MShoot: an Open Source Framework for Multiple Shooting MPC in Buildings

Arendt, Krzysztof; Veje, Christian

Published in:
Proceedings of the 16th IBPSA International Conference and Exhibition Building Simulation 2019

Publication date:
2019

Document version
Accepted manuscript

Document license
Unspecified

Citation for published version (APA):

Terms of use
This work is brought to you by the University of Southern Denmark through the SDU Research Portal. Unless otherwise specified it has been shared according to the terms for self-archiving.
If no other license is stated, these terms apply:
• You may download this work for personal use only.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying this open access version
If you believe that this document breaches copyright please contact us providing details and we will investigate your claim.
Please direct all enquiries to puresupport@bib.sdu.dk
MShoot: an Open Source Framework for Multiple Shooting MPC in Buildings

Krzysztof Arendt, Christian T. Veje
1Center for Energy Informatics, University of Southern Denmark, Odense, Denmark

Abstract

The paper presents a new open source MPC framework for building operation based on the multiple shooting method, called MShoot. The framework is suitable for both physical and data-driven models, and allows to connect models through one of the three available interfaces: FMI, scikit interface, and generic Python interface. The framework was tested on two different MPC setups, based on a physical and a data-driven control model, respectively. The results indicate that the framework is currently suitable for small scale optimization problems. A support for analytical derivatives, planned in the next version of the software, is a prerequisite for large scale problems.

Introduction

The research on Model Predictive Control (MPC) in buildings has gained a significant momentum in the recent years. Many studies showed that MPC has a potential of significantly reducing the building energy consumption and increasing the energy flexibility, compared to traditional PID- and schedule-based control strategies (Blum et al. (2019)). However, due to incomparably higher complexity of MPC systems, the process of large-scale implementation of MPC in real buildings is slow. Some of the major obstacles are e.g. model development and calibration, disturbance forecasting (weather, occupancy), scalability and robustness of dynamic optimization solvers. Therefore, the final MPC performance heavily depends on a multitude of factors (Blum et al. (2019)). As a result, most of the existing MPC frameworks are tied to specific modeling approaches or optimization solvers, and form complex toolchains.

In example, two recently developed MPC frameworks, MPCpy (Blum and Wetter (2017)) and TACO (Jориссен et al. (2019)), depend on Optimica (Акессон (2008)), an extension to Modelica available in the JModelica.org platform. They require models to be developed in Modelica, however with some restrictions, e.g. external C functions are not allowed. The Optimica framework is based on the collocation method. It also takes the advantage of analytic gradients provided by Modelica, significantly speeding up the computational time. That makes Optimica-based frameworks very scalable.

Gräber et al. (2012) developed an MPC framework suitable for models compliant with Functional Mock-up Interface (FMI) (Blochwitz et al. (2011)), enabling to use also non-Modelica models. The framework is based on a multiple shooting method implemented in MUSCOD-II library (improved version of MUSCOD developed by Bock and Plitt (1984)). It was successfully being used in various industrial applications (e.g. Fischer et al. (2017)), however, it is a proprietary software.

Rantil et al. (2009) developed a multiple shooting algorithm for optimal control based on the JModelica.org platform. The algorithm can be used with Modelica models. However, it is not included in the current JModelica.org version. The authors also did not mention if the code was released to public domain.

Since it is not clear yet what is the best modeling approach for MPC in buildings, it might be beneficial to have an MPC framework suitable for both physical and data-driven (machine learning) models. E.g. Arendt et al. (2018) showed that data-driven models outperform white- and gray-box thermal zone models in terms of accuracy. Data-driven models are also more scalable than purely-physics-based models. Consequently, machine learning is gaining increasing attention throughout the recent years and is being incorporated into custom MPC systems, e.g. using regression trees and convex optimization (Smarra et al. (2018)), neural networks and branch and bound method (Ferreira et al. (2012)), reinforcement learning (Yang et al. (2015)), or approximate MPC (Drgoňa et al. (2018)).

Aim of the paper

This paper presents a new open source framework for MPC based on multiple shooting method, suitable for both physical and data-driven models. The core functionality of the framework is presented on an exemplary case study in which a Modelica model and a Support Vector Regression (SVR) model are used to optimize the heating/cooling in a thermal zone.
Software description

MShoot is an open source Python library for defining and solving MPC problems, based on the multiple shooting method. It is easily installable using the default Python package manager pip. The project is hosted on GitHub: https://github.com/sdu-cfei/mshoot.

