implicit in this calibration process: First the calibration was carried out at room temperature of 20°C, where the styrofoam box operates in a certain environment where it has a constant heat capacitance. If now we try to conduct a *simulation* using a heat signal that is very different than the one used in fig. 15 (for instance, a always on signal that would drive the temperature of the heat bed to 80°C and maintain that temperature for 4 hours), a heat capacitance of the box could change due to the extended exposure to a high temperature thus rendering our simulation invalid. This is a common problem in calibration and validation of models, and it is important to keep in mind that *the model is only valid for the conditions under which it was calibrated.*

4.5 Visualization

We illustrate two DT visualisation approaches for the incubator prototype:

Incubator Dashboard InfluxDB provides a usable dashboard which gives a structured view of temperature data, fan/heater states, and controller parameters/state, enabling easy system monitoring. Figure 16 shows an example of

visualisation for a DT using the incubator. The dashboard combines graphs of evolving signals and a real-time system state overview. Visualisations may also include system controls and interactive elements, offering users deeper insights into specific data. In addition new queries can be made with sophisticated filters for aggregating information.

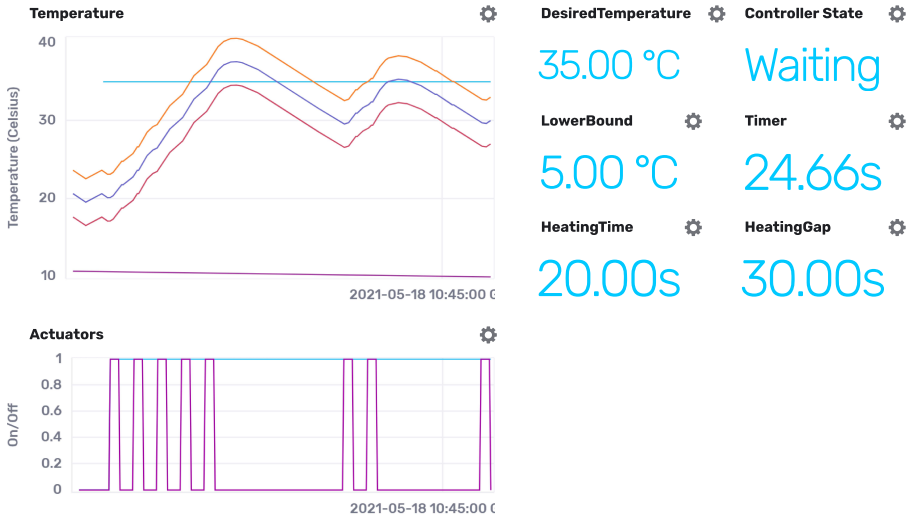

Fig. 16. A dashboard overview for the Incubator.

Incubator 4D Visualization A 4D visualization, shown in fig. 17, can be started using Godot Engine, a free and open-source game engine. The code subscribes to RabbitMQ and queues messages containing the sensor and actuator information and updates the graphical objects accordingly.

AR: Incubator Prototype AR offers enhanced interaction by visualizing the internal state of the incubator on a mobile device. This allows users to inspect the tempeh without disrupting the incubation process, improving user experience and system efficiency.

4.6 Monitoring

The monitoring service in the incubator DT is implemented using a Kalman filter. For the full details we refer the reader to the technical report in [18].

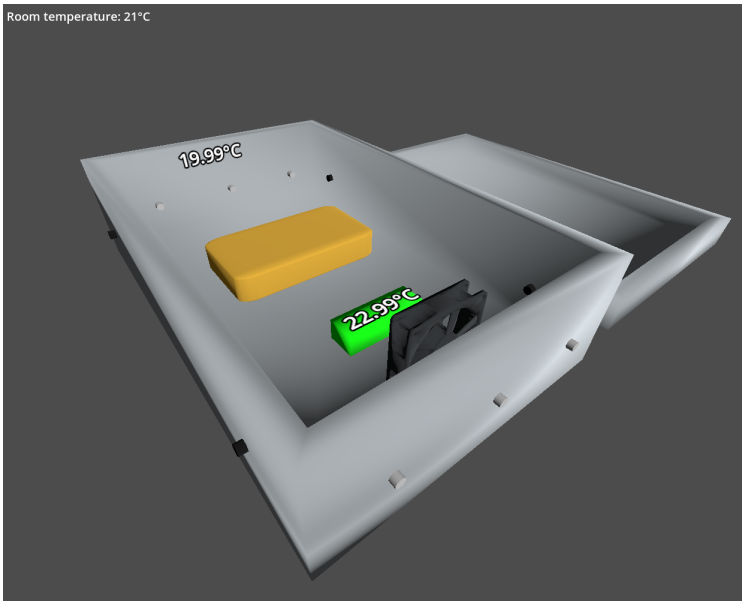

Fig. 17. Snapshot of a 4D visualisation of the incubator. From the online documentation [24].

