ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP033

APPLYING MODEL PREDICTIVE CONTROL TO REGULATE THERMAL STABILITY OF A HARD X-RAY MONOCHROMATOR USING THE KARABO SCADA FRAMEWORK

M. A. Smith*, G. Giovanetti, S. Hauf, I. Karpics, A. Parenti, A. Samadli,
L. Samoylova, A. Silenzi, F. Sohn, P. Zalden, European XFEL, Schenefeld, Germany

Abstract

Model Predictive Control (MPC) is an advanced method of process control whereby a model is developed for a real-life system and an optimal control solution is then calculated and applied to control the system. At each time step, the MPC controller uses the system model and system state to minimize a cost function for optimal control. The Karabo SCADA Framework is a distributed control system developed specifically for European XFEL facility, consisting of tens of thousands of hardware and software devices and over two million attributes to track system state.

This contribution describes the application of the Python MPC Toolbox within the Karabo SCADA Framework to solve a monochromator temperature control problem. Additionally, the experiences gained in this solution have led to a generic method to apply MPC to any group of Karabo SCADA devices.

MONOCHROMATORS AT XFEL

European XFEL [1] operates three beamlines capable of delivering hard and soft X-rays. For the hard X-ray beamlines, silicon monochromators are used to select the pass band of X-ray energies that continue through to the instrument. The monochromator works by using Bragg's law, which gives the relationship between the incident angle of X-rays on a crystal lattice and the reflected angle. The position of the crystals is adjustable via motors, in order to control the incident angle of the X-rays to the silicon crystal's lattice structure. Python MPC Toolbox [2] was used to solve a monochromator temperature control problem.

Thermal drift in a monochromator causes a drift of the transmitted photon energy. Silicon is commonly used for Hard X-ray Monochromators due to its good commercial availability and point of zero thermal expansion around 125 K [3]. To mitigate the impact of temperature jumps caused by the X-ray pulse trains at European XFEL, the monochromator temperature has to stay just below this temperature. A CAD drawing of the two crystals mounted inside a monochromator is shown in Fig. 1. The orange arrow in the drawing shows the path of the X-ray beam through the two crystals of the monochromator in a 2-bounce configuration. To cool the overall monochromator, the silicon crystals are attached to copper plates which are in turn connected to a single cryogenic cold head to cool the plates down to around 100 K. Each crystal also has a local heating element for fine control of each crystal's individual temperature, along with a temperature sensor for feedback. Because each crystal is connected to a common cold head, heat applied to one crystal will have a time-delayed effect on the temperature of the other crystal via conduction.

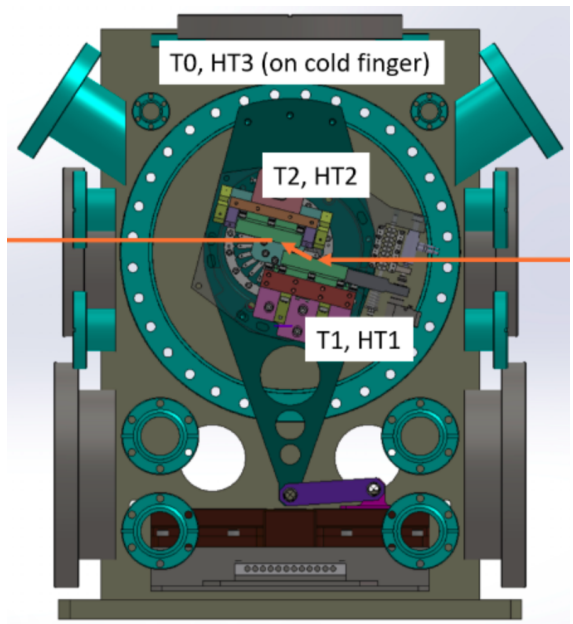

Figure 1: CAD drawing of the X-ray monochromator, showing the path of the X-ray beam through the two silicon crystals of the monochromator. Tx and HTx identify the temperature sensors and heater elements respectively [4].