The software has a modular structure, enabling to easily connect different model types or NLP solvers (Fig. 1). Currently, MShoot supports only virtual experiments in which a real building is replaced with an emulation model. In a virtual setup, two kind of simulation models are required: the control model and the emulation model. The control model is used to find the optimal control strategy. It is possible to use different types of models for control and emulation. In example, the control model can be a machine learning model whereas the emulation model can be a physical model. An sMAP-based (Dawson-Haggerty et al. (2010)) interface for communication with real buildings is under development.

There are currently two specialized simulation interfaces, (1) FMI and (2) the interface to machine learning models implemented in scikit-learn (Pedregosa et al. (2011)). The latter interface is referred to as scikit interface throughout the paper. It is also possible to use any model through a generic interface defined as a Python abstract base class with only one method, simulate(udf, x0), where udf is the input data frame, x0 is the initial state. The user-defined method should return two data frames, one with the resulting outputs and the other with the states.

The FMI interface is based on the PyFMI library (Andersson et al. (2016)) and it simply sets the initial state of the model, passes the inputs and reads the simulation outputs. The scikit interface, on the other hand, is more than just a simple input/output translation layer. It contains also an algorithm for instantiating and training different types of machine learning models. The models currently available are as follows: linear regression, Ridge regression, near-est neighbors, random forest, feed-forward neural network, and support vector regression (SVR). To preserve the dynamic characteristics of a physical system being modeled, all data-driven models are trained to predict the time derivative of the states rather than the absolute state values. The trained derivatives are subsequently utilized throughout a dynamic simulation to find the final state trajectories. Finally, the user can group model outputs into states and outputs. Technically, there is no difference between both, however this way it is compatible with the FMI interface. In MShoot, the states can be used in the constraint definition, whereas the outputs can be used in the cost function.

Currently, only one NLP solver is available: SLSQP interfaced through SciPy and based on the algorithm developed by Kraft (1988). The solver is based on the Sequential Quadratic Programming algorithm. However, any solver with a similar interface can be easily connected.

It should be noted, that not all of the data-driven models available in the scikit interface might be suitable for the currently implemented NLP solver. Some models might work better with interior-point methods or even global optimization methods, like constrained evolutionary algorithms (at least in terms of the solution quality, not the computational speed). Only one data-driven model example is presented in this paper. The relationship between various model types and optimization solvers will be investigated more in the future.

Model derivatives used in the optimization are approximated numerically. The support for analytical derivatives in models where they are available (e.g. some Modelica-based FMUs) is planned to be added in the future. It is expected that it will significantly improve the computational speed as well as reduce the reliance on the numerical integration method accuracy (low error tolerance is crucial for accurate approximation of gradients).

The control model is used in the MPC loop (Fig. 1) to find the optimal control strategy for each optimization horizon. The optimization is performed using the multiple shooting method, in which the optimization horizon is divided into N subintervals (Fig. 2). The state trajectories within each subinterval are obtained through simulation. The states are free to vary within the feasible region (hard constraints), however if the initial state violates the constraints, the constraints are temporarily relaxed in the affected subintervals so that the solver can proceed with the calculations. The NLP solver solves the augmented constrained minimization problem in the following form:
\[
\begin{align*}
\min_{u_1, u_2, \ldots, u_N} & \sum_{k=1}^{N} f(\hat{x}_k, \hat{y}_k) \\
\text{s.t.} & \quad x_{k,\text{max}} \geq x_k \geq x_{k,\text{min}}, \quad k = 1, 2, \ldots, N \\
& \quad x_{k+1}^L - x_k^R = 0, \quad k = 1, 2, \ldots, N - 1
\end{align*}
\]

where the subscript \( k \) refers to a specific subinterval, \( u \) is the input vector to be optimized, \( f \) is the cost function, \( \hat{x} \) is the normalized state vector, \( \hat{y} \) is the normalized output vector, \( \hat{x}_k^L \) and \( \hat{x}_k^R \) are the states at the beginning and the end of subinterval \( k \), respectively, and \( x_{k,\text{min}} \) and \( x_{k,\text{max}} \) are the state constraints in the subinterval \( k \). The function \( f(\hat{x}, \hat{y}) \) in Eq. (1a) as well as the constraints in Eq. (1b) are defined by the user. The continuity constraints, Eq. (1c), are added automatically.