Goal of the Kalman Filter. As illustrated in fig. 7, the Kalman Filter (KF) combines measurements T_k^b, inputs P_k, T_k^r, Gaussian distribution parameters $\mu_{T_{k-1}^h}, \Sigma_{T_{k-1}^h}$, and a model in eq. (11), to estimate the parameters $\mu_{T_k^h}, \Sigma_{T_k^h}$:

$$\begin{bmatrix} \mu_{T_k^h} \\ \Sigma_{T_k^h} \end{bmatrix} = KF\left(\begin{bmatrix} \mu_{T_{k-1}^h} \\ \Sigma_{T_{k-1}^h} \end{bmatrix}, \begin{bmatrix} P_k \\ T_k^r \end{bmatrix}, T_k^b\right), \quad \text{with } T_K^h \sim \mathcal{N}(\mu_{T_k^h}, \Sigma_{T_k^h}). \tag{9}$$

For using the Kalman filter, the first step is to write the discrete time state space model of the system. The incubator system is described by the following linear discrete time system:

$$\begin{bmatrix} T_k^h \\ T_k^b \end{bmatrix} = A \begin{bmatrix} T_{k-1}^h \\ T_{k-1}^b \end{bmatrix} + B \begin{bmatrix} P_k \\ T_k^r \end{bmatrix}, \qquad y_k = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} T_k^h \\ T_k^b \end{bmatrix} = T_k^b, \tag{10}$$

where T^h, T^b, and T^r are the temperature of heater, the temperature of the air inside the incubator, and the temperature of the room, respectively. P represents the power supply on/off function. A and B are 2 by 2 matrices containing the parameters of the system estimated by the calibration process.

The Kalman filter assumes that there is Gaussian process and measurement noise, so the above model is actually given as follows:

$$\begin{bmatrix} T_k^h \\ T_k^b \end{bmatrix} = A \begin{bmatrix} T_{k-1}^h \\ T_{k-1}^b \end{bmatrix} + B \begin{bmatrix} P_k \\ T_k^r \end{bmatrix} + \epsilon_k, \qquad y_k = T_k^b + \delta_k, \tag{11}$$

Fig. 18. An AR application for the incubator. Reproduced from [21].

where ϵ_k and δ_k are random variables representing process noise and measurement noise, satisfying $\epsilon_k \sim \mathcal{N}(0, R_k)$ and $\delta_k \sim \mathcal{N}(0, Q_k)$, respectively.

The algorithm was implemented into a service. We now show how the error in the predictions of the Kalman filter can be used to detect anomalies such as the lid being opened.

Opening the lid causes the system to violate the physical principles the model was originally built on, and will therefore make the KF fail to track what is happening. The result can be seen in fig. 19. The orange line is the control signal of the heater, "high" means turning on the heater and "low" is turning off. Green line, purple line, and blue line are predicted state form the model, estimated state from KF, and measurement from the sensors respectively.

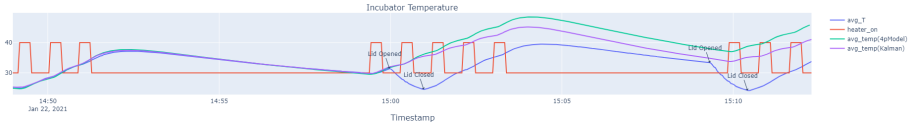

Fig. 19. Results of anomaly detection for incubator DT. Reproduced from [18] and the full interactive plot can be inspected online at `https://github.com/INTO-CPS-Association/example_digital-twin_incubator/blob/master/software/incubator/datasets/20210122_lid_opening_kalman/results.html`.