In addition to this, the first crystal in the X-ray path is heated by the X-ray beam. This causes changes in the photon energy after the first Bragg reflection, which in turn causes a change in angle for the Bragg reflection from the second crystal. Tight thermal regulation is very import for ensuring stability of photon energies and the X-ray beam position.

Modeling the Balance of Energies

The transfer of heat from one crystal to the cryo head can be modeled according to the thermal conduction Eq. (1), where U is the overall heat transfer coefficient of the cryo head, A is the cross sectional area, and T_{crystal} - T_{cryo} is the temperature difference between the crystal and the cryo head.

$$P_{\mathrm{cryo}} = \frac{Q_{\mathrm{cryo}}}{dt} = -U * A * (T_{\mathrm{crystal}} - T_{\mathrm{cryo}}) \qquad (1)$$

Additionally, the X-ray beam itself imparts heat energy onto crystals it encounters and needs to be taken into account when setting the heater outputs. This power reading is readily available in the control system from an X-ray gas monitor.

* michael.smith@xfel.eu

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP033

Only a small fraction of this energy is transferred to the crystal, and can be modeled with an experimentally-determined reduction factor k.

$$P_{\text{beam}} = \frac{Q_{\text{beam}}}{dt} = k * P_{\text{XGM}} \tag{2}$$

In order to regulate the crystal temperature, an external controllable heat source adds heat to offset the losses to the cryo head and addition from the X-ray beam. The system behaviour of one crystal can be described by an equation that balances the energy transfers of the system. Using Eq. (3), where m is mass and c is specific heat capacity, we can relate energy transfers to one crystal with its temperature change.

$$P_{\text{crystal}} = \frac{dQ_{\text{crystal}}}{dt} = mc\frac{dT_{\text{crystal}}}{dt} \tag{3}$$

$$\frac{dT_{\text{crystal}}}{dt} = \frac{1}{mc} * (P_{\text{cryo}} + P_{\text{heater}} + P_{\text{beam}}) \tag{4}$$

Equation (4) is an ordinary differential equation (ODE) that describes the rate of change of the system state (process variables) of one crystal with respect to its system inputs (energy inputs and outputs). An equation like this can be created for each crystal to describe the rates of change of crystal temperatures in terms of other measurable inputs from the system. With an ODE for each crystal temperature, the Python MPC toolbox can generate a software model and an optimal controller for this system that can be integrated into a XFEL's distributed control system.

SOFTWARE REALIZATION OF AN OPTIMAL MPC CONTROLLER

The Karabo SCADA Framework

The Karabo SCADA Framework is a distributed control system developed specifically for European XFEL facility, consisting of tens of thousands of hardware items and pluggable software components, called devices, that monitor over two million attributes. These devices provide a common software interface to control hardware, read data from sensors, or to create higher-level procedures and functionality by interfacing to other Karabo devices in a hierarchy. The Karabo SCADA Framework provides an Application Programming Interface (API) in Python called the Middlelayer API.

The Python MPC Toolbox

The Python MPC Toolbox is a comprehensive python library that supports creation, simulation, and runtime implementation of MPCs. This toolbox provides a framework in Python for describing a system's behaviour by defining the ordinary differential equations (ODEs) that relate the rate of change of its process variables to other measurable states of the system. Once these are defined, a 'model' instance is created and can be used by other parts of the MPC Toolbox to synthesize an optimal MPC controller as well as a model simulator.

The MPC toolbox's Python API lends itself well for use with the Karabo Middlelayer API. Using the MPC toolbox, one can program the implementation of all the device's behavioural logic into the definitions of the MPC's model. The MPC Toolbox is capable of incorporating arbitrary or non-linear constraints into the model as well. Outside the MPC model definition, the software instructions that remain are needed only to marshall model inputs and outputs to and from the SCADA network.

