JOURNAL OF BUILDING PERFORMANCE SIMULATION
2021, VOL. 14, NO. 5, 586–610
https://doi.org/10.1080/19401493.2021.1986574

Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings

David Blum[a], Javier Arroyo[b,c,d], Sen Huang[e], Ján Drgoňa[e], Filip Jorissen[b], Harald Taxt Walnum[f], Yan Chen[e], Kyle Benne[g], Draguna Vrabie[e], Michael Wetter[a] and Lieve Helsen[b,c]

[a]Lawrence Berkeley National Laboratory, Berkeley, CA, USA; [b]Department of Mechanical Engineering, KU Leuven, Leuven, Belgium; [c]EnergyVille, Thor Park, Waterschei, Belgium; [d]Flemish Institute for Technological Research (VITO), Mol, Belgium; [e]Pacific Northwest National Laboratory, Richland, WA, USA; [f]SINTEF Community, Oslo, Norway; [g]National Renewable Energy Laboratory, Golden, CO, USA

ABSTRACT
Development of new building HVAC control algorithms has grown due to needs for energy efficiency and operational flexibility. However, case studies demonstrating new algorithms are largely individualized, making algorithm performance difficult to compare directly. In addition, the effort and expertise required to implement case studies in real or simulated buildings limits rapid prototyping potential. Therefore, this paper presents the Building Optimization Testing Framework (BOPTEST) and associated software for simulation-based benchmarking of building HVAC control algorithms. A containerized run-time environment (RTE) enables rapid, repeatable deployment of common building emulators representing different system types. Emulators use Modelica to represent realistic physical dynamics, embed baseline control, and enable overwriting supervisory and local-loop control signals. Finally, a common set of key performance indicators are calculated within the RTE and reported to the user. This paper details the design and implementation of software and demonstrates its usage to benchmark a Model Predictive Control strategy.

ARTICLE HISTORY
Received 26 July 2021
Accepted 21 September 2021

KEYWORDS
Buildings; controls; HVAC; modelica; FMI; benchmarking

1. Introduction

1.1. Background

There is a growing focus on the control of building heating, ventilating, and air-conditioning (HVAC) systems. Needs for reducing CO_2 emissions, integrating renewable, variable, and distributed energy sources into electric and thermal grids, adapting to natural disasters and health emergencies, and operating complex system architectures have prompted efforts to improve upon existing control algorithms. For example, such efforts include ASHRAE's publishing of Guideline 36 (ASHRAE 2018), development of grid-friendly control strategies (Kim et al. 2016), resolute interest in Model Predictive Control (MPC) (Drgoňa et al. 2020), and rapidly growing interest in data-driven control (Vázquez-Canteli and Nagy 2019). While these new control algorithms are promising, challenges remain in deploying them widely in practice.

One key challenge is benchmarking the performance of these algorithms against state-of-the-art algorithms and each other. Often, each new or improved algorithm is demonstrated individually through simulation or field testing on a particular application to show the benefits they promise over an alternative deemed as a baseline. With such individualized studies and baseline definitions, it remains unclear how each control algorithm compares to another on a particular application in a quantitative way and whether similar benefits are observed in other applications. Such insights would enable building owners and operators to invest in the most effective solutions and the development community to identify areas of needed continued work.

A second key challenge is the effort and expertise required for setting up and executing such individualized tests. While testing in real buildings has the advantage of capturing the complex building control process, it is also risky, expensive, and time-consuming to complete. Building owners and operators need to ensure the services of the building are still provided and real buildings are subject to uncontrollable, unmeasurable, stochastic, and slow-changing operating environments. This makes it difficult to evaluate many control strategies under the same test conditions. Simulation-based testing offers solutions to these problems by providing a controlled, configurable testing environment. However, creating a realistic simulation of building operation requires expertise in building physics and systems modelling, which may

CONTACT David Blum dhblum@lbl.gov

not be shared by control, optimization, and data science experts. In addition, the diversity of simulation modelling approaches and evaluation metrics makes these studies difficult to compare.

The development of a common platform for simulation-based testing helps address the previously described challenges by enabling rapid prototyping of new solutions, their testing on new applications, and direct comparison to alternative approaches. This motivates the work presented in this paper: to provide a common framework and publicly available software tools necessary for simulation-based benchmarking of building HVAC control algorithms using high-fidelity building emulators. The resulting framework and software tools are known as the Building Optimization Testing (BOPTEST) Framework and are currently available in a GitHub repository at https://github.com/ibpsa/project1-boptest with version `v0.1.0`.

1.2. Paper objectives and contributions

This paper expands upon a previous conference publication that covered an earlier version of the work (Blum et al. 2019). That publication presented the main elements of the BOPTEST framework and their preliminary implementations, and demonstrated its use with a first-order test system and a feedback controller. This paper expands on the description of each element, introduces the development since (Blum et al. 2019), including new features and updates to the application programming interface (API), development of more complex emulator models, refinement of key performance indicators, improvements to computational performance and data management, and demonstrates the framework's usage for the evaluation of an MPC strategy. Earlier elements and previous versions of BOPTEST were also used in Arroyo, Spiessens, and Helsen (2020) for the model identification study of a multi-zone building, Walnum, Sartori, and Bagle (2020) for the evaluation of an MPC strategy for a building radiator served by a district heating system, Huang, Chen et al. (2018) for assessing ASHRAE guideline controls with a typical large office building, and Bünning et al. (2021) for the evaluation of three different data-driven modelling methods for use in MPC for a single-zone building radiator served by a boiler. Also, an OpenAIGym environment for BOPTEST is being developed as described in Arroyo et al. (2021) and a new emulator to be integrated with BOPTEST was described in Yang et al. (2020). Finally, a software platform for metamodelling and advanced controller development, such as MPC and Reinforcement Learning (RL), that interfaces with BOPTEST and high-fidelity Spawn building emulators is developed in Marzullo et al. (2021). With continued development since Blum et al. (2019) and a growing ecosystem of usage, the specific objectives of this paper are as follows:

(1) Describe the design and implementation of the BOPTEST software in detail.
(2) Explain its intended use.
(3) Demonstrate the use of BOPTEST for performance evaluation of an example MPC strategy.

BOPTEST contributes to the field by integrating high-fidelity building simulation using Modelica (Mattsson and Elmqvist 1997) and the Functional Mockup Interface Standard (FMI) (Blochwitz et al. 2011) with containerized software deployment using Docker to enable deployment of virtual building emulators in a reproducible computing environment locally cross-platform or in the cloud. In addition, the BOPTEST computing environment provides all of the functionality needed for advanced control evaluation, including overwriting control signals in emulators at the supervisory or local-loop levels, reading measurement signals, obtaining boundary condition forecasts, setting up testing scenarios, and computing Key Performance Indicators (KPI), through a generic HTTP API accessible by most programming languages. Finally, BOPTEST provides a set of ready-to-use building emulators, so-called *test cases*.

The structure of this paper is as follows. Section 2 provides a review of relevant literature on control performance evaluation, simulation technology, and previously-developed frameworks, and discusses implications on the development of BOPTEST. Then, Section 3 introduces framework requirements, presents the overall concept of BOPTEST, and describes the design, implementation, and usage of its components. Section 4 demonstrates the use of the framework to evaluate the performance of an MPC strategy. Section 5 discusses the performance of the framework, associated limitations, and future work. Finally, Section 6 concludes the paper.

2. Literature review

2.1. Control performance evaluation

The initial challenge of evaluating new control algorithms is the large number of factors that influence performance. A comprehensive review on MPC for buildings (Drgoňa et al. 2020) suggests these factors stem from case study characteristics, such as building size, occupancy and usage, climate, HVAC system design, controllability, measurement availability, and control objective, controller characteristics, such as general approach, specific algorithms, and software implementation, and evaluation metrics. As summarized by Afram and Janabi-Sharifi (2014), common evaluation metrics include energy, cost, peak load shifting capability, transient response, steady-state response, control of variables within bounds, reduction in fluctuations from a setpoint, system efficiency, robustness to disturbances and changes, indoor air quality (IAQ), thermal comfort, and computational time. Other potential metrics suggested by Drgoňa et al. (2020) target the controller implementation specifically, such as computer hardware and software requirements, data requirements, implementation effort, and installer expertise. Another review (Zhan and Chong 2021) discusses specifically the influence of data availability on MPC model approaches and performance. All three studies (Afram and Janabi-Sharifi 2014; Drgoňa et al. 2020; Zhan and Chong 2021) conclude that further comparison of approaches is needed, and the lack of a framework that includes common test cases and evaluation methods remains a barrier to efficiently identifying the promising approaches and needs for continued development in MPC. The same problem has been identified in the RL community. Two review publications, Vázquez-Canteli and Nagy (2019) presenting 105 studies and Wang and Hong (2020) presenting 77 studies, join (Drgoňa et al. 2020; Zhan and Chong 2021) to remark that the inability to benchmark algorithms on common test cases remains a barrier to efficiently identifying the most promising approaches for given applications. Finally, Wölfle, Vishwanath, and Schmeck (2020) discusses the need for benchmark environments for building energy optimization and further proposes seven topics such environments should address: *scenario*, *relevance*, *scope*, *realism*, *performance measure*, *reproducibility*, and *interface*. Each of these topics is addressed in this paper.

A second challenge is the choice of a comparative baseline for the chosen case study, which can highly influence the reported benefits of new control. Typically, baseline control is implemented with Rule-Based Control (RBC), though the performance of such control can vary widely. For example, consider the commonly-studied multi-zone Variable Air Volume (VAV) system. When evaluating improvements to heating and cooling enabling conditions, supply air and boiler temperature resets, enthalpy-based economizer control, night flush ventilation, and zone temperature set points individually and in groups, Mansson and McIntyre (1997) found differences in energy use compared to a baseline ranging from −10% to 35%. Similarly, when evaluating different strategies for zone temperature set points, optimal start, economizer, minimum outside air, static pressure reset, supply air temperature reset, and terminal box minimum airflow, Pang, Piette, and Zhou (2017) found energy savings of 'good practice' strategies to be upwards of 67% and 64% (in two climates) if compared against 'poor practice' baselines, and only 12% and 14% (in the same two climates) if compared against 'average practice' baselines. Finally, Fernandez et al. (2017) shows energy savings ranges from 4% to 59% due to ideal implementation of 37 fault correction and RBC strategies, across many building types and climates, depending on if the baseline is 'efficient', 'typical', or 'inefficient'. Similar arguments can be made for hydronic thermally active building structures (TABS), often cited as systems with high efficiency and load shifting potential that benefit from advanced controls. The review by Romaní, de Gracia, and Cabeza (2016) classifies typical control strategies into flow-controlled and supply water temperature controlled with or without dependence on outside air temperature or feedback from indoor temperature. More advanced controllers utilize predictive capabilities and can be compared to any one of these typical strategies. For example, Prívara et al. (2011) compared MPC to open-loop supply temperature reset as a function of outside temperature, while Sourbron, Verhelst, and Helsen (2013) compared MPC to a similar baseline control, but with additional feedback control on the TABS surface temperature as a function of room comfort limits.

2.2. Simulation technology

The underlying simulation technology of the building emulator is critical to enabling evaluation with realistic test cases. Primary factors that were considered include the fidelity of the model, time-resolution of the simulation, and representation of both current and future system designs. Other important factors include software availability, maintenance and support, co-simulation interfaces for external controllers, and commercial licensing requirements.

EnergyPlus (Crawley et al. 2001) and TRNSYS (Klein 2017) are two popular building energy simulation programs, with EnergyPlus being freely available and supported by the US Department of Energy, and TRNSYS being commercially available. Both programs have strong capabilities for modelling envelope physics as well as a large variety of HVAC components and systems. However, the load-based HVAC performance calculation procedure in EnergyPlus is based on control inputs and outputs that differ from how actual building systems are controlled. EnergyPlus also does not model the relationship between pressure and flow in HVAC pipe and duct networks. These modelling approaches, along with a fixed timestep integration method with a lower limit of one minute, precludes it from explicitly representing the controllers and associated control logic present in a real building, such as a local-loop PI controller. TRNSYS system models are composed of individual component models with defined input and output variables that are linked together. While this allows for flexibility in the system architecture to be modelled, Wang (1999) reported serious convergence difficulty when simultaneously simulating a VAV duct network model using pressure-flow relationships with thermal models, and concluded that such problems would need to be addressed for continued usage. For the simulation to converge, the system pressure-flow balance needed to be implemented in a single, separate component from the thermal and control models. This approach would be cumbersome to duplicate for other building and system designs. Later attempts described in Blum (2013) to simulate a duct network with three flow splits using pressure-flow relationships failed to converge to a solution when the pressure difference driving flow through the system approached zero. HVACSIM+ (Park, Clark, and Kelly 1985) is similar to TRNSYS in the definition of system models using components with defined input/output interfaces and a library of dynamic component models for the evaluation of HVAC controls was developed for both in (Haves and Norford 1997). However, HVACSIM+ lacks strong developer and user communities.

In contrast, other software that use equation-based model descriptions and declarative programming is growing in popularity. Three examples are the basis of the SPARK program (Sowell et al. 1986), the Neutral Model Format (NMF) (Sahlin and Sowell 1989) used in the commercial software IDA/ICE (Björsell et al. 1999), and the Modelica language (Mattsson and Elmqvist 1997), whose open-source specification is utilized across the automotive, aerospace, process, and buildings industries. A primary advantage of equation-based model descriptions for the building systems domain is the ability to symbolically manipulate model equations such that computationally efficient code can be generated for the simulation of building energy systems, including heat transfer physics, pressure-flow networks, and explicit control formulations (Wetter, Bonvini, and Nouidui 2016). Modelica tools combine such code with variable time-step solvers that are capable of handling nonlinear, hybrid, and stiff mixed continuous and discrete-time systems of differential-algebraic equations (DAE) that often occur when coupling building physics, HVAC, and controls. Such approaches, therefore, can be used for explicit representation of the controllers and control logic present in real buildings and used for realistic control evaluation. Open-source development of Modelica libraries for building energy simulation has gained significant traction since the development of four open-source component libraries (Nytsch-Geusen et al. 2013; Wetter et al. 2014; Mueller et al. 2016; Jorissen et al. 2018a), their unification into a kernel library under IEA Annex 60 (Wetter et al. 2015), and continued development under IBPSA Project 1 (Wetter et al. 2019).