During the experiment, we opened the lid at `14:59:58` and `15:09:19`. After opening the lid for around `1` minute, we closed the lid at `15:00:58` and `15:10:25` respectively. As can be seen in fig. 19, after opening the lid at `14:59:58`, the anomaly was detected since there was a discrepancy between the estimated state from KF and the sensory data.

After closing the lid, the purple line and the blue line were merging gradually, i.e. the discrepancy reduced gradually. Before they converge together we opened the lid again, the discrepancy started to increase again. Should we have waited longer before opening the lid again, the discrepancy would have become virtually zero.

Through the running example, the KF successfully demonstrated the ability of state estimation. Such estimation succeeds in detecting an anomaly in the incubator system.

The state estimation service is a crucial one, since its results can be used to get valid initial states for the simulations conducted in decision support services (introduced next), and the anomalies detected can be used to trigger the reconfiguration service (detailed in subsection 3.4).

4.7 Decision Support

We present two decision support services offered by the incubator DT.

Power Supply Replacement. In the first example, consider a scenario where the incubator is running, but the power supply needs replacement without interrupting the tempeh fermentation process. A feasible solution involves boosting the temperature before the replacement, allowing it to drift within a safe range during the power outage. Several simulations can be performed to determine the optimal heater boost time (H). As shown in fig. 20, these simulations help identify the best value for H to avoid excessive overheating.

The figure illustrates three simulations that predict how the PTtwin will behave under different temperature boosts. One simulation corresponds to the current configuration, allowing an estimate of the time available for the power supply swap. Another configuration causes overheating, while the third shows the optimal balance between maintaining temperature and maximizing the time available for replacement. The optimal temperature boost is then applied to reconfigure the controller's parameter (H) for best performance.

Design Space Exploration. In the second example, we explore an optimization problem: balancing ideal temperature maintenance with minimizing actuator effort. These objectives conflict because frequent power cycling to maintain temperature increases actuator wear. To solve this, multiple simulations are run, each prioritizing either temperature accuracy or actuator longevity. The simulation results, displayed in fig. 21, reveal the trade-offs, with the Pareto front showing optimal solutions that balance these objectives.

The parameter b controls how far the temperature can drop below the ideal before triggering the heater. As shown in the figure, minimal actuator effort

corresponds to a large value of b, but this leads to greater temperature error, showing a clear trade-off between actuator longevity and temperature stability.

4.8 Reconfiguration

Aside from the reconfiguration related to the decision support services, another example was implemented in the incubator, following the MAPE-K loop (recall subsection 3.4). It reflects the fact that during tempeh fermentation, the user is expected to open the lid to inspect the contents of the incubator.

If the lid of the incubator is opened unexpectedly, the system's thermal model and controller become invalidated. Using the MAPE-K loop:

- **Monitor**: The reconfiguration service collects temperature data and detects the anomaly caused by the open lid.
- **Analyze**: It determines that the deviation is due to the lid being open, affecting the thermal properties.
- **Plan**: The reconfiguration service decides to recalibrate the thermal model and reoptimize the controller parameters to accommodate the new conditions. This means setting a slightly lower temperature setpoint to both save energy and avoid a safety hazard where the temperature overshoots if the lid is closed [46].
- **Execute**: Updated parameters are applied to the controller.

Figure 22 shows the results of this process, highlighting the moment the lid is opened and then closed. Notice how the controller profile changes to adapt to the new conditions.

The reconfiguration service makes use of other services: the monitor phase uses the results of the state estimation service to detect anomalies in its prediction error, and the plan phase uses the results of the decision support services to both estimate new parameters for the model (from a calibration service), and estimate new controller optimal parameters (from the DSE service).

5 Summary and Future Work

This tutorial covered the fundamentals of Digital Twin technology, the building blocks necessary to develop a DT, and a practical case study of an incubator system, in a top down manner. Most content has been published elsewhere in a fragmented manner. The value in this tutorial is to bring everything together in a coherent manner.

One aspect that has not been covered in this tutorial, is compositions of DTs. A logical extension of the incubator to illustrate this, is to consider the scenario of multiple Incubator DTs. Consider a factory producing large quantities of Tempeh through many incubators that need to be managed both at unit level and at the aggregate level. The PTs can interfere with each other, for example, by sharing adjacent walls. The environment is also shared, for example, through the same air conditioning system. The corresponding DT services of each incubator

may need to cooperate to ensure optimal operation. Distributed DSE needs to be used, to optimally configure controllers. A similar argument can be made for reconfiguration services. Further challenges spanning from this scenario have been sketched in [16].