The aim of the normalization of the states and outputs is to improve the convergence of the NLP solver. Many solvers, including the one currently implemented in \( M\text{Shoot} \), prefer data normalized to the same order of magnitude, typically around 1. In \( M\text{Shoot} \), the user is asked to provide nominal values for the outputs, whereas the states are normalized based on the given constraints.

The division of the optimization horizon into subintervals and adding the continuity constraints improves the convergence and numerical stability in nonlinear problems by preventing the growth of the error due to poor initial guess (Bock and Plitt (1984)). In addition, this approach enables to use parallel processing, as each subinterval has to be simulated separately. In the current implementation, however, the parallel processing is disabled, as it did not improve the overall computational time due to the global interpreter lock (GIL) in Python (in pure Python models) and due to long Functional Mock-up Unit (FMU) instantiation time compared to simulation time (in FMU models). Therefore, more work on the parallelization is needed in the future.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2.png}
\caption{Division of the state \( x \) trajectory into \( N \) subintervals with the feasible region marked in gray.}
\end{figure}

### Examples

The presented examples are aimed to highlight the main functionality of the developed software. The main aspects discussed in the paper are: verification of the optimization solution, analysis of the influence of the integration error tolerance on the optimization solution, scalability, and a demonstration of an MPC setup based on a data-driven model.

The following subsections contain a description of the experimental setup, including the model development and calibration, and the results related to the listed aspects.

#### Experimental setup

Two MPC setups were developed, each aimed at a different aspect of the framework (Table 1). Both setups were used to optimize heating/cooling rate \( q \) [W] in the room while maintaining indoor temperature within comfortable constraints.

#### Table 1: Case study overview (filled circles mark tested aspects).

<table>
<thead>
<tr>
<th>Components</th>
<th>Setup 1</th>
<th>Setup 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control model (interface)</td>
<td>R1C1 (FMI)</td>
<td>SVR (scikit)</td>
</tr>
<tr>
<td>Emulation model (interface)</td>
<td>R1C1 (FMI)</td>
<td>R1C1 (FMI)</td>
</tr>
<tr>
<td>Tested aspect</td>
<td>Setup 1</td>
<td>Setup 2</td>
</tr>
<tr>
<td>Solution verification</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Integration error tolerance</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Scalability</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Data-driven approach</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

The first model used in the example is an R1C1 thermal network model (Fig. 3) developed in Modelica/Dymola. In Setup 1 it is used as both the emulation and the control model. In Setup 2 it is used as the emulation model, whereas the control model is based on a Support Vector Regression (SVR) model. The models represent the gray-box and the black-box modeling approach, respectively. Both approaches are generic and scalable, and rely on measured data for training. However, unlike the black-box model, the gray-box model is partially physics-based leading to less data needed for training.

Both models are connected to the MPC loop through different interfaces: FMI (R1C1) and the scikit interface (SVR) (Fig. 1).

The R1C1 model (in both setups) is calibrated based on the measured data from one of the classrooms in a 8500 m² teaching university building OU44 at the SDU Campus Odense (Jradj et al. (2017)). The measured data includes indoor temperature, radiator valve position, ventilation damper position, number of occupants, outdoor temperature, and solar radiation. The classroom has a floor area of 139 m² and is connected to a balanced ventilation system. By design, the maximum ventilation air supply to the room
is 4800 m$^3$/h. The calibration of the model was performed using ModestPy (Arendt et al. (2018)), which is a Python library for facilitating parameter estimation in Functional Mock-up Units (FMUs). The parameters were estimated based on measured data from the period of 4-9 April 2018 (120 h). The model was additionally validated on the period of 9-13 April 2018 (96 h). The comparison between the simulation results and the measurements from the training and validation data sets is presented in Fig. 4.

The preliminary tests (not reported in this paper) showed no increase in the accuracy for higher-order models. Possibly, the major source of error is the assumption that the windows are always closed (due to unavailable sensor data), whereas in reality occupants open the windows frequently during that period of the year. Finally, since the model reasonably represents the dynamics of the thermal zone and the focus of the paper is not on the model accuracy, the R1C1 model was selected for further tests.