In addition to model implementation, it is important to consider the model interfaces for external controller integration. EnergyPlus, TRNSYS, and the Modelica Buildings Library (Wetter et al. 2014) all provide native interfaces with Python, while TRNSYS and IDA/ICE provide native interfaces with MATLAB. An EnergyPlus-MATLAB tool has been developed in Nghiem (2010). Meanwhile, more generic coupling between building simulation tools, as well as other potential external programs, was provided by the Building Controls Virtual Test Bed (BCVTB) (Wetter 2011), which has subsequently been used frequently for the evaluation of control algorithms. The Functional Mockup Interface Standard (FMI) (Blochwitz et al. 2011) is a generic model interface standard supported by over 150 tools across industries that allows for supporting tools to package models, solvers (optionally), and other necessary data into Functional Mockup Units (FMU) that can be imported into different supporting tools and/or co-simulated with other FMUs. Together, EnergyPlus, Modelica, and FMI are fundamental to the development of a next-generation building controls simulation environment, called Spawn (Wetter et al. 2020).

2.3. Previous and existing frameworks

The use of building emulators for control algorithm benchmarking was introduced during the International Energy Agency (IEA) Energy in Buildings and Communities Program (EBC) Annex 17 (Mansson and McIntyre 1997). There, participants built building emulators to be coupled with real Building Energy Management Systems (BEMS). Although successful case studies were reported, there are two major limitations associated with the emulator implementations. First, they relied on real BEMS hardware for control algorithm implementation, and this limits their support of control algorithm development. Second, they did not use a software architecture for encapsulating and sharing emulators. This leads to potentially inconsistent results due to variations in model implementation and settings used for testing. As IEA EBC Annex 30 (Warren 2002) points out, an additional obstacle is the requirement for control developers and design engineers to become familiar with simulation tools and dynamic system modelling approaches. The development of building emulators for controls testing continued with control-hardware-in-the-loop environments, such as an environment utilizing SPARK as a simulation engine (Xu, Haves, and Deringer 2004) and the Virtual Cybernetic Building Testbed (VCTB) utilizing HVACSIM+ as a simulation engine (Bushby et al. 2010), and hardware-in-the-loop environments, such as that using Modelica as a simulation engine for zone dynamics and real air handing unit (AHU) equipment (Huang et al. 2018). As Huang et al. (2018) points out, and similar to the observations above, testing environments with hardware-in-the-loop are useful for evaluating practical issues surrounding the implementation of controls. However, they are limited in their scalability to different test cases or wide variety of control algorithms. For example, analogue-digital (A/D) converters have a limited number of I/O channels and need re-wiring or configuration for new test cases, and real HVAC equipment installations, such as fans, chillers, and flow networks, can only represent a limited number of test cases.

Entirely software-based testing and benchmarking environments have also emerged. Environments using EnergyPlus as a simulation engine have been developed with RESTful, in Pallonetto et al. (2019), and Haystack, in (NREL 2021), APIs for controller interfacing. The engines for these examples have been deployed as web-services, and the use of Docker, as shown in (NREL 2021), allows for rapid, flexible deployment locally. The CityLearn environment (Vázquez-Canteli et al. 2019) was implemented in Python for the benchmarking of Reinforcement Learning controllers integrating building, storage, and PV generation systems. Carefully designed testing scenarios and internal key performance metric (KPI) calculation enabled a grand challenge to be issued (Vázquez-Canteli 2020). However, building energy use is determined by pre-calculated heating and cooling loads, rather than explicit dynamic simulation of the HVAC system. An office building emulator including explicit dynamic simulation of the HVAC system and controls, as well as stochastic occupancy, was implemented and made available online for testing external control strategies in Togashi and Miyata (2019). Communication between the server emulator and client test controller was established using BACnet and a VPN. It was used to conduct a grand challenge known as The First World Championship in Cybernetic Building Optimization (WCCBO), which attracted 33 participant teams over a two-month period to compete to reduce annual energy use and improve thermal comfort (Togashi, Miyata, and Yamamoto 2020). The work succeeded in providing an open platform for control strategy evaluation and benchmarking, though the implementation of component models using custom software limits the approach's scalability to new test cases and the lack of forecasting service precludes the testing of predictive controllers, such as MPC and RL. Finally, Energym (Scharnhorst et al. 2021) is a Python-based, open-source library of building models, implemented in EnergyPlus and Modelica, for benchmarking control algorithms. Though similar in concept to BOPTEST, execution differs in a few different ways, including the following. The Modelica-based models in Energym utilize first-order models for envelopes, zone temperature limits are constant for all time, KPI calculation does not utilize time integration, thereby metrics like total energy use or cost are not calculated, the emulation models do not have the ability for overwriting of local-loop and supervisory control signals, and the interface is based in Python, limiting the potential for testing of controllers implemented in other languages.

2.4. Implications

BOPTEST development is motivated by the discussions in Section 2.1. Not only does providing a framework for, and set of, common test cases allow for comparison of new and improved controllers to one another, but each control algorithm tested contributes to a singular, cumulative collection of potential baseline comparisons. Furthermore, from the discussion in Section 2.2, BOPTEST uses Modelica for emulator model implementation to provide realistic controls simulation and enable leveraging of a growing ecosystem of open-source model and related computational tool development. New Modelica blocks are developed to facilitate overwriting embedded control logic at the supervisory and local-loop levels by test controllers and making available measurements. In addition, BOPTEST utilizes the Functional Mockup Interface Standard (FMI) to integrate the simulation of emulators with external controllers since FMI provides a standard interface that is supported for both compilation and simulation by a growing number of tools across industries. Finally, BOPTEST improves upon previous and existing frameworks, discussed in Section 2.3, by integrating this emulator modelling with functionality needed for control algorithm benchmarking into a rapidly and repeatably deployable computing environment accessible through a generally-accessible HTTP API using Docker.

3. Methods

With the motivation and key technologies established in Section 2.4, this section details the implementation and intended use of BOPTEST. First, Section 3.1 presents specific requirements for the framework leading to the general concept of implementation. Then, Sections 3.2–3.4 detail the implementation of three primary components: *Run-Time Environment (RTE)*, *Test Cases*, and *Key Performance Indicators (KPI)*.

3.1. Framework requirements and concept

The primary use case of BOPTEST is to evaluate the performance of control algorithms for benchmarking purposes. The framework can also be used for identifying and correcting control implementation errors. To derive specific framework development requirements for the benchmarking use case, a given building is represented in the mathematical form of a dynamic system as

$$f(\dot{x}(t), x(t), y(t), u(t), \omega(t), \theta) = 0, \quad \text{for } t \in [t_s,\ t_f], \tag{1}$$

$$x(t_s) = x_s, \tag{2}$$

where $x(\cdot)$ denotes the states, such as zone air and surface temperatures, $y(\cdot)$ are the measurements, such as zone air temperatures or equipment power, $u(\cdot)$ are the control inputs, such as actuator signals and supervisory set points, $\omega(\cdot)$ are disturbances, such as weather conditions, occupant schedules, and electricity prices, θ are time-invariant parameters, such as construction material properties, and t_s is the start time and t_f is the final time of a simulation.

The framework should then provide:

- An emulator model $f(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$ and numerical integration solver for computing the trajectories $\dot{x}(\cdot)$, $x(\cdot)$, $y(\cdot)$, $\omega(\cdot)$, the parameters θ, the initial state x_s, and a baseline control policy $u_b(x(\cdot))$ or a baseline control sequence $u_b(\cdot)$. Emulator models for different building and system types should be available in a repository.
- Ability for an external controller under test to choose $u_c \subset u_b$ and set the values of u_c at discrete time moments. Let τ_K be the set of these time moments such that $\tau_K = \{t_k\}_{k=1}^{K} \subset [t_s, t_f]$ for some $K \in \mathbb{N}$, where $\mathbb{N}$ is the set positive integers. Then, the interval $(t_{k+1} - t_k)$ is the control step, over which the values $u_c(t_k)$ are constant.
- Access by the test controller to $y(t)$ at all the $t \in \tau_K$.
- Access by the test controller to forecasts of disturbances, $\{\tilde{\omega}(t + \frac{i}{N} h)\}_{i=1}^{N}$ for some $N \in \mathbb{N}$ and $t \in \tau_K$, where $h > 0$ is the horizon of the forecast.
- Access by the test controller to $y^*(t) = Y(x(t), y(t), u(t), \omega(t))$, a set of values for KPIs calculated for some $t \in [t_s,\ t]$. Note that $y^*(t_f)$ denotes the final KPI values of the test simulation.

This abstracted use case informs the overall concept of BOPTEST, presented in Figure 1, and suggests the development of three primary components: 1) Run-Time Environment (RTE), 2) Repository of Reference Emulators, known as Test Cases, and 3) Key Performance Indicator (KPI) Calculation. The concept is that the user would use the standardized RTE to simulate a chosen building emulator with a user-specified test controller and receive a report with the values of the performance metrics evaluated based on the operation of the emulator. With this in mind, each component then has the more concrete development requirements as follows.

(1) **Run-Time Environment (RTE)**
   (a) (a)The simulation environment for high-fidelity building simulation models, also referred to as emulators, must be standardized so that results for benchmarking different test controllers are consistent. This includes the solver(s) and tolerance(s), computing environment, and other necessary software.

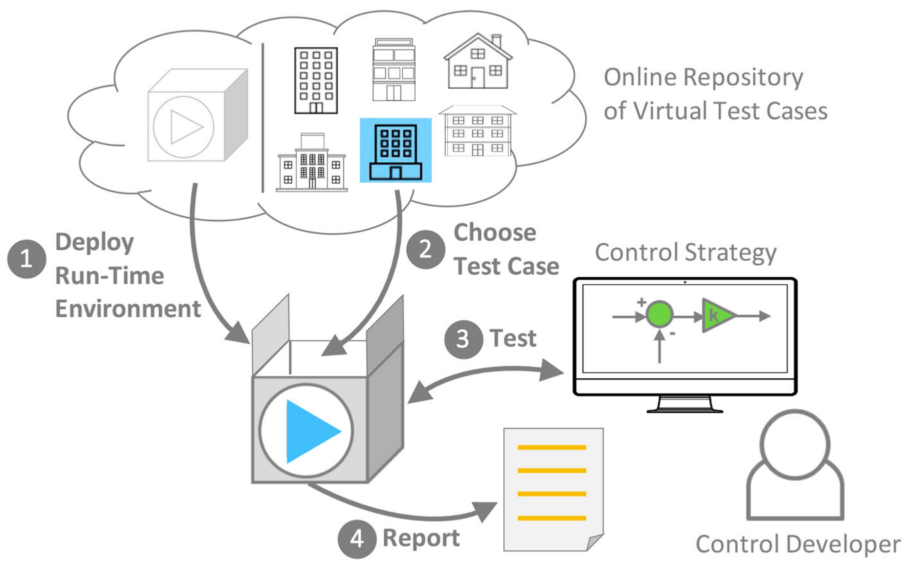


Figure 1. The BOPTEST concept.

(b) (b)Data exchange between a test controller and emulator should be facilitated by an interface that is independent of the emulator model and test controller programming languages, opening up opportunities to test diverse controller implementations with the framework.
(c) (c)Flexibility in emulator-controller synchronization times should be provided to meet different application requirements. In Option 1, the controller defines the control steps, which do not have to be regular, and the simulation should advance to the next control step when the controller returns with a control action. This is easier for controller development and allows for reproducible tests. In Option 2, the simulation should advance to the next time step according to real-time, representing a realistic building-controller interaction. Option 1 has been prioritized for implementation, while Option 2 is considered as a future extension.

(2) **Repository of Reference Test Cases**
(a) (a)Reference test cases provide emulators for co-simulation with a test controller. Emulators should simulate the dynamic evolution and relationships between physical quantities, and provide data at the time-resolution necessary for controls design and control performance assessment, including nonlinear characteristics associated with mechanical and electrical equipment such as duct and pipe flow networks, valve and damper dynamics, and variable equipment efficiencies.
(b) (b)Emulators should include complete baseline control implementations such that a test controller may overwrite these baseline control signals as needed at the supervisory and local-loop levels. This embedded control also serves as an initial benchmark for control performance.
(c) (c)Emulators should only expose for overwriting control signals that could be actuated in real building control systems, and only expose measurement data for variables that can be measured in engineering practice.
(d) (d)In addition to an emulator model, all exogenous data that defines a test case should be provided by the framework, such as weather, occupancy schedules, and energy prices. To accommodate testing of predictive controllers, this data should be made available to controllers as forecasts over a horizon of configurable length. Providing deterministic forecasts has been prioritized for implementation, while stochastic forecasts are considered as a future extension.
(e) (e)The framework should also define testing scenarios for each test case with specific parameters of interest, such as the start and end times or choice among different electricity tariffs with different price variability.

(3) **Key Performance Indicator (KPI) Calculation**
(a) (a)A set of key performance indicators (KPI) should be specified for each test case, including equations or guidelines to unambiguously quantify them.
(b) (b)The framework should calculate and report the KPIs based on simulation data collected during a controller test. This will ensure a fair and clear comparison between controllers.

(4) **Open-Source Software**
(a) (a)All software, including reference emulator models and other framework components, should be provided open-source and documented to allow for inspection of the underlying assumptions and implementation, as well as promote community-based development and maintenance.

With the development requirements defined, the following sections describe the implementation of each component.

3.2. *Run-time environment*

The Run-Time Environment (RTE), shown in Step 1 of Figure 1, is used for simulating the response of the building emulator to external control signals, calculating KPI values, and conducting other test and data management tasks. It is constructed in the form of a Docker container (2021) with an exposed RESTful HTTP API. The architecture of the RTE and user interaction is presented in Figure 2. This section is further split into two sub-sections: Section 3.2.1 describes the front-end user interaction with the API interface and will be more of interest to a reader interested in using BOPTEST. Section 3.2.2 describes the back-end contents and operation of the software within the RTE Docker container and will be more of interest to a reader interested in understanding how BOPTEST works.