Digital Twin technology holds significant potential for transforming industries by enabling more efficient, reliable, and adaptable systems. As research and technology continue to advance, the applications of DTs are expected to expand, offering new opportunities for innovation. We hope this tutorial plays an important role in that direction by breaking down this complex technology into simple, well understood concepts.

Acronyms

AR Augmented Reality
CPS Cyber-Physical System
DSE Design Space Exploration
DT Digital Twin
EKF Extended Kalman Filter
IVP Initial Value Problem
KF Kalman Filter
ODE Ordinary Differential Equation
PT Physical Twin
SOA Service-Oriented Architecture

Acknowledgments. The authors are grateful to the Poul Due Jensen Foundation, which has supported establishing a new Centre for Digital Twin Technology at Aarhus University.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Babu, V.S., Behl, M.: F1tenth.dev – An Open-source ROS based F1/10 Autonomous Racing Simulator. In: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE). pp. 1614–1620 (Aug 2020). https://doi.org/10.1109/CASE48305.2020.9216949
2. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification, LNCS, vol. 10457, pp. 135–175. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
3. Bartocci, E., Falcone, Y.: Lectures on Runtime Verification: Introductory and Advanced Topics, vol. 10457. Springer (2018)

4. Basin, D., Klaedtke, F., Zalinescu, E.: The MonPoly Monitoring Tool. In: RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools. pp. 8–19 (2017). https://doi.org/10.29007/89hs
5. Bogomolov, S., Gomes, C., Isasa, C., Soudjani, S., Stankaitis, P., Wright, T.: Reachability Analysis of FMI Models Using Data-Driven Dynamic Sensitivity. SIMULATION (2023). https://doi.org/10.1177/00375497241261409
6. Böttjer, T., Tola, D., Kakavandi, F., Wewer, C.R., Ramanujan, D., Gomes, C., Larsen, P.G., Iosifidis, A.: A review of unit level digital twin applications in the manufacturing industry. CIRP Journal of Manufacturing Science and Technology **45**, 162–189 (Oct 2023). https://doi.org/10.1016/j.cirpj.2023.06.011
7. Buhagiar, A.J., Christensen, R.F., Larsen, P.G., Freitas, L., Scott, W.E., Gonçalves Gomes, C.Â.: Understanding pancreas-machine interactions during preservation: A mathematical approach. In: Transplantation. vol. 107, pp. 34–35 (Oct 2023). https://doi.org/10.1097/01.tp.0000993916.49745.ee
8. Butcher, J.: Numerical Methods for Ordinary Differential Equations. John Wiley & Sons, Ltd, Chichester, UK (Jun 2003). https://doi.org/10.1002/0470868279
9. Cellier, F.E.: Continuous System Modeling. Springer Science & Business Media (1991)
10. Cellier, F.E., Kofman, E.: Continuous System Simulation. Springer Science & Business Media (2006)
11. Cengel, Y.A., Boles, M.A., Kanoğlu, M.: Thermodynamics: An Engineering Approach, vol. 5. McGraw-Hill New York (2011)
12. Deb, K.: Multi-objective optimisation using evolutionary algorithms: An introduction. In: Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing, pp. 3–34. Springer (2011)
13. Denil, J., Klikovits, S., Mosterman, P.J., Vallecillo, A., Vangheluwe, H.: The experiment model and validity frame in M&S. In: Proceedings of the Symposium on Theory of Modeling & Simulation. pp. 1–12. TMS/DEVS '17, Society for Computer Simulation International, San Diego, CA, USA (Apr 2017)
14. Ejersbo, H., Lausdahl, K., Frasheri, M., Esterle, L.: Dynamic Runtime Integration of New Models in Digital Twins. In: 2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 44–55. IEEE, Melbourne, Australia (May 2023). https://doi.org/10.1109/SEAMS59076.2023.00016
15. Ernst, R.: Codesign of embedded systems: Status and trends. In: Readings in Hardware/Software Co-Design, pp. 45–54. Elsevier (2002). https://doi.org/10.1016/b978-155860702-6/50006-5
16. Esterle, L., Gomes, C., Frasheri, M., Ejersbo, H., Tomforde, S., Larsen, P.G.: Digital twins for collaboration and self-integration. In: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C). pp. 172–177. IEEE, DC, USA (Sep 2021). https://doi.org/10.1109/ACSOS-C52956.2021.00040
17. Feng, H., Gomes, C., Gil, S., Mikkelsen, P.H., Tola, D., Larsen, P.G., Sandberg, M.: Integration Of The Mape-K Loop In Digital Twins. In: 2022 Annual Modeling and Simulation Conference (ANNSIM). pp. 102–113. IEEE, San Diego, CA, USA (Jul 2022). https://doi.org/10.23919/ANNSIM55834.2022.9859489
18. Feng, H., Gomes, C., Larsen, P.G.: Model-Based Monitoring and State Estimation for Digital Twins: The Kalman Filter (Apr 2023)