The R1C1 model inputs in the calibration phase were: solar radiation $q_{\text{sol}}$ [W/m$^2$], outdoor temperature $T_{\text{out}}$ [°C], number of occupants $n_{\text{occ}}$ [-], damper position $d_{\text{pos}}$ [%], and radiator valve position $v_{\text{pos}}$ [%]. The R1C1 model used in the MPC simulation does not use the valve and damper positions as inputs. Instead, the heating/cooling rate $q$ [W] (MPC control variable) is given directly. It is assumed that $q$ can vary from -2000 to 2000 W. In a real setup the indoor temperature calculated by the model could then be used as a setpoint for the HVAC system (supervisory MPC).

In Setup 2, the control model is based on a linear SVR, through the scikit interface (Fig. 1). Internally, the SVR model implementation in scikit-learn (Pedregosa et al. (2011)) is based on the LIBSVM library (Chang and Lin (2011)). The SVR model was trained based on the R1C1 model results and not on the actual measurements. The are two reasons for this approach. Firstly, the SVR model uses the heating/cooling rate $q$ as input, which was not measured in the test room, so the model could not be trained directly from the measured data. Secondly, it is easier to excite the R1C1 model and get a wide range dynamic response for training the data-driven model, than to train the data-driven model directly on the measured data, especially on such a short measurement period. A sinusoidal signal $q$ from -2000 W to 2000 W was used to excite the R1C1 model. The validation results are presented in Fig. 5. The authors find the adopted approach justified, since the focus of the paper is on the MPC framework itself. The SVR training and validation periods are exactly the same as the ones used in the calibration of the R1C1 model, i.e. 4-9 April and 9-13 April 2018.

In both setups, the MPC simulations were carried out using measured inputs from 5-8 April 2018 (72 h) (Fig. 6). Also, in both setups the same cost function
and constraints were used:

\[
\begin{align}
\min_{q_1, q_2, \cdots, q_N} & \sum_{k=1}^{N} \left( \frac{q_k}{q_{nom}} \right)^2 \\
\text{s.t.} \quad & T_{k,\text{max}} \geq T_k \geq T_{k,\text{min}}, \quad k = 1, 2, \ldots, N
\end{align}
\]

(2a)

(2b)

where \( q_k \) is the heating/cooling rate \([W]\) during subinterval \( k \), \( N \) is the total number of subintervals, \( q_{nom} \) is the nominal heating/cooling rate \([W]\), \( T_k \) is the indoor temperature \([\degree C]\) during the subinterval \( k \), \( T_{k,\text{max}} \) and \( T_{k,\text{min}} \) are the temperature constraints \([\degree C]\). In other words, the MPC aims to minimize energy consumption, while maintaining the indoor temperature within the constraints.

Since the multiple shooting method is used, Eq. (2) is automatically supplemented with additional state continuity constraints:

\[
T_{k+1}^L - T_k^R = 0, \quad k = 1, 2, \ldots, N - 1
\]

(3)

where \( T_k^L \) and \( T_k^R \) are the temperatures at the beginning and the end of subinterval \( k \), respectively.

Five optimization horizons are considered in total: 2h, 4h, 6h, 8h, and 10h. The optimized input trajectory of each horizon is sent to the emulation model, but only the inputs for the next 1h are used. After 1h of the simulation time, \( M\text{Shoot} \) optimizes next horizon, and the simulation progresses in these steps until reaching the final time.

Finally, to test the influence of the integration solver error tolerance in physics-based models on the MPC solution quality, the R1C1 model was exported to multiple FMUs with different solver tolerances set in Dymola, from 1e-4 to 1e-9. All the FMUs are based on FMI for co-simulation using the DASSL integration algorithm (Dymola solver). CVODE solvers, also available in Dymola, were not tested. It was found that the integration algorithm tolerance does not affect the parameter estimation, therefore only the influence on the MPC solution is reported in the following subsection.

Results

The MPC solution can be verified in Fig. 7, presenting the optimized heating/cooling signal \( q \) and the indoor temperature profiles \( T \) for different optimization horizons using an integration tolerance of 1e-11. As shown, extending the optimization horizon leads to earlier and more planned response of the MPC to the anticipated changes in temperature constraints (black dashed lines). In the case of 2h optimization horizon, MPC heats indoor air just before the anticipated change in the constraints, using more power than for longer horizons. In overall, the profile of \( q \) is piece-wise smooth, without any oscillations, suggesting that the solver converged to the optimal trajectory. The indoor temperature seriously violated the temperature constraints during the second and third day, but this is due to significant indoor heat gains (Fig. 6) and insufficient heating/cooling capacity of the system (2000/-2000 W). The low capacity was assumed intentionally, in order to show that despite using hard constraints, the framework has no stability issues when the feasible region is not reachable for the given system. The increased optimization horizon leads to smaller temperature constraint violations, especially in the third day, but at the cost of increased energy consumption. The total energy consumption for the 2h optimization horizon is 40.9 kWh, whereas it is 44.0 kWh for the 10h horizon.