3.2.1. *Front-end*

Once a user installs the Docker software on their host computing resource, they can then build and deploy the RTE with their choice of *test case* from the reference set available in the BOPTEST repository, using a single command for each action. Each test case contains an emulator model, documentation, and other necessary data for a specific building and system type, as is described in more detail in Section 3.3. Once deployed,

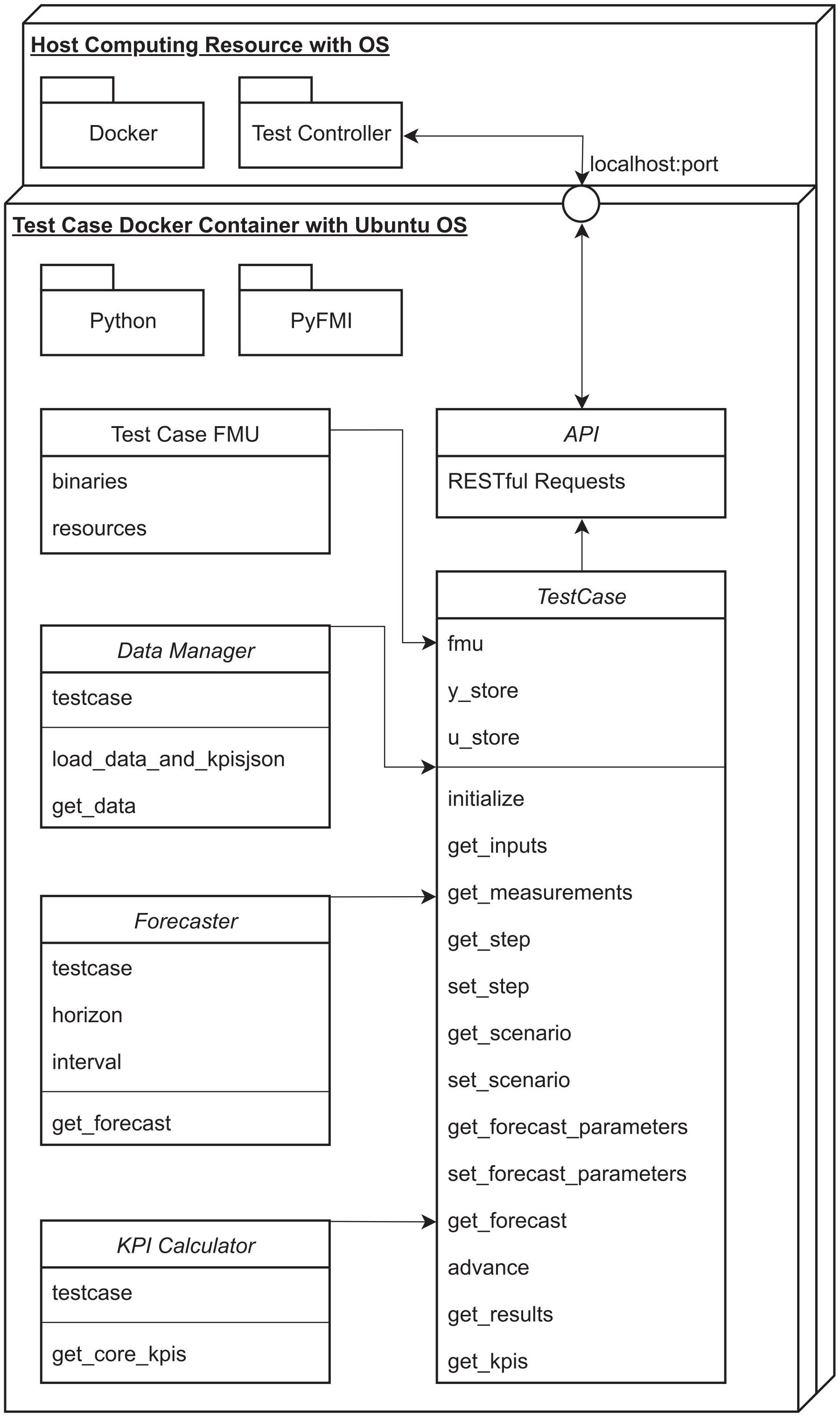


Figure 2. Architecture diagram for Run-Time Environment (RTE) showing the key classes and applications. For boxes representing classes, the name is in the top section, key attributes are in the middle section, and key methods are in the bottom section.

the API is made accessible through the network port at `localhost:5000`. This API is summarized in Table 1. In addition, Figure 3 presents the command line commands and a minimal code example using Python that interfaces a simple feedback test controller with a deployed RTE. The code is a snippet of that available in the repository in `examples/python/testcase1_scenario.py`.

The Python code in Figure 3 executes a feedback controller. The code starts with importing the necessary package for making HTTP requests as well as a class

Table 1. Run-time environment (RTE) application programming interface (API) summary.

Request endpoint	Description
GET/name	Receive test case name
GET/inputs	Receive available input point names and metadata
GET/measurements	Receive available measurement point names and metadata
GET/step	Receive active control step
PUT/step	Set control step
PUT/initialize	Initialize simulation by simulating the emulator with the built-in baseline controller for $t \in [t_s - w, t_s]$, where t_s is the specified start time and $w \geq 0$ is the initialization period
GET/scenario	Receive active test scenario parameters
PUT/scenario	Set test scenario parameters available for the test case such as time period, which invokes `/initialize` with specific t_s, w, and end time t_f, and electricity price profile
GET/forecast_parameters	Receive active boundary condition forecast parameters
PUT/forecast_parameters	Set boundary condition forecast parameters such as horizon and time interval
GET/forecast	Receive boundary condition forecast from current control step
POST/advance	Advance simulation one control step with control input data
GET/kpi	Receive core KPI values from start time through current control step
PUT/results	Receive simulated data trajectories for a specified measurement or input point over a specified time period

representing the test controller. Next, high-level parameters for the script are defined. The `url` is the location of the RTE interface port, `scenario` defines a specific testing time period and electricity price profile available within the test case, described in Section 3.3, and `step` defines the control step to be used during the test. Then, the `/measurements` and `/inputs` API endpoints are used to receive information about the available measurement and control input points, including names, units, descriptions, and minimum and maximum values. After this, the test is set up by first using the `/step` API endpoint to set the control step and then the `/scenario` API endpoint to set a specific testing time period and electricity price profile. Then, the test is executed using a `while` loop that first computes a control signal based on measurement data and then uses the `/advance` API endpoint to advance the simulation forward one control step with the computed control signal and to receive current-time measurements. Once the testing period is completed as defined by the specified scenario, and indicated by an empty measurement return, the core KPIs, calculated for every test case and described in Section 3.4, are retrieved using the `/kpi` API endpoint and the simulation trajectory for the entire testing period is retrieved for the point named `'TRooAir_y'` (room air temperature) using the `/results` API endpoint. Note that once the end of scenario time is reached, the simulation can no longer be advanced until the start time is reset using the `/initialize` or `/scenario` API endpoints.

A few additional notes are as follows. This example demonstrated an interface for a feedback controller. Advanced control algorithms, such as MPC and RL, will also need to use the `/forecast` API endpoint for disturbance prediction and additional usage of the `/results` API endpoint state history. An example usage of BOPTEST for testing an MPC strategy is demonstrated in Section 4. An RL controller may also need additional usage of the `/kpis` API endpoint to update the reward function at each control step. Simulations with the baseline control implemented with the emulator model can be run in the same way as presented, without passing any control signal data during the usage of `/advance` API endpoint. Simulating the test case outside of the defined scenario period, for instance to obtain data for MPC or RL model identification, can be done using the `/initialize` API endpoint. Finally, while Python is used in this example, any language capable of making HTTP RESTful requests can be used.

3.2.2. *Back-end*

As shown in Figure 2, the RTE Docker container utilizes the Ubuntu operating system and required dependency software applications. We highlight here the Python interpreter and PyFMI (Andersson, Åkesson, and Führer 2016), a Python package for the simulation of FMUs. Described in more detail in Section 3.3, test case emulator models and associated boundary condition data are packaged as FMUs. The FMUs may be generated by different Modelica environments.

Besides the application dependencies and test case FMU, the other objects within the RTE are singular instances of Python classes responsible for providing functionality and data management. The primary class is `TestCase`, which is responsible for loading and simulating the FMU, storing result data in memory, and getting and setting data as needed by the client or simulation process. `TestCase` utilizes three helper classes to perform its duties. The first is `Data Manager`, which is responsible for loading and fetching boundary condition data packaged within a test case FMU. The second is `Forecaster`, which is responsible for providing forecasts of boundary condition data defined by a horizon and interval. The third is `KPI Calculator`, which is responsible for calculating the core KPIs defined in Section 3.4. Finally, the class `API` exposes the API of `TestCase`, and in general the RTE, to a client through a network port and in the form of HTTP requests. A more detailed description of the API can be found in

```
# Build RTE Docker image with specified test case
$ make build TESTCASE=<test_case_name>
# Launch RTE and expose REST API on 127.0.0.1:5000
$ make run TESTCASE=<test_case_name>
```

```
# GENERAL PACKAGE IMPORT
import requests
import numpy as np
# TEST CONTROLLER IMPORT
from controllers import pid
# SET TEST PARAMETERS
# Set URL for test case
url = "http://127.0.0.1:5000"
# Set testing scenario
scenario = {"time_period":"test_day",
            "electricity_price":"dynamic"}
# Set control step
step = 300
# GET TEST INFORMATION
# Get test case name
name = requests.get("{0}/name".format(url)).json()
# Get inputs available
inputs = requests.get("{0}/inputs".format(url)).json()
# Get measurements available
measurements = requests.get("{0}/measurements".format(url)).json()
# RUN TEST CASE
# Set control step
requests.put("{0}/step".format(url), data={"step":step})
# Set test case scenario
y = requests.put("{0}/scenario".format(url),
                 data=scenario).json()["time_period"]
# Record test start time
start_time = y["time"]
# Simulation Loop
while y:
    # Compute control signal
    u = pid.compute_control(y)
    # Advance simulation with control signal
    y = requests.post("{0}/advance".format(url), data=u).json()
# GET RESULTS
# Get KPIs
kpi = requests.get("{0}/kpi".format(url)).json()
# Get zone temperature over test period
args = {"point_name":"TRooAir_y",
        "start_time":start_time,
        "final_time":np.inf}
res = requests.put("{0}/results".format(url), data=args).json()
```

Figure 3. Command line commands for building and running the RTE with a specified test case (top) and example controller interface with RTE using Python (bottom).

Table 1. Upon deployment of the RTE container with a selected test case FMU, `TestCase` loads and initializes the FMU to a default start time t_s using PyFMI, instantiates `Data Manager`, `Forecaster`, and `KPI Calculator` classes, loads the names and metadata of available control input and measurement points, sets the control step, forecast parameters, and KPI calculation scenario to the default configuration, sets the FMU simulation solver and tolerance, and initializes in-memory data structures for the storage of simulation results. Once deployed, the RTE is idle until a request is made through the API, in which case it is handled and then returned to an idle state.

The operation of the RTE for the basic use case presented in the minimal code example in Figure 3 is explained with a swimlane diagram in Figure 4. Additionally, this swimlane diagram shows usage of the `/forecast` and `/results` API endpoints that are needed for predictive controllers. Some additional notes on the back-end operation of the RTE in this use case are as follows. Upon receiving a request to initialize a simulation, `TestCase` uses the PyFMI `simulate` method to initialize the FMU by simulating it for $t \in [t_s - w, t_s]$ using the embedded baseline controller. Here, t_s and $w \geq 0$ are defined explicitly by the user if they use `/initialize` or by the framework if they use `/scenario`. Upon

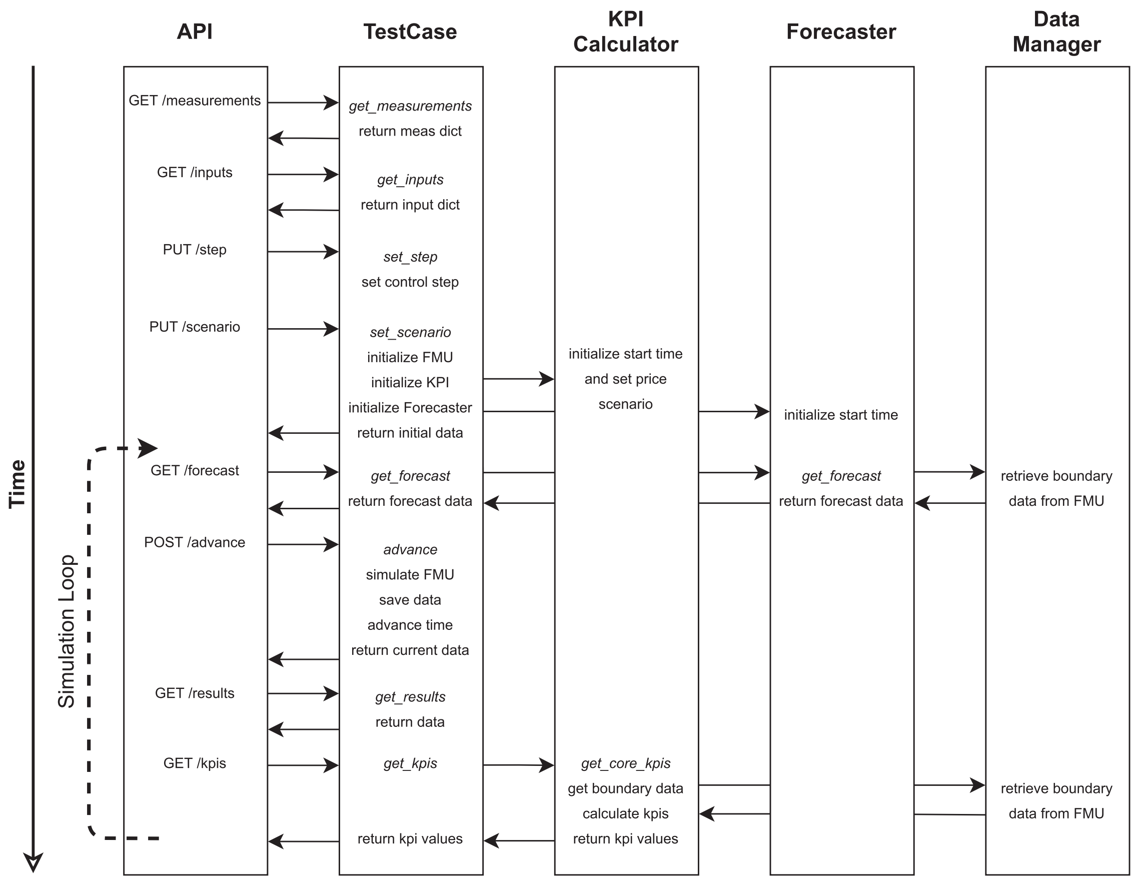


Figure 4. Swimlane diagram for basic use case of Run Time Environment (RTE) to run a test simulation. Italics indicate a specific method within the indicated class.

completion of this initialization, the trajectories of control input and measurement points defined by the test case are saved, the current time values for measurement points are returned to the user, and the RTE is again idle. Upon usage of the `/advance` API endpoint to advance the simulation one control step, `TestCase` uses the PyFMI `simulate` method to advance the FMU simulation forward with control input values if provided. Then, the trajectories of control input and measurement points defined by the test case are saved at 30 second time resolution and without event points using the appropriate simulation options within PyFMI. The number of communication points within the PyFMI simulation options at each simulation step corresponds to the current control step size. Note that the choice of storing results at a fixed time step without event points ensures consistency for KPI calculation and data reporting to a user such that both are independent of the solver time step, which can vary depending on control action, control step size, and simulation events. The 30 second time step was chosen as a trade-off between the accurate representation of data trajectories and the low memory requirements and simulation time. A user may still choose a control step to be less than 30 seconds. In this case, the results are stored at the resulting time resolution of the solver. Once the advance is complete, the current time values for measurement points are returned to the user and the RTE is again idle. If the end of the scenario time period has been reached, empty measurements are returned and an internal flag indicates the end of a scenario. Further simulations are prevented unless the emulator is initialized to a new start time using the `/initialize` API endpoint or a new scenario time period is requested using the `/scenario` API endpoint. Upon a request using the `/results` API endpoint, `TestCase` returns the stored time trajectory for the point and time period specified in the request. Upon a request using the `/kpis` API endpoint, `TestCase` calls upon `KPI Calculator` to utilize the stored measurement point trajectories and any necessary boundary condition data from the Test Case FMU to calculate and return the values of the core KPIs described in Section 3.4. Here, trapezoidal numeric integration is applied to the saved data.