19. Feng, H., Gomes, C., Thule, C., Lausdahl, K., Iosifidis, A., Larsen, P.G.: Introduction to Digital Twin Engineering. In: 2021 Annual Modeling and Simulation Conference (ANNSIM). pp. 1–12. IEEE, Fairfax, VA, USA (Jul 2021). https://doi.org/10.23919/ANNSIM52504.2021.9552135
20. Feng, H., Gomes, C., Thule, C., Lausdahl, K., Sandberg, M., Larsen, P.G.: The Incubator Case Study for Digital Twin Engineering. Tech. rep., Aarhus University, Department of Engineering (Feb 2021)
21. Fitzgerald, J., Gomes, C., Larsen, P.G. (eds.): The Engineering of Digital Twins. Springer International Publishing, Cham (2024). https://doi.org/10.1007/978-3-031-66719-0
22. Fuller, A., Fan, Z., Day, C., Barlow, C.: Digital Twin: Enabling Technologies, Challenges and Open Research. IEEE Access **8**, 108952–108971 (2020). https://doi.org/10.1109/ACCESS.2020.2998358
23. Gil, S., Mikkelsen, P.H., Gomes, C., Larsen, P.G.: Survey on open-source digital twin frameworks–A case study approach. Software: Practice and Experience p. spe.3305 (Jan 2024). https://doi.org/10.1002/spe.3305
24. Gomes, C.: INTO-CPS-Association/example_digital-twin_incubator. The INTO-CPS Association (May 2024)
25. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A Survey. ACM Computing Surveys **51**(3), 49:1–49:33 (2018). https://doi.org/10.1145/3179993
26. Griffiths, D.F., Higham, D.J.: Numerical Methods for Ordinary Differential Equations. Springer Undergraduate Mathematics Series, Springer, London (2010). https://doi.org/10.1007/978-0-85729-148-6
27. Havelund, K., Roşu, G.: Monitoring Java Programs with Java PathExplorer. Electronic Notes in Theoretical Computer Science **55**(2), 200–217 (Oct 2001). https://doi.org/10.1016/S1571-0661(04)00253-1
28. IEEE: International Standard ISO/IEC/IEEE 15288:2015(E), Systems and Software Engineering — System Life Cycle Processes. ISO/IEC and IEEE Computer Society (2015)
29. Jones, D., et al.: Characterising the digital twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology **29**, 36–52 (2020)
30. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (Jan 2003). https://doi.org/10.1109/MC.2003.1160055
31. Kühne, T.: What is a Model? In: Language Engineering for Model-Driven Software Development. vol. 04101. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) (2005). https://doi.org/10.4230/DagSemProc.04101.15
32. Liu, M., Fang, S., Dong, H., Xu, C.: Review of digital twin about concepts, technologies, and industrial applications. Journal of Manufacturing Systems **58**, 346–361 (Jan 2021). https://doi.org/10.1016/j.jmsy.2020.06.017
33. Madsen, E., Tola, D., Hansen, C., Gomes, C., Larsen, P.G.: AURT: A Tool for Dynamics Calibration of Robot Manipulators. In: 2022 IEEE/SICE International Symposium on System Integration (SII). pp. 190–195. IEEE, Narvik, Norway (Jan 2022). https://doi.org/10.1109/SII52469.2022.9708769
34. Maler, O., Nickovic, D.: Monitoring Temporal Properties of Continuous Signals. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, vol. 3253, pp. 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12