In contrast, the heating/cooling profile obtained with the integration tolerance 1e-4 is highly unstable and oscillating (Fig. 8). Although the indoor temperature mostly stayed within the constraints, the system used almost 3 times as much energy than in the case of tolerance 1e-11 (Fig. 9). Therefore, the integration tolerance has a significant effect on the accuracy of the numerically approximated derivatives, which are used to optimize the control input trajectory.
The relationship between the temperature constraint violation and the total energy consumption for different integration tolerances is presented in Fig. 9. The constraint violation is given in Kh, i.e. the number of hours multiplied by temperature offset from the constraint. The FMUs with tolerances 1e-9 and 1e-11 provided similar solutions. The higher tolerances (1e-4 through 1e-7) resulted in significantly higher energy consumption, but the relationship is nonlinear and even nonmonotonic. E.g. the highest energy consumption was obtained with the tolerance 1e-6, but at the same time this case resulted a slightly lower constraint violation than other cases.

The results of the MPC setup based on the SVR control model (Setup 2) are presented in Fig. 10. The indoor temperature trajectory in this case resembles the one with the R1C1 control model. The major temperature constraint violations also occur during the second and third day of the simulation. The increase of the optimization horizon leads to an earlier adaptation of the heating/cooling trajectory, which is as expected, however the q profiles slightly differ from the ones yielded by the R1C1 control model (Fig. 7). The difference is likely due the model mismatch (Fig. 5). Compared to Setup 1 (no model mismatch), Setup 2 with the SVR control model resulted in around 10-20% higher overall energy consumption (depending on the optimization horizon).

The computational time and its dependence on the optimization horizon (and thus the number of optimization variables) is presented in Fig. 11. Firstly, it was found that the MPC simulation based on the SVR control model was approximately twice as fast as the one based on the R1C1 FMU. For the 10h optimization horizon the total computational times were 58-73 minutes for R1C1 (depending on the integration tolerance) and 29 minutes for SVR. However, since the total CPU time is affected by both the single-simulation time and the number of NLP solver iterations, this result is not easily generalizable. In example, the decrease of the integration tolerance in R1C1 FMU from 1e-9 to 1e-11 increased the single-simulation time by 10%, but reduced the overall MPC simulation time by 25% due to less NLP iterations (Fig. 11). On the other hand, all models show almost linear increase in the computational time with the increasing optimization horizon.

Since the models have one state (T) and one control input (q), and the control input was discretized with a 1h-resolution grid, the total number of optimization variables in the NLP problem is equal to the optimization horizon in hours times two (top axis in Fig. 11). Therefore, the presented case is a small scale MPC example. It is expected, that the utilization of the analytical gradients (optional feature in the FMI standard) significantly increase the computational performance of the framework and will allow to use it also in large scale problems.
Conclusion

The paper presented the new open source MPC framework based on the multiple shooting method. It is written in Python and supports both physical and data-driven simulation models, which can be connected through one of the three available interfaces. The presented results tested selected aspects of the framework. The main conclusions from the results are as follows:

- Due to the numerical approximation of model derivatives, the MPC solution is susceptible to integration errors. Thus, very low integration solver tolerance (1e-11) is advised.
- The current computational demand of the framework allows use for small scale systems. Adding support for analytical gradients (e.g. through FMI) will likely significantly speed up the computations.
- Unlike frameworks based on the collocation method, MShoot can also be used with data-driven models, but more work is required to understand which model types are suitable for specific NLP solvers.

Therefore, the main focus of the future development will be on: adding the support for analytical gradients in physical models, new NLP solvers, and an sMAP-based interface for communication with real buildings. In addition, further work on the parallelization of the algorithm is needed to address also the scalability issues of MPC setups based on data-driven models.

Acknowledgment

This work was in part supported by the Innovation Fund Denmark for the project COORDICY (4106-00003B) and by the IBPSA Project 1 CEI - SDU Deltagelse project funded by the Danish Energy Agency under the Energy Technology Development and Demonstration Program (EUDP Project no. 64018-0518).

References