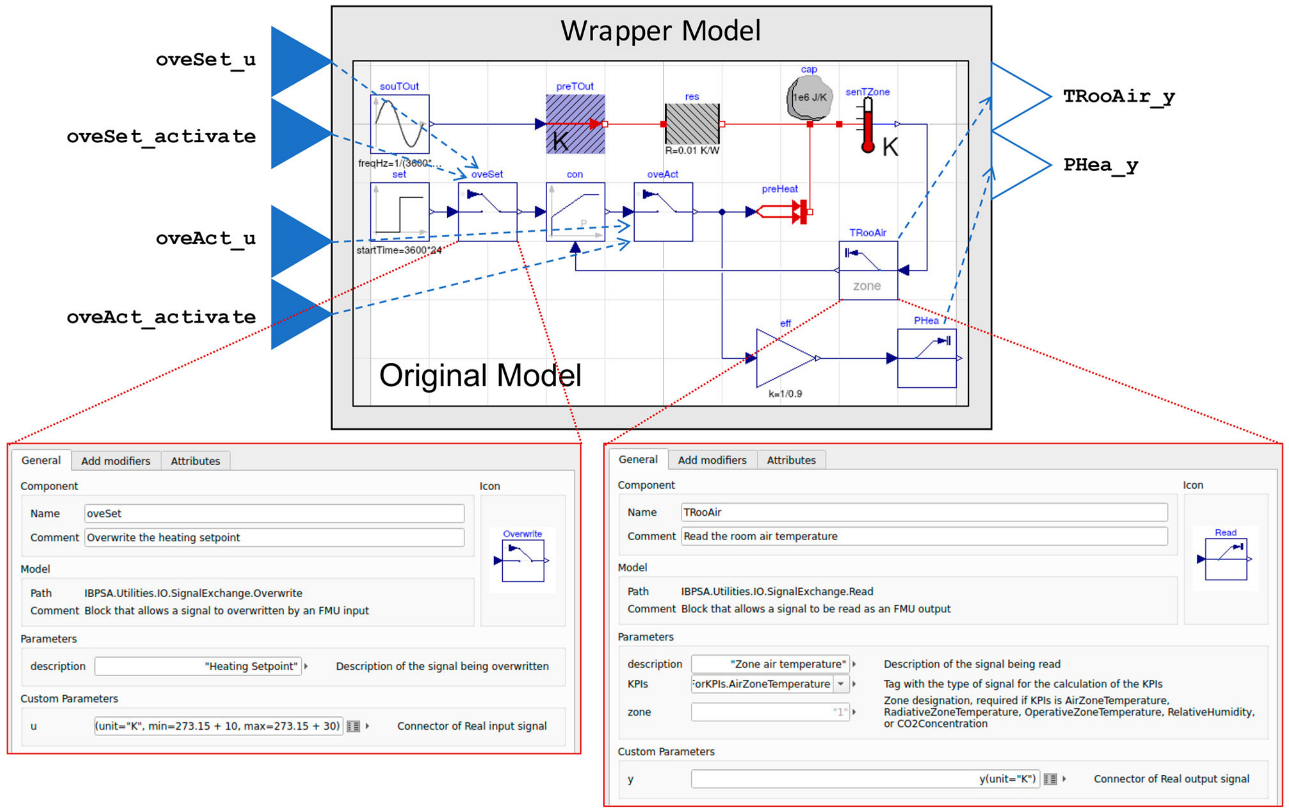
3.3. Test cases

Test cases contain all of the information related to a specific virtual building to be used for testing. A test case repository is available for use in the RTE as shown in Step 2 of Figure 1. A single test case is composed of an FMU and standardized documentation in the form of a `.html` file and necessary image files. The FMU contains the building emulation model, a file used to map model outputs to KPI calculation called `kpis.json`, a file used to define test scenario time periods called `days.json`, a file used to specify default configuration values when the test case is initialized within the RTE called `config.json`, and boundary condition data in the form of `.csv` files.

The emulator models themselves are implemented in Modelica. Two new 'signal exchange' Modelica blocks have been developed in the IBPSA Modelica Library (Wetter et al. 2019) to enable overwriting of particular control signals by a test controller, `IBPSA.Utilities.IO.SignalExchange.Overwrite`, and reading of particular measurement signals by the test controller, `IBPSA.Utilities.IO.SignalExchange.Read`. The overwrite block is implemented as a switch that toggles between two input signals, one being from the Modelica model itself and one being from the external test controller. Overwrite blocks can be used for supervisory or actuator control signals. Activating the switch to use the test controller value is controlled by the test controller, through usage of the `/advance` API endpoint, allowing the test controller to choose which signals it overwrites and which it leaves to the embedded control. This enables, for instance, a test controller to optimize supervisory set points and leave local-loop control to the controllers embedded in the Modelica model, much like in the case of many field demonstrations. All activation is false by default. Configuration of the overwrite block in the emulator model takes a description, minimum and maximum values, and unit. Configuration of the read block takes a description, unit, whether or not the signal is needed for KPI calculation, and depending on the need for KPI calculation, a zone identifier. If the measurement is needed for KPI calculation, an enumeration parameter is available to choose from a number of KPI IDs, which help the internal process of KPI calculation described later. A Python module, `parser.py` has been developed for test case developers to compile the model into an FMU and make use of the overwrite and read blocks. It does so by first writing a new top-level Modelica model (the 'wrapped' model) in which the original model is instantiated, unique activation and signal inputs are added and connected to corresponding overwrite blocks, and unique outputs are added and connected to corresponding read blocks. Then, the Python module compiles this 'wrapped' model into an FMU. Figure 5 shows the use and configuration of signal exchange blocks in a simple example model of a 1R1C envelope model with heater and feedback thermostat control.

The `kpis.json` file is used by the RTE to map which model outputs are to be included in the calculation of KPIs. The map has the structure `kpi id`: [`output id`]. Here, the `kpi id` is an identifier for a particular KPI calculation and [`output id`] is a list of model outputs that are needed to calculate that KPI. For example, consider a KPI of total energy use. Then, one `kpi id` is `ElectricPower`, and the corresponding [`output id`] will contain a list of all model output names that should be used in the calculation of total electric power (e.g. supply fan power, pump power, heat pump power, etc.). Other available KPI IDs, for example used for total energy use calculation, include `GasPower` and `DistrictHeatingPower`. For calculation of KPIs in multi-zone buildings that require zone identifiers to properly match zone-specific limits with zone-specific measurements, the `kpi id` can contain such a zone identifier. For example, consider a KPI of total thermal discomfort. Then, one `kpi id` is `AirZoneTemperature[north]`, and the corresponding [`output id`] will contain the model output name for the zone air temperature of zone `north`. In addition to compiling the emulation model FMU, `parser.py` also uses information in the read blocks to generate the `kpis.json` file.

The `days.json` file is used by the RTE to map the names of test scenario time periods to the actual simulation time periods. The map has the structure `time period name`: `day number`. Here, the `time period name` is a string identifier for the name of the test scenario time period and `day number` is an integer specifying a reference day number for determining the simulation period. For example, for the `'peak_heat_day'` scenario time period, the reference day number corresponds to the day with peak heating load, and the RTE uses this value to define the starting and ending time of the corresponding test simulation. Currently, all scenario time periods are defined by a two-week testing period centred on the reference day along with a one-week warm-up period using the controllers embedded within the emulator model for state initialization at the start of the two-week testing period. The available scenario time periods for each test case are defined within their respective documentation. Currently, peak heating or cooling, typical heating or cooling, and mixed heating and cooling time period scenarios are consistently defined across available test cases. Defining these scenarios separately as two-week periods allows for highlighting differences in controller performance that may occur under these different operating conditions. The `day number` associated with peak is the day with maximum 15-minute system heating or cooling load. For typical, it is the day with the maximum 15-minute system



General | Add modifiers | Attributes

Component

Name: oveSet

Comment: Overwrite the heating setpoint

Icon: Overwrite

Model

Path: IBPSA.Utilities.IO.SignalExchange.Overwrite

Comment: Block that allows a signal to overwritten by an FMU input

Parameters

description: "Heating Setpoint" — Description of the signal being overwritten

Custom Parameters

u: (unit="K", min=273.15 + 10, max=273.15 + 30) — Connector of Real input signal

General | Add modifiers | Attributes

Component

Name: TRooAir

Comment: Read the room air temperature

Icon: Read

Model

Path: IBPSA.Utilities.IO.SignalExchange.Read

Comment: Block that allows a signal to be read as an FMU output

Parameters

description: "Zone air temperature" — Description of the signal being read

KPIs: orKPIs.AirZoneTemperature — Tag with the type of signal for the calculation of the KPIs

zone: "1" — Zone designation, required if KPIs is AirZoneTemperature, RadiativeZoneTemperature, OperativeZoneTemperature, RelativeHumidity, or CO2Concentration

Custom Parameters

y: y(unit="K") — Connector of Real output signal

Figure 5. Example Modelica model of a 1R1C envelope model with heater and feedback thermostat control using two instances of `IBPSA.Utilities.IO.SignalExchange.Overwrite` that would enable a test controller to overwrite the heating zone temperature set point (`oveSet_u` and `oveSet_activate`) or heater power (`oveAct_u` and `oveAct_activate`) and two instances of `IBPSA.Utilities.IO.SignalExchange.Read` that would enable a test controller to read the zone air temperature (`TRooAir_y`) and gas power consumed by the heater (`PHea_y`). Example parameterization for overwrite and read blocks are shown in lower images.

heating or cooling load that is closest from below to the median of all 15-minute maximum heating or cooling loads of all days in the year. For mixed, it is a day, d_{mix}, with both significant heating and cooling loads as defined by

$$d_{\text{mix}} = \arg\max_{d \in D}\Big(\big(H(d) + C(d)\big) - \big|H(d) - C(d)\big|\Big), \tag{3}$$

where $H(d)$ and $C(d)$ are the total heating and cooling loads for day d in all days of the year D.

Boundary condition data are represented as timeseries in `.csv` files and ultimately utilized for providing forecasts and calculating KPIs. They include weather, energy pricing, CO_2 emission factors, occupancy and internal gain schedules, zone air or operative temperature comfort limits, and zone air quality comfort limits. While the actual data is specific to the test case (e.g. geographic location and building type), the column names of the data are standardized so as to enable use by the RTE in a generic way. Currently, different scenarios are available for electricity pricing, including constant, dynamic, and highly dynamic price profiles. The available electricity price scenarios for each test case are defined within the test case documentation.

Once the Modelica model, `kpis.json`, `days.json`, `config.json`, and boundary condition data are created, `parser.py` can be used to compile the Modelica model into the wrapped FMU and move the `kpis.json`, `days.json`, `config.json`, and specified boundary condition `.csv` files into the 'resources' directory of the FMU. This completes the packaging of a test case FMU. Each test case FMU is accompanied by documentation in the form of a `.html` file describing the building and HVAC system design, baseline control implementation, additional relevant system design, input and output points list and meta-data, important modelling assumptions, and information about sources of energy pricing and CO_2 emission factors. A template has been developed and is utilized by all test case developers for consistency across test cases. Finally, the test cases are collected in a test case repository, where they are available for users and also subject to continuous integration testing. The emulator models are provided in the repository along with specifications of the dependent Modelica library versions as corresponding GitHub commits so that test case versions can be properly defined and maintained.

3.4. Key performance indicators

Key Performance Indicators (KPI) are metrics used to evaluate the performance of a test controller. As presented in Section 2.1, there are many KPIs to consider. However, it is important for BOPTEST to calculate a core set of KPIs in order to facilitate a fair comparison. Some are easy to quantify objectively, such as energy use and cost, while some are more difficult, such as controller data requirements or implementation effort. In addition, the more KPIs that are utilized, the harder it is to compare control strategies and some KPIs may be more applicable to certain tests cases than others. Therefore, the core KPIs included in BOPTEST aim to cover the primary aspects of control performance evaluation, be objectively quantifiable, and not overwhelm the comparison process. The RTE calculates these core KPIs for every test case using measurement data saved during a test simulation and boundary condition data compiled into the test case FMU. These KPIs are reported to a user as shown in Step 4 of Figure 1. Optionally, users can also implement customized KPIs and evaluate them using data from the `/results` API endpoint. As those KPIs may be only applicable to their specific controller evaluation, they are not calculated by the framework.

The core KPIs calculated by the framework are defined in the sections below. Note that floor areas used for normalization are calculated according to the definitions of the Commercial Buildings Energy Consumption Survey (CBECS terminology 2021) and Residential Energy Consumption Survey (2015 RECS square footage methodology 2021).

3.4.1. Thermal discomfort

The *Thermal Discomfort*, reported with units of $[K\,h]$ per zone, defines the cumulative deviation of zone temperatures from upper and lower comfort limits that are predefined within the test case FMU for each zone, averaged over all zones. Air temperature is used for air-based systems and operative temperature is used for radiant systems. The *Thermal Discomfort* is calculated as

$$K_{\text{tdis}} = \frac{\sum_{z=1}^{N} \int_{t_s}^{t_f} \big(\max(T_z(t) - T_{z,\text{coo}}(t), 0) + \max(T_{z,\text{hea}}(t) - T_z(t), 0) \big)\, \mathrm{d}t}{N}, \quad (4)$$

where $T(t)$ is the zone temperature of zone z, $T_{z,\text{coo}}(t)$ and $T_{z,\text{hea}}(t)$ are the cooling and heating set points of zone z, N is the number of zones, and t_s and t_f are the start time and the end time of the evaluation period.