35. Ničković, D., Yamaguchi, T.: RTAMT: Online Robustness Monitors from STL. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis. pp. 564–571. LNCS, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_34
36. Oakes, B., Gomes, C., Larsen, P.G., Denil, J., DeAntoni, J., Cambeiro, J., Fitzgerald, J.: Examining Model Qualities and Their Impact on Digital Twins. In: Annual Modelling and Simulation Conference. pp. 220–232. Ontario, Canada (2023)
37. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
38. Rizzi, S.: What-if analysis. Encyclopedia of Database Systems pp. 1–6 (2009)
39. Rose, M., Fitzgerald, J.: Genetic algorithms for design space exploration of cyber-physical systems: An implementation in into-CPS. In: Macedo, H.D., Thule, C., Pierce, K. (eds.) Proceedings of the 19th International Overture Workshop. Overture (Oct 2021)
40. Simon, D.: Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. John Wiley & Sons (2006)
41. Stachowiak, H.: Allgemeine Modelltheorie. Springer-Verlag, Wien and New York (1973)
42. Steinkraus, K.H., Hwa, Y.B., Van Buren, J.P., Provvidenti, M.I., Hand, D.B.: Studies on tempeh. An Indonesian fermented soybean food. Food Research **25**, 777–788 (1960)
43. Stewart, J.: Calculus: Early Transcendentals. Cengage Learning (2012)
44. van Amerongen, J.: Dynamical Systems for Creative Technology. Controllab Products B.V., Enschede (2010)
45. Woodcock, J., Gomes, C., Macedo, H.D., Larsen, P.G.: Uncertainty Quantification and Runtime Monitoring Using Environment-Aware Digital Twins. In: Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends. LNCS, vol. 12479, pp. 72–87. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-83723-5_6
46. Wright, T., Gomes, C., Woodcock, J.: Formally Verified Self-adaptation of an Incubator Digital Twin. In: Leveraging Applications of Formal Methods, Verification and Validation. Practice. vol. 13704, pp. 89–109. Springer Nature, Switzerland (2022). https://doi.org/10.1007/978-3-031-19762-8_7
47. Zeigler, B.P.: Theory of Modelling and Simulation. New York, Wiley (1976)
48. Zheng, Y., Yang, S., Cheng, H.: An application framework of digital twin and its case study. Journal of Ambient Intelligence and Humanized Computing **10**(3), 1141–1153 (Mar 2019). https://doi.org/10.1007/s12652-018-0911-3