Integration of the MPC Toolbox

Fixed model parameters are inputs to the model that are not sourced from a remote device in the control network. These are defined in the Karabo device just like other numeric parameters of the device. Karabo has a declarative API which enables the creation of a generic, reusable datatype that can update the MPC model parameters seamlessly.

```
""" example declaration of Karabo parameter
"""
connectionTimeout = Float(
  displayedName="Connection Timeout",
  description="Maximum time to wait for remote
    "
              "device connections.",
  unitSymbol=Unit.SECONDS,
  maxInc=10.0
)

""" example declaration of Karabo parameter
    that auto-updates MPC parameter
"""
temperature1Setpoint = MPCFloat(
  displayedName="XTAL1 Temperature Setpoint",
  description="Temperature setpoint for the "
        "first crystal in the X-ray path",
  unitSymbol=Unit.DEGREE_CELSIUS,
  minInc=-200.0,
  # the datatype's 'alias' is used to declare
  # the name of the MPC model variable
  alias='T_setpoint1'
)
```

Using these derived classes, MPC model parameters are exposed in the Karabo device, can be configured at any time, and the MPC model gets automatically updated. This is used to implement configurable process control setpoints, fixed model constants, and even maximum/minimum boundaries on MPC variables.

Non-linear and other arbitrary constraints can be imposed on the model through the MPC toolbox API as well. A Karabo property can be defined and its value used to set lower and upper terminal bounds on any MPC variable in the model. The MPC toolbox will respect the limits when it calculates the next control step. Enable and disable toggles can also be incorporated into the MPC model this way by creating a Karabo boolean parameter and multiplying the parameter's numeric value of 0 or 1 against a MPC variable in the model.

With these considerations integrated into the Karabo device parameters, the main process control loop in the Karabo device becomes very straight-forward. Model inputs are collected from remote Karabo devices, the next step of the

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP033

controller is calculated, then model outputs are written to remote Karabo devices.

```
while True:
  self.elapsed_time = time.time()

  # read system state from device proxies
  # (get feedback variables from system)
  y = [proxy.value for proxy
       in self.temperatureProxies]

  # calculate next set of control actions
  u = self.mpc.make_step(y)

  # apply control actions (write
  # new values to device proxies)
  self.heaterProxies[0].targetPower = u[0]
  self.heaterProxies[1].targetPower = u[1]

  # time between control actions is defined
  # in the MPC as self.mpc.t_step
  time_step = self.mpc.t_step - \
      (time.time() - self.elapsed_time)
  await sleep(time_step)
```

Simulation and Unit Testing

With a synthesized model, the MPC toolbox can also create a model simulator for the process under control. This is of course very useful for confirming the model equations are defined correctly, and for testing the effect of various inputs on the model. In this case where we have integrated MPC into a Karabo device, we also integrate the model simulator into our Karabo device unit tests.

The Karabo SCADA Framework provides a unit testing API that allows the creation of mock devices for unit testing. Creating mock devices with temperature inputs and heater outputs is a straight-forward effort. With a model simulator, synthesized from the MPC model, the logic loop in the mock devices also becomes straight-forward to implement.

```
self.h1 = MockHeaterControl(
    {"_deviceId_": "MOCK_H1",
     "targetPower" : 0.0}
)
self.t1 = MockAnalogInput(
    {"_deviceId_": "MOCK_T1",
     "value": -177.70028}
)

print('Simulating control response...')
while True:
  # wait for MPC under test to
  # write new heater output values
  await waitUntilNew(self.h1.targetPower,
                     self.h2.targetPower)

  # use those outputs to simulate the
  # next increment in system behaviour
  u0 = [self.h1.targetPower.value,
        self.h2.targetPower.value]

  y_next = self.simulator.make_step(u0)

  # set temperature sensors to
  # simulated values
  self.t1.value = y_next[0]
  self.t2.value = y_next[1]
```

Leveraging the models generated by the MPC Toolbox, the effort of simulating the system behaviour in unit tests is greatly reduced. From this same model, a simulator can be synthesized and programming effort can now be re-directed into writing verification of the MPC controller's behaviour in the unit test framework. For example, a unit test could setup initial conditions meant to drive heater outputs to their maximums, and then test that the boundary conditions in the device configuration:

1. have been passed to the model, and
2. the heater output calculated by the model does not exceed them.

RESULTS

Previously, the setpoint temperature for the monochromators was controlled by manually adjusting the two heater outputs per monochromator until an equilibrium was achieved around -180 °C. The temperature changes over time were monitored using the Karabo control system and occasionally heater adjustments were made.