3.4.2. Indoor air quality (IAQ) discomfort

The *IAQ Discomfort*, reported with units of [ppm h] per zone, defines the extent that the CO_2 concentration levels in zones exceed bounds of the acceptable concentration level, which are predefined within the test case FMU for each zone, averaged over all zones. The *IAQ Discomfort* is calculated as

$$K_{\text{idis}} = \frac{\sum_{z=1}^{N} \int_{t_s}^{t_f} \max(C_z(t) - C_{z,\max}(t), 0)\, \mathrm{d}t}{N}, \quad (5)$$

where $C_z(t)$ is the CO_2 concentration of zone z at time t and $C_{z,\max}(t)$ is the higher bound for the acceptable CO_2 concentration of zone z.

3.4.3. Energy use

The *Energy Use*, reported with units of [kWh/m^2], defines the HVAC energy usage and is calculated as

$$K_{\text{ener}} = \frac{\sum_{j\in J} \sum_{i\in I_j} \int_{t_s}^{t_f} P_{ij}(t)\, \mathrm{d}t}{A}, \quad (6)$$

where $P_{ij}(t)$ is the power measurement by the device i with energy source j at time t, J is the set of all energy sources, such as electricity and gas, I_j is the set of all devices with energy source j, and A is the total floor area of the building.

3.4.4. Cost

The *Cost*, reported with units of [\$/m^2] or [€/m^2], defines the cost associated with the HVAC energy usage. The *Cost* is calculated as

$$K_{\text{cost}} = \frac{\sum_{j\in J} \sum_{i\in I_j} \int_{t_s}^{t_f} p_j^{\tau}(t) P_{ij}(t)\, \mathrm{d}t}{A}, \quad (7)$$

where $P_{ij}(t)$ is the power measurement by the device i with energy source j at time t, J is the set of all energy sources, I_j is the set of all equipment with energy source j, $p_j^{\tau}(t)$ is the price for energy source j at time t with a tariff τ, and A is the total floor area of the building. The $p_j^{\tau}(t)$ are predefined within the test case FMU. Note that demand costs, typically calculated for commercial customers in the US on a monthly basis in addition to energy costs, are not included in this core KPI.

3.4.5. Emissions

The *Emissions*, reported with units of [kg CO_2/m^2], defines the CO_2 emissions from the HVAC energy usage and is calculated as

$$K_{\text{emis}} = \frac{\sum_{j\in J} \sum_{i\in I_j} \int_{t_s}^{t_f} e_j(t) P_{ij}(t)\, \mathrm{d}t}{A}, \quad (8)$$

where $P_{ij}(t)$ is the power measurement of the equipment i with energy source j at time t, J is the set of all energy sources, I_j is the set of all devices with energy source j, $e_j(t)$ is the emissions factor for energy source j at time t, and A is the total floor area of the building. The $e_j(t)$ are predefined within the test case FMU.

3.4.6. Computational time ratio

The *Computational Time Ratio* defines the average ratio between the controller computation time and the test simulation control step. The controller computation time is measured as the time between two emulator simulations. The *Computational Time Ratio* is calculated as

$$K_{\text{timr}} = \frac{\sum_{k=1}^{N} \frac{\delta T_k}{\delta t_k}}{N}, \tag{9}$$

where δT_k is the elapsed controller computational time at step k, N is the number of control steps, and δt_k is the interval of control step k.

4. Demonstration

4.1. Test case description

This section demonstrates the capabilities of BOPTEST as in version `v0.1.0` of the repository. The test case used is a single-zone residential building with a hydronic radiant floor heating system and heat pump, known as BESTEST Hydronic Heat Pump in the BOPTEST repository. This is a high-fidelity, yet relatively simple, test case that allows for demonstrating the functionality of BOPTEST described in Section 3 and capability for test controller benchmarking. The test case represents a residential dwelling located in Brussels, Belgium. The weather data, the boundary data, and the heating system are characteristic of a building of this location. Particularly, the building is exposed to an oceanic climate with a mild Winter and a cool Summer, and the building envelope materials are based on the BESTEST case 900 building (Judkoff and Neymark 1995). The building is modelled as a single zone with a rectangular floor plan of 12 m by 16 m (192 m^2) and a height of 2.7 m, with a south-oriented facade that has several windows with a total surface area of 24 m^2. The insulation levels for each of the elements of the building envelope are summarized in Table 2. The thermal mass of indoor walls is modelled assuming that there are approximately 12 rooms in the building.

A family of five members inhabits the building and follows a typical residential weekly schedule, where the building is occupied before 7:00 h and after 20:00 h every weekday and full time during weekends. The comfort range defines the boundaries of the indoor operative zone temperature and is 21–24°C during occupied hours and 15–30°C otherwise.

Table 2. Material layers (outside to inside) of the emulator building envelope.

Wall type	Description	d [mm]	λ [W/mK]	c [J/(kgK)]
Outer wall	Wood siding	9	0.14	900
	Insulation	61.5	0.04	1400
	Concrete block	100	0.51	1000
Floor	Concrete	150	1.4	840
	Insulation	200	0.02	1470
	Screed	50	0.6	840
	Tile	10	1.4	840
Ceiling	Roof deck	19	0.14	900
	Fiber glass	111.8	0.04	840
	Plaster board	10	0.16	840
Fenestration	Glass	3.175	1.06	750
	Air	13	0.0241	1008
	Glass	3.175	1.06	750

Note: d is the layer thickness, λ is the thermal conductivity, and c is the specific thermal capacity of the material.

The heating system consists of an air-to-water modulating heat pump of 15 kW, which is coupled to a floor heating system. The embedded baseline controller for heat pump modulation uses a PI logic to track the operative zone temperature. This controller, while reactive, has been carefully tuned to provide adequate indoor comfort without excessive energy use. The floor heating circulation pump and evaporator fan operate only when the heat pump is on. No cooling is considered in this test case, which is justifiable in a Belgian climate.

The envelope model of this emulator is implemented using the IDEAS Modelica library (Jorissen et al. 2018b) and integrates, among others: dynamic zone air temperature, air infiltration assuming a constant n_{50} value, dynamic heat transfer through walls, floor, ceiling and fenestration, and nonlinear conduction, convection and radiation models. The main model components used from the library are: `IDEAS.Buildings.Validation.Cases.Case900Template` for the building zone, `IDEAS.Fluid.HeatExchangers.RadiantSlab.EmbeddedPipe` for the floor heating system, and `IDEAS.Fluid.HeatPumps.ScrollWaterToWater` for the heat pump. The latter has been configured to use air media through the evaporator circuit instead of water. Additionally, the heat pump model parameters have been fitted to manufacturer performance data following the calibration procedure explained by Cimmino and Wetter (2017). Air humidity condensation and start-up losses are not considered.

This demonstration examines two testing periods of two weeks each, that represent peak and typical heating periods. These periods are January 17th–31st and April 19th–May 3rd, respectively. For each of them, three price scenarios are considered, namely: constant, dynamic, and highly-dynamic. The constant price scenario uses a constant price of 0.2535 €/kWh. For the dynamic price scenario the test case uses a dual-rate that alternates between an on-peak price of 0.2666 €/kWh between 7:00 h and 22:00 h, and an off-peak price of 0.2383 €/kWh otherwise. The highly dynamic price scenario uses the Belgian day-ahead energy prices as determined by the BELPEX wholesale electricity market in the year 2019. All pricing scenarios include the same constant component for transmission fees and taxes as determined by Eurostat and obtained from European Commission, Directorate-General for Energy (2020). Particularly, this component is 0.2 €/kWh for the assumed test case location, and only the rest of the price is the energy cost from the utility company or the wholesale market.

An MPC is formulated and implemented to operate the thermal systems for each of the scenarios defined above. In all cases, the objective of the controller is to maximize comfort with minimum operational cost. Additionally, the effect of using different control steps and prediction horizon periods for the same MPC formulation is investigated.

4.2. MPC description

The first step towards the implementation of an MPC is to gather data for system identification or model calibration. Both, building metadata and building operational data are useful for this task. In BOPTEST, the metadata can be obtained from the documentation folder of each test case. Operational data can be obtained by interacting with the test case with the `/initialize` and sequential `/advance` API calls using either the test case baseline controller or any other arbitrary controller to overwrite control actions during a so-called training period. In both cases, the training data can be collected using the `/results` API call at the end of the training period. Notice that the training period should not overlap any of the testing periods envisioned by the test case.

For this particular example, we gather four weeks of operational data with the baseline controller to train and validate a grey-box building model for use within the MPC. The Grey–Box Toolbox (Coninck et al. 2016) is used to prototype the model and train its parameters. The selected model structure is shown in Figure 6. This model has four inputs: heat pump modulation signal for compressor frequency u_{hp}, ambient temperature T_a, solar direct normal irradiation $\dot{Q}_{rad}$, and internal occupancy gains $\dot{Q}_{occ}$. The model further consist of five states: zone operative temperature T_z, internal temperature of the indoor thermal mass like internal walls T_i, envelope wall temperature T_w, floor temperature T_f, and water temperature from the heat pump condenser T_c. The parameters to be estimated for the building envelope are the thermal capacitances for zone air, wall, internal, condenser, and floor thermal, respectively C_z, C_w, C_i, C_c, and C_f, the thermal resistors for the wall, infiltration, internal, condenser, and floor, respectively $R_{w,1}$, $R_{w,2}$, R_{inf}, R_i, R_c, and R_f, and the solar transmittance, gA.

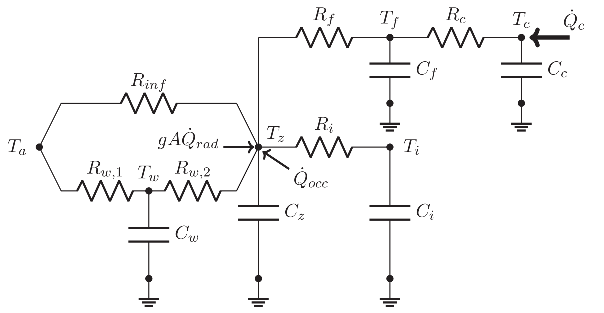


Figure 6. Grey–Box model of the building envelope that is used for control optimization.

The heat pump is modelled as

$$\dot{Q}_c = (a_c + b_c(T_c - T_{c,n}) + c_c(T_a - T_{a,n}))k_c u_{hp} \tag{10a}$$

$$\dot{Q}_e = (a_e + b_e(T_c - T_{c,n}) + c_e(T_a - T_{a,n}))k_e u_{hp} \tag{10b}$$

$$P_{hp} = \dot{Q}_c - \dot{Q}_e \tag{10c}$$

$$COP = \frac{\dot{Q}_c}{P_{hp}}, \tag{10d}$$

where $\dot{Q}_c$ is the condenser thermal power, $\dot{Q}_e$ is the evaporator thermal power, and P_{hp} is the electrical power used by the compressor. Nominal constant values of condenser water temperature $T_{c,n}$ and of ambient temperature $T_{a,n}$ are arbitrarily chosen to compute thermal powers from temperature differences instead of from absolute temperatures. The heat pump modulation signal for compressor frequency is assumed to depend linearly on the thermal power. The model is therefore bilinear since $\dot{Q}_c$ is linear in both T_c and u_{hp}. The condenser and evaporator proportionality factors k_c and k_e, and the other heat pump parameters a_c, b_c, c_c, a_e, b_e, and c_e, are estimated using dynamic training data. The training data are generated by simulation using the baseline controller during the first two weeks of February. The data collected for the next two weeks in February are used for cross-validation. Figure 7 shows the open loop simulation performance of the model that has been trained to fit zone operative temperature T_z, condenser thermal power $\dot{Q}_c$, and compressor electrical power P_{hp}. Measurement data of these variables have been gathered for system identification. The Root Mean Square Error (RMSE) for training and validation respectively for zone operative temperature are 0.17 and 0.27°C, for compressor electric power are 0.21 and 0.27 kW, and for condenser thermal power are 0.30 and 0.31 kW.

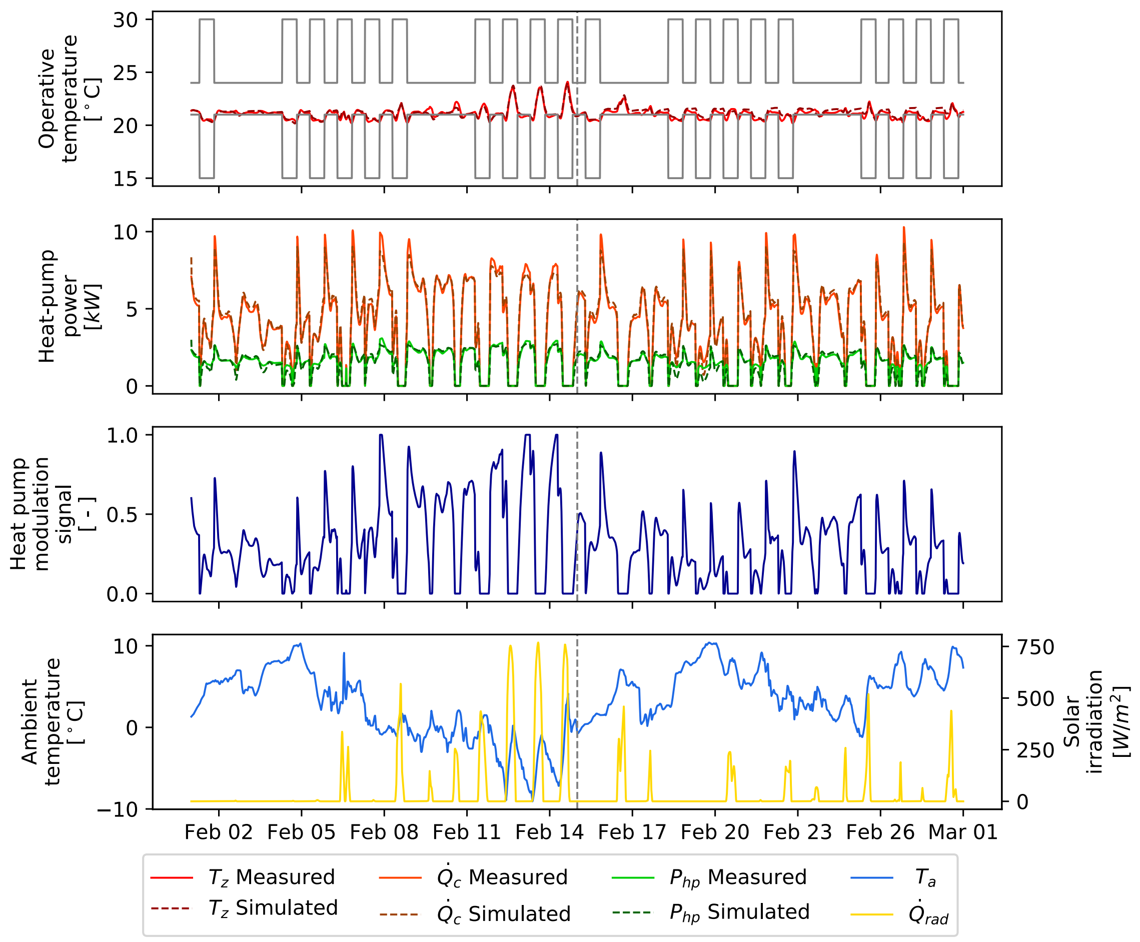


Figure 7. Open loop model simulation for two weeks of training data and two weeks of validation. The training period is at the left of the vertical grey dashed line and the validation period at the right. The inputs are presented in the bottom two plots and the simulated outputs are presented in the top two plots and compared with the measured data.