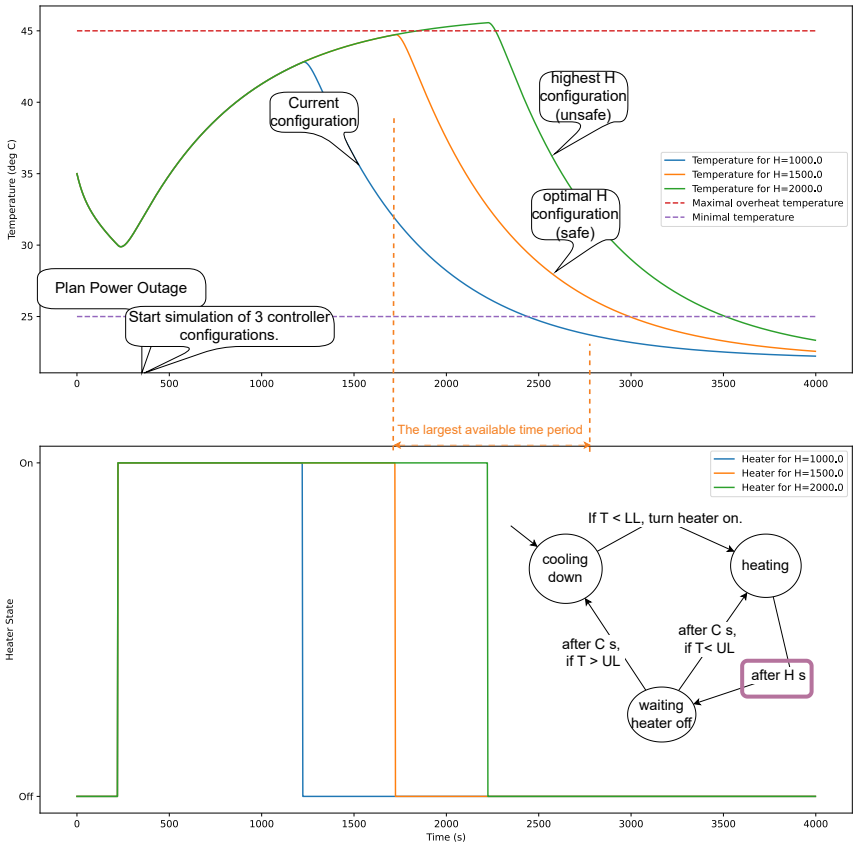

Fig. 20. What-if simulation results for controller configuration. At roughly 400 seconds a power outage plan is initiated which leads to the beginning of 3 consecutive simulations with 3 different controller configurations, each representing a different value for the parameter H of the controller. One of the simulations is unsafe because it leads toa very high maximum temperature. The optimal configuration of the controller is indicated by the Orange Line.

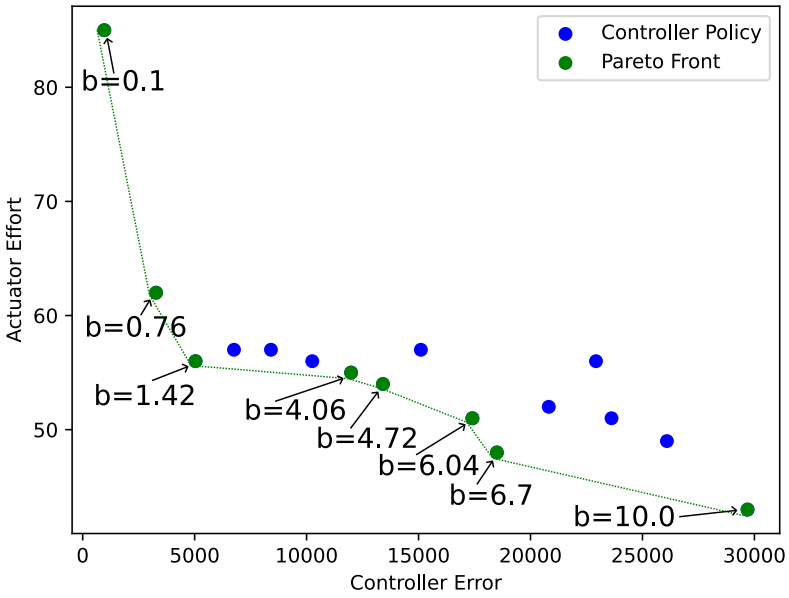

Fig. 21. DSE results for the incubator case-study with two conflicting objectives. Reproduced from [21].

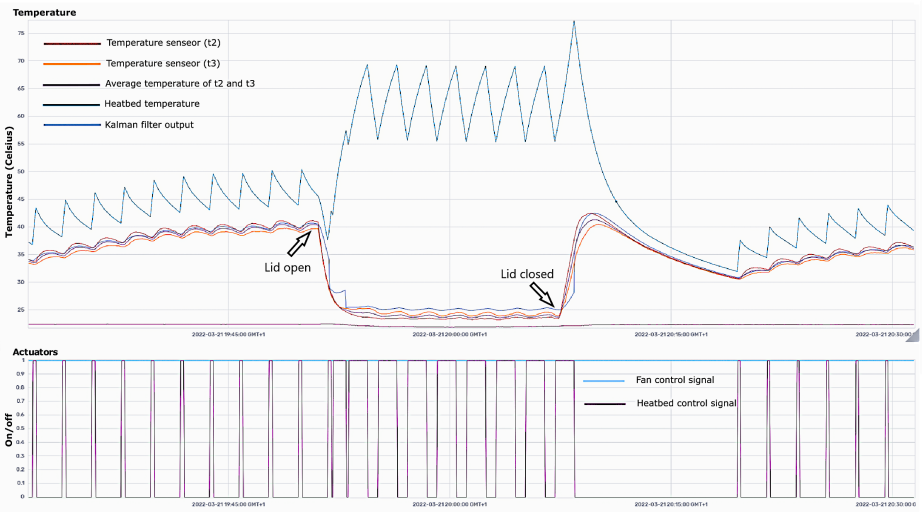

Fig. 22. Experimental result of self-adaptation. Adapted from [17].