Finally, the fan and pump electricity use is modelled as equal to their nominal values $P_{\text{fan},n}$ and $P_{\text{pum},n}$, if $u_{hp} > 0$, and 0 otherwise, indicated in Equations (11) as

$$P_{\text{fan}} = \begin{cases} P_{\text{fan},n}, & u_{hp} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{11a}$$

$$P_{\text{pum}} = \begin{cases} P_{\text{pum},n}, & u_{hp} > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{11b}$$

For every control step of each testing scenario, the model is used in an optimization problem to compute the heat pump modulation signal that minimizes operational cost and discomfort in the building over a prediction horizon. The MPC formulation is

$$\min_{u_{hp}} \int_{t=t_i}^{t_h} (p^{e,\tau}(P_{hp} + P_{\text{fan}} + P_{\text{pum}}) + w\delta^{T_z})\,\text{dt} \tag{12a}$$

$$\dot{T}_z, P_{hp}, P_{\text{fan}}, P_{\text{pum}} = F(u_{hp}, \dot{Q}_{\text{rad}}, \dot{Q}_{\text{occ}}, T_a, T_z, T_c, T_f, T_i, T_w) \tag{12b}$$

$$\underline{T}_z - \delta^{T_z} \le T_z \le \bar{T}_z + \delta^{T_z} \tag{12c}$$

$$\delta^{T_z} \ge 0 \tag{12d}$$

$$0 \le u_{hp} \le 1. \tag{12e}$$

In Equations (12), the notation that indicates the time dependency has been omitted for brevity because all the variables are time dependent with the exception of the weighting factor w. This optimization problem is solved for the time period between the initial time t_i and the end of the prediction horizon t_h. In the objective, the terms accounting for electrical power, i.e. P_{hp}, P_{fan} and P_{pum}, are summed and multiplied by the electricity price $p^{e,\tau}$ where the tariff τ corresponds to a constant, dynamic, or highly dynamic tariff depending on the selected pricing scenario. The weight w is scaled to account for the different orders of magnitude between the power terms and the discomfort δ^{T_z}, which is defined as the deviation of the zone operative temperature out of the comfort range. This comfort range is determined by the lower comfort bound denoted $\underline{T}_z$ and the upper comfort bound denoted $\overline{T}_z$. The function $F(\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)$ represents the system model comprising the building envelope model represented in Figure 6 and the set of Equations (10). The fan and pump power P_{fan} and P_{pum} are approximated by a sharp logistic function as a smooth representation of a step over the origin with u_{hp} as an independent variable. This avoids the use of integer optimization variables and allows for the use of nonlinear programming solvers.

The first element from the trajectory result from each optimization is used in every `/advance` call to overwrite the modulation signal of the heat pump operation for the next control step. No other variables need to be overwritten since the baseline controller already dictates the operation of the circulation pump and the heat pump evaporator fan when the heat pump is on.

The MPC implementation of this example relies on the JModelica toolbox (Åkesson et al. 2010). This toolbox allows for formulating optimization problems using the Optimica language (Åkesson 2008), an extension of Modelica that includes optimization constructs like constraints and objective functions. Particularly, the nonlinear JModelica MPC module developed by Axelsson, Magnusson, and Henningsson (2015) is used here and has been extended to enable mutable external data, a feature that is missing from the JModelica MPC module but that is required to handle the external time-varying disturbances associated with the weather and occupancy conditions. We refer to Axelsson, Magnusson, and Henningsson (2015) for the details of the MPC implementation.

The MPC module is combined with the unscented Kalman filter of the JModelica toolbox for state estimation. The specific state estimation algorithm is the non-augmented version described in Sun, Li, and Wang (2009), and the sigma points are chosen according to Wan and Merwe (2000). This state observer is particularly suitable to provide state estimates of Modelica models since it benefits from the simulation capabilities of PyFMI to obtain the distribution of states from empirical simulations rather than analytical computation. For this, the `/results` API call is used at each step to retrieve historic data of observed points.

Boundary condition data is retrieved using the BOPTEST forecaster module. The forecast parameters, i.e. prediction horizon and interval, are set at the beginning of each test consistently with the control step and prediction horizon of the MPC. For this, the `/forecast_parameters` API call is used. Then, at every control step, the `/forecast` call returns the forecast data that is fed to the optimization. In this example, the perfect deterministic forecast provided by BOPTEST is assumed.

4.3. Results

Figure 8 shows the control performance results for the baseline controller and for variations of the MPC in each of the six BOPTEST case scenarios analyzed. The baseline controller is the embedded controller with PI logic as described in Section 4.1. Particularly, variations on the prediction horizon of 12, 24, or 48 hours, and on the MPC control step of 15, 30 or 60 minutes are studied. In Figure 8, the baseline performance is represented by a green dot, the different prediction horizons are distinguished using different colours, and variations of the control step are indicated with variations of marker shapes. From the six core KPIs that BOPTEST provides, we show thermal discomfort and operational cost, since these are the ones being minimized in the objective (Equation (12a)). However, all core KPIs for the peak heating period with dynamic price scenario and $T_s = 30$ minutes are shown in Figure 9, and a summary of all core KPIs obtained in this demonstration example is provided as supplemental data accompanying the online version of this article.

The results show that the baseline controller obtains the same total discomfort independently of the pricing scenario since its control logic ignores the pricing signal. The MPC generally outperforms the baseline controller with discomfort savings up to 90.8% and with cost savings up to 27.2% when using a prediction horizon of 48 hours and a control step of 15 minutes on the peak heat period with dynamic pricing. However, the baseline controller already delivers good performance, and designing the predictive controller to beat the baseline has proven to be a challenging task, especially in terms of operational cost. It is worth noting that the inclusion of the fan and pumping powers in the objective function is key to obtain operational cost savings. Neglecting these terms leads the MPC to work at low partial heat pump load, without considering the low but steady energy use of these auxiliary equipment. Another critical practice to achieve cost savings is to train the controller model not only to fit the zone operative temperature, but also the electrical and thermal powers of the heat pump. This way, the controller model learns an accurate representation of the heat pump COP behaviour, which is exploited during operation later on.

More dynamic pricing scenarios only result in small operational cost savings for the typical heating period scenarios. The main reason is that the relative pricing variations are hidden by the high constant pricing component for transportation fees and taxes included in all scenarios, which is relatively large in most of the European countries, including the one where the test case of this

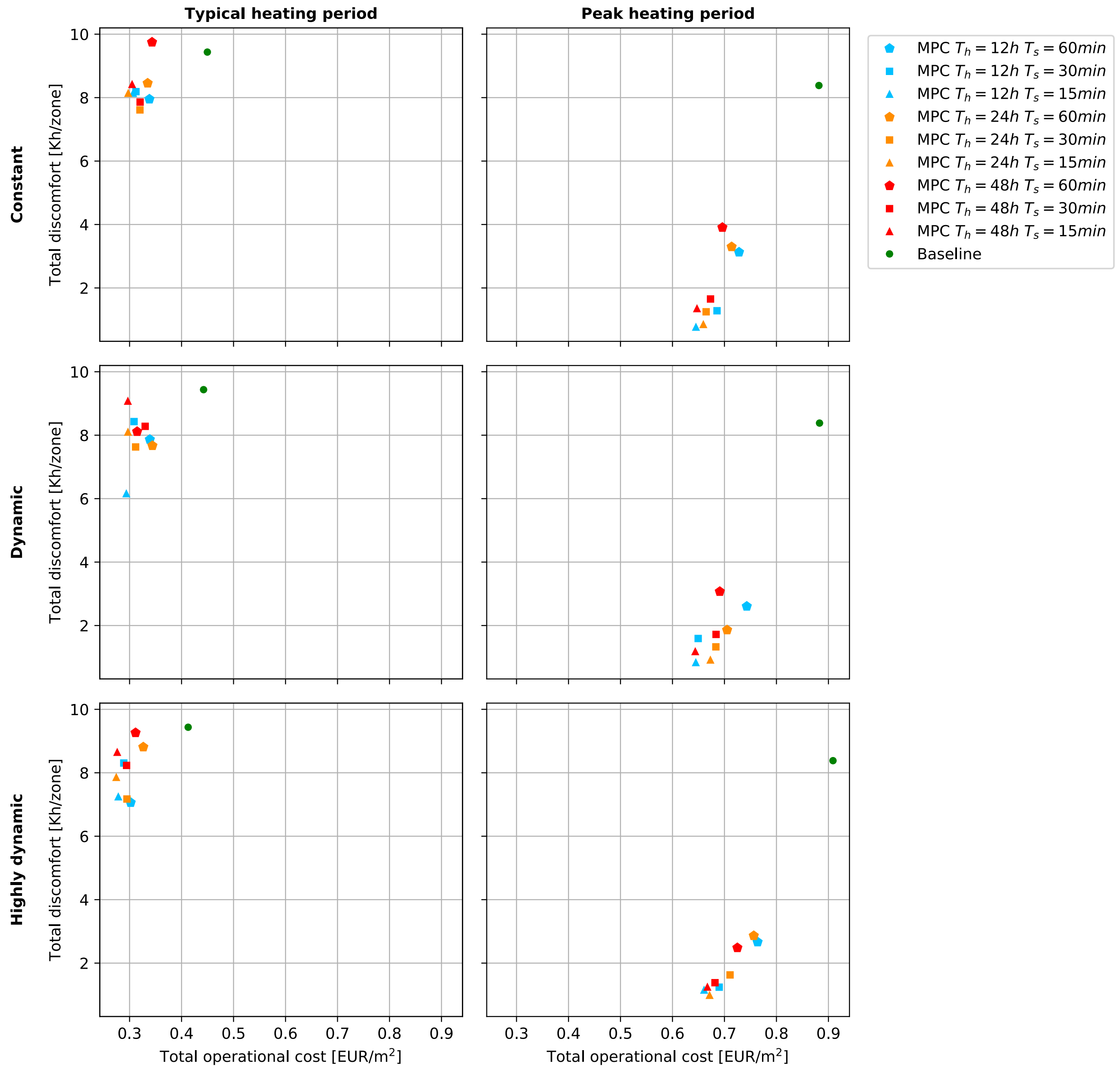


Figure 8. Control performance results for different test case runs in each of the BOPTEST scenarios. The rows show the results for the same pricing scenario, and the columns for the same period scenario. The results are obtained for variations on control step T_s and prediction horizon T_h of the same MPC.

example is located. This reduces the monetary incentive to shifting load. Notice that the peak heating scenarios are even less sensitive to pricing variations since they do not offer much freedom to operate without hampering comfort. The flexibility potential is further limited in this test case by higher electricity prices occurring when the ambient temperature is higher, and therefore when the heat pump could benefit from a larger COP. Shorter control steps usually lead to higher performance, especially for the peak heating period scenarios. However, longer prediction horizons do not seem to provide added value for this particular building type, as we observe similar results when using the same control step with different prediction horizons.

The improved performance of the predictive controllers can be clearly understood from Figure 10 which compares the simulation results for the peak heating period with dynamic pricing when using (1) the baseline controller and (2) the MPC with a control step of 30 minutes and a prediction horizon of one day. The first two graphs show the evolution of the zone operative temperature and the heat pump modulation signal respectively. The evolution of the total thermal discomfort and that of the operational cost are also shown and

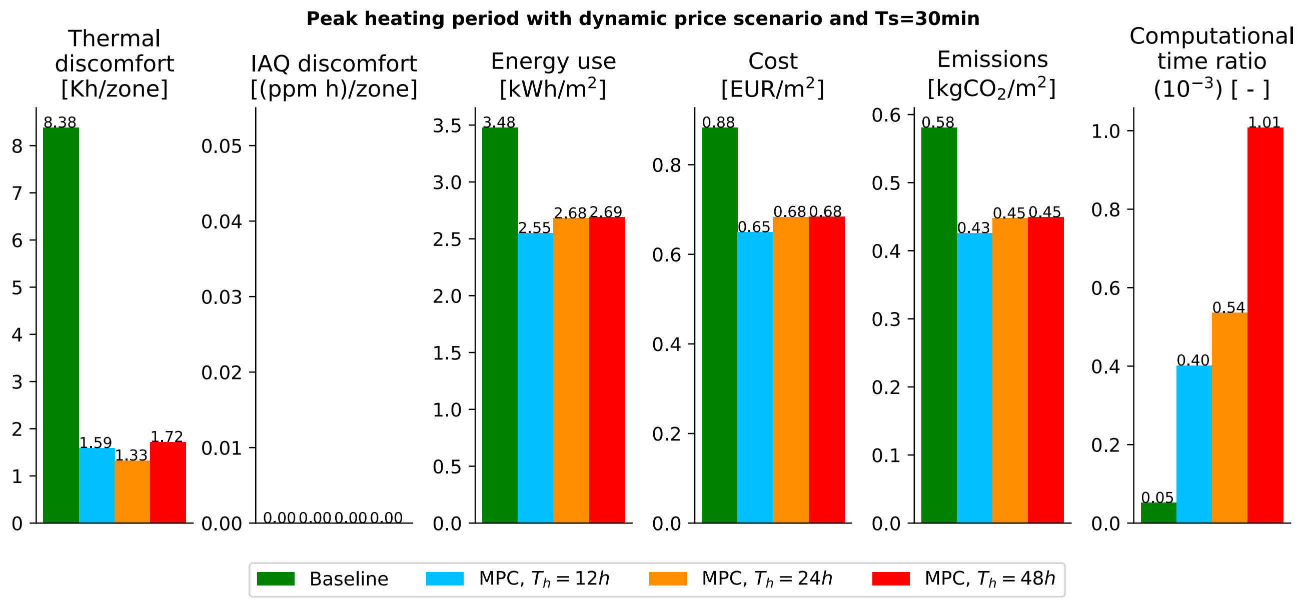


Figure 9. Summary of all KPIs for the peak heating period and dynamic price scenario with a fixed control step of $T_s = 30$ min and variations on the prediction horizon T_h of the same MPC.

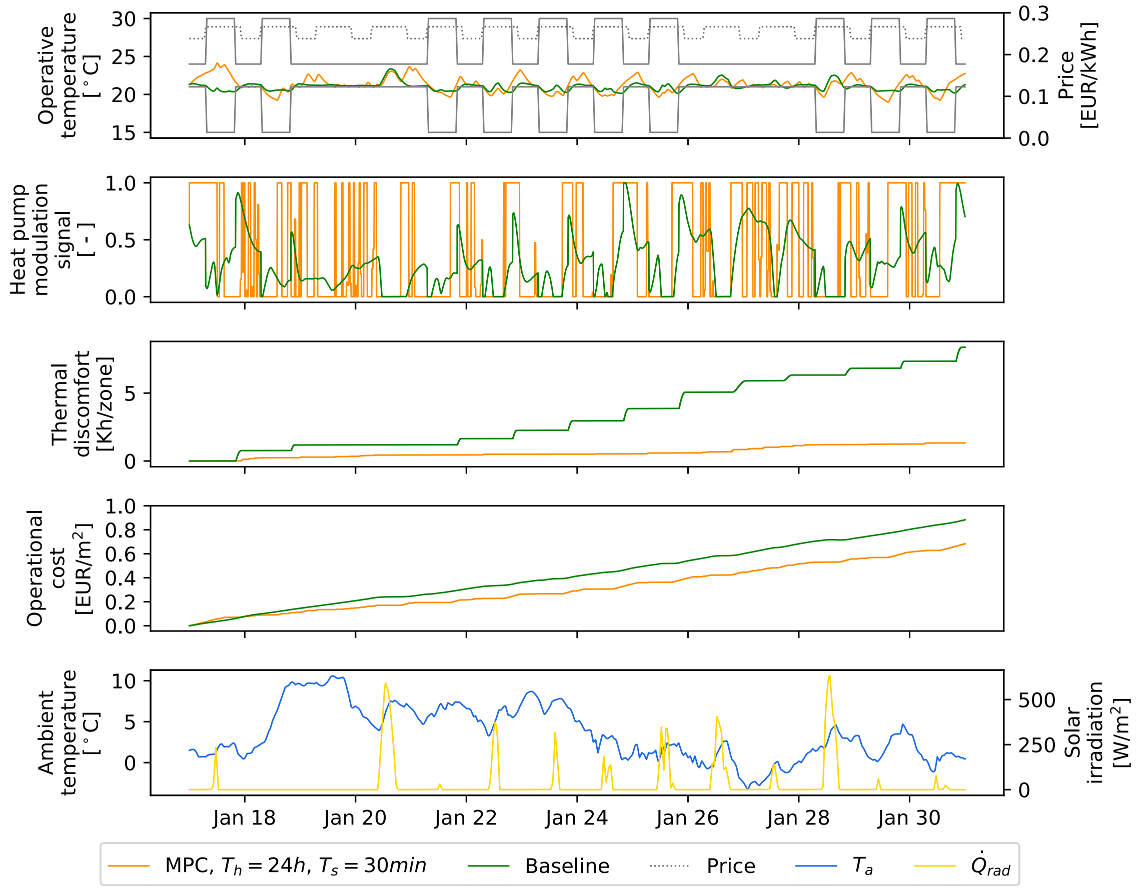


Figure 10. Simulation results for the peak heating period with dynamic pricing scenario. Results are shown for the baseline controller, and the MPC with prediction horizon of $T_h = 24$ h and control step of $T_s = 30$ min, assuming perfect prediction of ambient temperature, solar irradiation, and occupancy loads.

have been obtained by using the /kpis API function and storing the returned KPI results for every control step. The weather conditions of this test, i.e. direct normal solar irradiation and ambient temperature, are presented in the last graph. Particularly, the MPC outperforms the baseline in terms of lower discomfort by anticipating the temperature set point change in the morning. In terms of cost, the MPC obtains savings by working at full load which leads to higher COP and reduces the amount of time that the circulation pump and heat pump evaporator fan need to operate. However, this is an idealization since minimal run-times have not been imposed and start-up losses were not modelled. The former would avoid heat pump cycling while the latter would penalize heat pump cycling.

5. Discussion

The demonstration illustrates how the implementation of BOPTEST is used to evaluate the performance of an MPC strategy which was developed independently of BOPTEST. We expect that other MPC formulations and implementations or other control algorithms can also be tested. Furthermore, the demonstration was performed on a high-fidelity model of a single-zone floor heating system with a modulating heat pump, indicating the implementation of BOPTEST is sufficient to utilize such physics-based detailed emulator models. Though successful with respect to the primary use-case of such controller benchmarking as demonstrated here, there are opportunities to further develop the framework.

One limitation of the current framework is the deterministic treatment of emulator and operational performance. Real-world sensing and measurement systems are characterized by noise or suffer from biases, and disturbance forecasts contain uncertainty. Such uncertainty influence controller response and often require correction techniques. Though not implemented currently, BOPTEST is well-suited to incorporate uncertainty to forecast data coming from the forecasting module, once such uncertainty is characterized. Such effort is currently ongoing within IBPSA Project 1. Sensor bias can be added to the underlying Modelica models in a straightforward manner for discrete-time controllers. However, preliminary efforts to add sensor noise in continuous time control loops resulted in significant increases in the simulation time due to the high resolution of sampling required during simulation. Methods to implement such noise without sacrificing computational performance need to be further investigated.

A second limitation is the existence of one year of weather data for each test case, namely the latest TMY data that is available for the given test case location. This limits the ability for training models or controllers that require multiple years equivalent of operational data. A potential solution is the inclusion of different versions of TMY data for a particular location, for instance TMY2, TMY3, and TMYX (Climate.onebuilding.org 2021), as well as data collected from real meteorological stations, such that different data sets can be designated for the testing period, while others can be designated for training periods. Similarly, year-long simulations can become computationally cumbersome, for training or as a future testing scenario. Selecting a finite number of 'representative days' may alleviate the computational complexity (Bhattacharya et al. 2020).

Another consideration to note is the reliance on users to be vigilant with their use of BOPTEST for benchmarking and evaluation of controller performance. There are a few aspects to this. First, simulation data from time periods designated as testing periods should not be used for the training or calibration of predictive controllers. The intent of such testing periods is for them to be "unseen" by the controller. Second, there are many variations of sensor, control, and forecast points available at any given building site. Though BOPTEST limits the points a controller has access to, there will be some that are readily available at real installations, while others may have higher cost or may be available only at buildings with advanced operational implementations. Though BOPTEST does not currently explicitly quantify and compare the data requirements for one controller against another, users, and future performance evaluators, should be aware of the data and cost requirements they implicitly place on their controllers by making use of a given data point.

Work is ongoing for maintenance, and feature enhancement of the BOPTEST framework. Initial test case implementation has focused on small, simple systems, including the single-zone radiant floor system described here, other single-zone systems with fan coil unit or radiator, and a multi-zone residential system with radiators. These test cases are available in the repository. However, new, more complex test cases are in development, including those introduced in Blum et al. (2019). In addition, and with respect to the requirements set out in Section 3.1, it is a challenge for emulator developers to identify and remove all potential model idealizations as well as to maintain the models as Modelica library versions evolve. Therefore, work will continue on emulator revision as-needed, with versioning to maintain benchmarking capability. Additional KPIs that are identified as important to be included as core KPIs, such as peak electricity demand and a metric to quantify actuator and equipment cycling, may be added in the future. Furthermore, a service-based software architecture based on Alfalfa (NREL 2021) is in development that would allow the BOPTEST framework to be hosted as a web-service, with controllers acting as clients, rather than implementation on a local computing resource.

Finally, while BOPTEST provides a means to objectify performance across controllers across a number of KPIs, evaluation of which controller is 'better' may still be subject to some subjectivity, especially when a controller performs better in one KPI and worse in another. For example, whether a given reduction in operating cost is worth an increase in thermal discomfort in practice is ultimately up to the building owner, facility operator, and occupants. Similarly, reducing CO_2 emissions may come at the expense of increased operating cost if the utility tariff is not aligned with time-varying electric grid CO_2 emissions rates. The many practical situations confronted by building control and potential evaluation metrics have stopped BOPTEST short of providing a single evaluation metric. A centralized dashboard for collecting and viewing test results from BOPTEST is also under development to provide further information on performance and facilitate a user's ability to compare performance across different controllers and test cases.

6. Conclusion

While advanced control for building HVAC systems addresses a critical need for energy efficient grid responsive buildings (Roth and Reyna 2019), benchmarking the performance of control algorithms is challenging. In the literature, advanced controllers are still evaluated individually on case studies that differ in building or system type, evaluation metrics, and comparative baselines. Furthermore, development of test case studies requires significant effort and expertise, whether they utilize real buildings or building emulators. The BOPTEST framework and associated software tools have been developed to address these challenges. The framework incorporates a publicly available set of test cases, which include building emulators, a standardized Run-Time Environment (RTE), and common calculation of key performance indicators (KPIs) to enable reproducibility, testing, and evaluation. An example evaluation of an MPC strategy with BOPTEST was presented to demonstrate BOPTEST's ability to provide evaluation for advanced controllers.

One key development aim was to provide a framework that can be used with any potential test controller. To address this, the RTE is developed to contain all utilities for emulator simulation, data-exchange with a test controller, including providing forecasts and historical data point trajectories, and KPI calculation. Its implementation using Docker and exposure through a RESTful HTTP API allow for rapid deployment on any major computing platform and accessibility to most languages.

Another key development aim was the implementation of representative building emulators with high fidelity physical representation and enabling them to be controlled by arbitrary controllers, at either supervisory or local loop level. For this, the emulators are implemented with Modelica, which is supported by a multi-industry community of library and tool developers and provides the capability to model the necessary physical details. In addition, the emulators are exported as FMUs, with specially designed Modelica blocks that enable overwriting of embedded supervisory or local loop control signals, enabling their portability to the RTE.

BOPTEST defines a small, yet representative, set of core KPIs that are calculated within the RTE using data created during a test. BOPTEST also defines specific testing scenarios for each emulator, such as test time periods and electricity price profiles. These enable unambiguous quantification of controller performance. However, BOPTEST does not define a global ranking system, since the importance of certain KPIs relative to others can change based on the evaluation context. This setting relies on users to make the final interpretation of the performance data BOPTEST provides.

The BOPTEST framework has potential for future work, as was discussed in Section 5. For example, means to address forecast uncertainty will be incorporated. In addition, noise and bias will be added to measurement signals once methods for efficient simulation are developed. The usage of a single year of TMY weather data limits the ability for training models in controllers that require more than a year of data, though the inclusion of alternative versions of TMY data may alleviate this. Similarly, the computational burden of full year simulations may be alleviated through selection of representative days. Finally, development of a service-based architecture for cloud-hosting of BOPTEST, as well as development of an online dashboard for results sharing and viewing, will expand its usability.

Acknowledgements

This work emerged from the IBPSA Project 1, an international project conducted under the umbrella of the International Building Performance Simulation Association (IBPSA). Project 1 will develop and demonstrate a BIM/GIS and Modelica Framework for building and community energy system design and operation.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231, Contract No. DE-AC05-76RL01830, and Contract No. DE-AC36-08GO28308. The work of Javier Arroyo is financed by (Vlaamse Instelling voor Technologisch Onderzoek) VITO through a PhD Fellowship under Grant No. 1710754. The work of Filip Jorissen is financed by the University of Leuven (KU Leuven) through C3 project C32/18/063. The work of Harald Taxt Walnum is financed by the Research Council of Norway through the Research Centre on Zero Emission Neighbourhoods in Smart Cities (FME ZEN) under Grant No. 257660.

ORCID

David Blum http://orcid.org/0000-0003-3231-7937
Javier Arroyo http://orcid.org/0000-0003-2778-2973
Ján Drgoňa http://orcid.org/0000-0003-1223-208X
Filip Jorissen http://orcid.org/0000-0002-9273-0179
Harald Taxt Walnum http://orcid.org/0000-0002-8718-6797
Lieve Helsen http://orcid.org/0000-0002-9643-8204

References

2015 RECS square footage methodology. 2021. https://www.eia.gov/consumption/residential/reports/2015/squarefootage/.

Afram, A., and F. Janabi-Sharifi. 2014. “Theory and Applications of Hvac Control Systems–A Review of Model Predictive Control (MPC).” *Building and Environment* 72: 343–355. https://www.sciencedirect.com/science/article/pii/S0360132313003363.

Åkesson, J. 2008, March. “Optimica-an Extension of Modelica Supporting Dynamic Optimization.” B. Bachmann, ed. [Online]. https://www.researchgate.net/publication/253089188.

Åkesson, J., K. E. Årzén, M. Güfvert, T. Bergdahl, and H. Tummescheit. 2010, November. “Modeling and Optimization with Optimica and JModelica.org-languages and Tools for Solving Large-scale Dynamic Optimization Problems.” *Computers and Chemical Engineering* 34: 1737–1749.

Andersson, C., J. Åkesson, and C. Führer. 2016. *PyFMI: A Python Package for Simulation of Coupled Dynamic Models with the Functional Mock-up Interface*, Tech. Rep. LUTFNA-5008-2016, Centre for Mathematical Sciences, Lund University.

Arroyo, J., C. Manna, F. Spiessens, and L. Helsen. 2021. “An OpenAI-Gym Environment for the Building Optimization Testing (BOPTEST) Framework.” In *Proceedings of the 17th IBPSA Conference*, Bruges, Belgium.

Arroyo, J., F. Spiessens, and L. Helsen. 2020. “Identification of Multi-zone Grey-box Building Models for Use in Model Predictive Control.” *Journal of Building Performance Simulation* 13 (4): 472–486.

ASHRAE. 2018. “Guideline 36-2018: High Performance Sequences of Operation for HVAC Systems”.

Axelsson, M., F. Magnusson, and T. Henningsson. 2015. “A Framework for Nonlinear Model Predictive Control in JModelica.org.” In *Proceedings of the 11th International Modelica Conference*, Versailles, France, Vol. 118, 301–310.

Bhattacharya, S., Y. Chen, S. Huang, and D. Vrabie. 2020. “A Learning-based Time-efficient Framework for Building Energy Performance Evaluation.” *Energy and Buildings* 228: 110411.

Björsell, N., A. Bring, L. Eriksson, P. Grozman, M. Lindgren, P. Sahlin, A. Shapovalov, and M. Vuolle. 1999. “Ida Indoor Climate and Energy.” In *Proc. of the 6-th IBPSA Conference*, 1035–1042.

Blochwitz, T., M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, and A. Junghanns, et al. 2011. “The Functional Mockup Interface for Tool Independent Exchange of Simulation Models.” In *Proceedings of the 8th International Modelica Conference*.

Blum, D. 2013. “Analysis and Characterization of Ancillary Service Demand Response Strategies for Variable Air Volume HVAC Systems.” Master’s thesis, Department of Architecture, Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/82162.

Blum, D., F. Jorissen, S. Huang, Y. Chen, J. Arroyo, K. Benne, and Y. Li, et al. 2019. “Prototyping the BOPTEST Framework for Simulation-Based Testing of Advanced Control Strategies in Buildings.” In *Proceedings of the 16th IBPSA Conference*, Rome, Italy, 2737–2744.

Bünning, F., C. Pfister, A. Aboudonia, P. Heer, and J. Lygeros. 2021. “Comparing Machine Learning Based Methods to Standard Regression Methods for MPC on a Virtual Testbed.” In *Proceedings of the 17th IBPSA Conference*, Bruges, Belgium.

Bushby, S. T., M. A. Galler, N. S. Milesi-Ferretti, and C. D. Park. 2010. “The Virtual Cybernetic Building Testbed – a Building Emulator.” *ASHRAE Transactions* 116 (1): 37–44.

CBECS terminology. 2021. https://www.eia.gov/consumption/commercial/terminology.php.

Cimmino, M., and M. Wetter. 2017. “Modelling of Heat Pumps With Calibrated Parameters Based on Manufacturer Data.” In *The 12th International Modelica Conference*, 219–226, Prague, Czech Republic.

Climate.onebuilding.org. 2021. http://climate.onebuilding.org.

Crawley, D. B., L. K. Lawrie, F. C. Winkelmann, W. Buhl, Y. Huang, C. O. Pedersen, and R. K. Strand, et al. 2001. “Energyplus: Creating a New-generation Building Energy Simulation Program.” *Energy and Buildings* 33 (4): 319–331. https://www.sciencedirect.com/science/article/pii/S0378778800001146.

De Coninck, R., F. Magnusson, J. Åkesson, and L. Helsen. 2016. “Toolbox for Development and Validation of Grey-box Building Models for Forecasting and Control.” *Journal of Building Performance Simulation* 9 (3): 288–303. https://doi.org/10.1080/19401493.2015.1046933.

Docker. 2021. https://www.docker.com.

Drgoňa, J., J. Arroyo, I. C. Figueroa, D. Blum, K. Arendt, D. Kim, and E. P. Ollé, et al. 2020. “All You Need to Know About Model Predictive Control for Buildings.” *Annual Reviews in Control* 50: 190–232.

European Commission, Directorate-General for Energy. 2020. “Energy Prices and Costs in Europe.” Tech. Rep. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52016DC0769.

Fernandez, N. E., S. Katipamula, W. Wang, Y. Xie, M. Zhao, and C. D. Corbin. 2017. *Impacts of Commercial Building Controls on Energy Savings and Peak Load Reduction*. Pacific Northwest National Lab.(PNNL), Richland, WA (United States), Tech. Rep. https://www.pnnl.gov/main/publications/external/technical_reports/PNNL-25985.pdf.

Haves, P., and L. K. Norford. 1997. *Ashrae 825-rp Final Report: A Standard Simulation Testbed for the Evaluation of Control Algorithms and Strategies*. ASHRAE

Huang, S., Y. Chen, P. W. Ehrlich, and D. L. Vrabie. 2018. "A Control-Oriented Building Envelope and HVAC System Simulation Model for a Typical Large Office Building." In *Proceedings of Building Performance Modeling Conference and SimBuild*.

Huang, S., W. Wang, M. R. Brambley, S. Goyal, and W. Zuo. 2018. "An Agent-based Hardware-in-the-loop Simulation Framework for Building Controls." *Energy and Buildings* 181: 26–37. http://www.sciencedirect.com/science/article/pii/S037877881831764X.

Jorissen, F., G. Reynders, R. Baetens, D. Picard, D. Saelens, and L. Helsen. 2018a. "Implementation and Verification of the Ideas Building Energy Simulation Library." *Journal of Building Performance Simulation* 11 (6): 669–688.

Jorissen, F., G. Reynders, R. Baetens, D. Picard, D. Saelens, and L. Helsen. 2018b. "Implementation and Verification of the IDEAS Building Energy Simulation Library." *Journal of Building Performance Simulation* 11: 669–688.

Judkoff, R., and J. Neymark. 1995. *International Energy Agency Building Energy Simulation Test (BESTEST) and Diagnostic Method*, National Renewable Energy Laboratory, Tech. Rep. https://www.osti.gov/biblio/90674.

Kim, Y.-J., D. H. Blum, N. Xu, L. Su, and L. K. Norford. 2016. "Technologies and Magnitude of Ancillary Services Provided by Commercial Buildings." *Proceedings of the IEEE* 104 (4): 758–779.

Klein, S. 2017. "Trnsys 18: A Transient System Simulation Program." http://sel.me.wisc.edu/trnsys.

Mansson, L.-G., and D. McIntyre. 1997. "Technical Synthesis Report: A Summary of IEA Annexes 16 and 17 Building Energy Management Systems." https://www.iea-ebc.org/Data/publications/EBC_Annex_16-17_tsr.pdf.

Marzullo, T., S. Dey, N. L. Long, J. A. Leiva Vilaplana, K. Benne, and G. P. Henze. 2021. "A High-Fidelity Building Performance Simulation Test Bed for the Development and Evaluation of Advanced Controllers." *Journal of Building Performance Simulation*. To Be Submitted.

Mattsson, S. E., and H. Elmqvist. 1997. "Modelica-an International Effort to Design the Next Generation Modeling Language." *IFAC Proceedings Volumes* 30 (4): 151–155.

Mueller, D., M. Lauster, A. Constantin, M. Fuchs, and P. Remmen. 2016. "Aixlib – An Open-Source Modelica Library Within the IEA-EBC Annex 60 Framework." In *Proceedings of the conference BauSIM*, Dresden, Germany.

Nghiem, T. X. 2010. "Mle+: A Matlab-Energyplus Co-Simulation Interface".

NREL. 2021. "Alfalfa." https://github.com/NREL/alfalfa.

Nytsch-Geusen, C., J. Huber, M. Ljubijankic, and J. Rdler. 2013. "Modelica Buildingsystems Eine Modellbibliothek Zur Simulation Komplexer Energietechnischer Gebudesysteme." *Bauphysik* 35 (1): 2129.

Pallonetto, F., E. Mangina, F. Milano, and D. P. Finn. 2019. "Simapi, a Smartgrid Co-simulation Software Platform for Benchmarking Building Control Algorithms." *SoftwareX* 9: 271–281. https://www.sciencedirect.com/science/article/pii/S2352711018300463.

Pang, X., M. A. Piette, and N. Zhou. 2017. "Characterizing Variations in Variable Air Volume System Controls." *Energy and Buildings* 135: 166–175. https://www.sciencedirect.com/science/article/pii/S0378778816315754.

Park, C., D. R. Clark, and G. E. Kelly. 1985. "An Overview of HVACSIM+: A Dynamic Building/HVAC/Control Simulation Program." In *Proceedings from 1st Annual Building Energy Simulation Conference*. http://www.ibpsa.org/proceedings/BS1985/BS85_175_185.pdf.

Prívara, S., J. Široký, L. Ferkl, and J. Cigler. 2011. "Model Predictive Control of a Building Heating System: The First Experience." *Energy and Buildings* 43 (2): 564–572. https://www.sciencedirect.com/science/article/pii/S0378778810003749.

Romaní, J., A. de Gracia, and L. F. Cabeza. 2016. "Simulation and Control of Thermally Activated Building Systems (TABS)." *Energy and Buildings* 127: 22–42. https://www.sciencedirect.com/science/article/pii/S0378778816304261.

Roth, A., and J. Reyna. 2019. "Grid-Interactive Efficient Buildings Technical Report Series: Whole-Building Controls, Sensors, Modeling, and Analytics." https://www.osti.gov/biblio/1580329.

Sahlin, P., and E. F. Sowell. 1989. "A Neutral Format for Building Simulation Models." In *Proceedings of the Second International IBPSA Conference*, Vancouver, Canada, 147–154.

Scharnhorst, P., B. Schubnel, C. Fernández Bandera, J. Salom, P. Taddeo, M. Boegli, T. Gorecki, Y. Stauffer, A. Peppas, and C. Politi. 2021. "Energym: A Building Model Library for Controller Benchmarking." *Applied Sciences* 11 (8): 3518. https://www.mdpi.com/2076-3417/11/8/3518.

Sourbron, M., C. Verhelst, and L. Helsen. 2013. "Building Models for Model Predictive Control of Office Buildings with Concrete Core Activation." *Journal of Building Performance Simulation* 6 (3): 175–198.

Sowell, E. F., W. F. Buhl, A. E. Erdem, and F. C. Winkelmann. 1986. "A Prototype Object-Based System for HVAC Simulation." In *Proceedings of the Second International Conference on System Simulation in Buildings, Liege*.

Sun, F., G. Li, and J. Wang. 2009. "Unscented Kalman Filter Using Augmented State in the Presence of Additive Noise." 379–382. IEEE.

Togashi, E., and M. Miyata. 2019. "Development of Building Thermal Environment Emulator to Evaluate the Performance of the HVAC System Operation." *Journal of Building Performance Simulation* 12 (5): 663–684.

Togashi, E., M. Miyata, and Y. Yamamoto. 2020. "The First World Championship in Cybernetic Building Optimization." *Journal of Building Performance Simulation* 13 (3): 391–408.

Vázquez-Canteli, J. R. 2020. "The City Learn Challenge." https://www.citylearn.net.

Vázquez-Canteli, J. R., J. Kämpf, G. Henze, and Z. Nagy. 2019. "Citylearn v1.0: An Openai Gym Environment for Demand Response With Deep Reinforcement Learning." In *Proceedings of the 6th ACM International Conference on. Systems for Energy-Efficient Buildings, Cities, and Transportation, ser. BuildSys '19*, 356–357. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3360322.3360998.

Vázquez-Canteli, J. R., and Z. Nagy. 2019. "Reinforcement Learning for Demand Response: A Review of Algorithms and Modeling Techniques." *Applied Energy* 235: 1072–1089.

Walnum, H. T., I. Sartori, and M. Bagle. 2020. "Model Predictive Control of District Heating Substations for Flexible Heating of Buildings." In *SINTEF Proceedings no 5*, ser.

BuildSim-Nordic 2020. Oslo, Norway, Oct. 13–14: International Conference Organised by IBPSA-Nordic, 123–130. https://sintef.brage.unit.no/sintef-xmlui/handle/11250/2683181.

Wan, E. A., and R. V. D. Merwe. 2000. "The Unscented Kalman Filter for Nonlinear Estimation," In *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, AS-SPCC 2000*, 153–158.

Wang, S. 1999. "Dynamic Simulation of Building VAV Air-conditioning System and Evaluation of EMCS on-line Control Strategies." *Building and Environment* 34 (6): 681–705.

Wang, Z., and T. Hong. 2020. "Reinforcement Learning for Building Controls: The Opportunities and Challenges." *Applied Energy* 269: 115036. https://www.sciencedirect.com/science/article/pii/S0306261920305481.

Warren, P. 2002. "Bringing Simulation to Application." https://www.iea-ebc.org/Data/publications/EBC_Annex_30_tsr_final.pdf.

Wetter, M. 2011. "Co-simulation of Building Energy and Control Systems with the Building Controls Virtual Test Bed." *Journal of Building Performance Simulation* 4 (3): 185–203.

Wetter, M., K. Benne, A. Gautier, T. S. Nouidui, A. Ramle, A. Roth, H. Tummescheit, S. Mentzer, and C. Winther. 2020. "Lifting the Garage Door on Spawn, an Open-Source BEM-Controls Engine." In *Proceedings of the 2020 Building Performance Modeling Conference and SimBuild co-organized by ASHRAE and IBPSA-USA*.

Wetter, M., M. Bonvini, and T. S. Nouidui. 2016. "Equation-based Languages – a New Paradigm for Building Energy Modeling, Simulation and Optimization." *Energy and Buildings* 117: 290–300. https://www.sciencedirect.com/science/article/pii/S0378778815303315.

Wetter, M., M. Fuchs, P. Grozman, L. Helsen, F. Jorissen, D. Müller, C. Nytsch-Geusen, D. Picard, P. Sahlin, and M. Thorade. 2015. "IEA EBC Annex 60 Modelica Library-an International Collaboration to Develop a Free Open-Source Model Library for Buildings and Community Energy systems." In *Proceedings of building simulation 2015*, 395–402.

Wetter, M., C. van Treeck, L. Helsen, A. Maccarini, D. Saelens, D. Robinson, and G. Schweiger. 2019, September. "IBPSA Project 1: BIM/GIS and Modelica Framework for Building and Community Energy System Design and Operation – Ongoing Developments, Lessons Learned and Challenges." *Proc. of the Sustainable Built Environment Conference* 323: 012114. https://doi.org/10.1088/1755-1315/323/1/012114.

Wetter, M., W. Zuo, T. S. Nouidui, and X. Pang. 2014. "Modelica Buildings Library." *Journal of Building Performance Simulation* 7 (4): 253–270.

Wölfle, D., A. Vishwanath, and H. Schmeck. 2020. "A Guide for the Design of Benchmark Environments for Building Energy Optimization." In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, ser. BuildSys '20*. 220–229. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3408308.3427614.

Xu, P., P. Haves, and J. Deringer. 2004. "A Simulation-Based Testing and Training Environment for Building Controls." In *Proceedings of SimBuild 2004*.

Yang, T., K. Filonenko, K. Arendt, and C. Veje. 2020. "Implementation and Performance Analysis of a Multi-Energy Building Emulator." In *2020 6th IEEE International Energy Conference (ENERGYCon)*, 451–456.

Zhan, S., and A. Chong. 2021. "Data Requirements and Performance Evaluation of Model Predictive Control in Buildings: A Modeling Perspective." *Renewable and Sustainable Energy Reviews* 142: 110835. doi:10.1016/j.rser.2021.110835.