In Fig. 2, the temperature stability of the system over several hours can be observed, both before and after the MPC regulation is activated. In addition, it is very difficult to tune the system so that both crystal temperatures are exactly the same since heater input from one crystal will eventually affect the temperature of the other. Additionally, the temperatures will also vary depending on the power of the X-ray beam passing though them.

The Karabo device based on the MPC toolkit was deployed to control two monochromator devices in the Femosecond X-ray Experiments (FXE) instrument in September 2022. When the regulator is active, it is able to bring the two crystal temperatures to the setpoint of -180 °C within a few minutes and hold the temperature with a standard deviation of 0.006 °C. A zoomed-in view of the regulator being activated is shown in Fig. 3. The process to be controlled has quite a slow time scale, so it was sufficient for the MPC to calculate and update the heater outputs once every 10 seconds to get this result.

The constant parameters in the MPC formula were exposed as Karabo device attributes and were easily tunable during testing to find ideal values. Once these values were determined, they were used by the software as device configuration for the unit tests. This allows the simulator to respond the same way as the actual system would for our software unit tests.

The overall result is improved temperature stability in the monochromator devices. In addition, the simulator synthesized based on the same model use to synthesize the controller can be used to observe how the controller would behave under any initial conditions defined in a software test.

CONCLUSION

This implementation demonstrates that it is possible to capture all the control logic of a high-level process control device into a software model using the MPC Toolbox.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP033

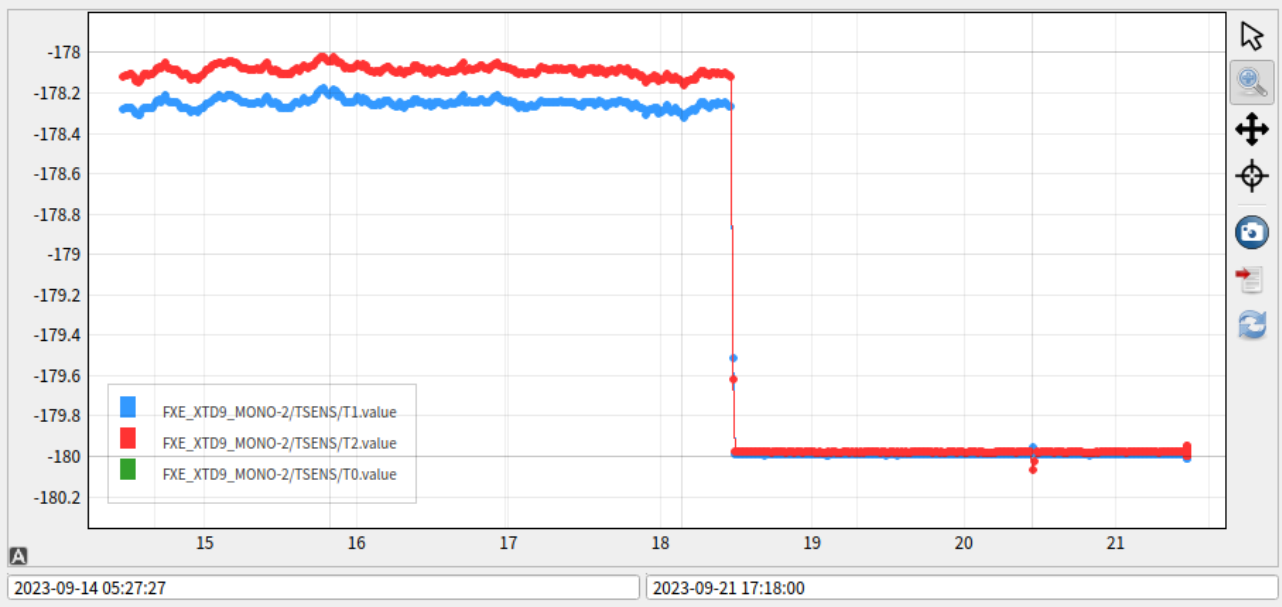

Figure 2: Temperature plot of both monochromator crystals in °C, showing the temperature regulation before and after the MPC is activated.

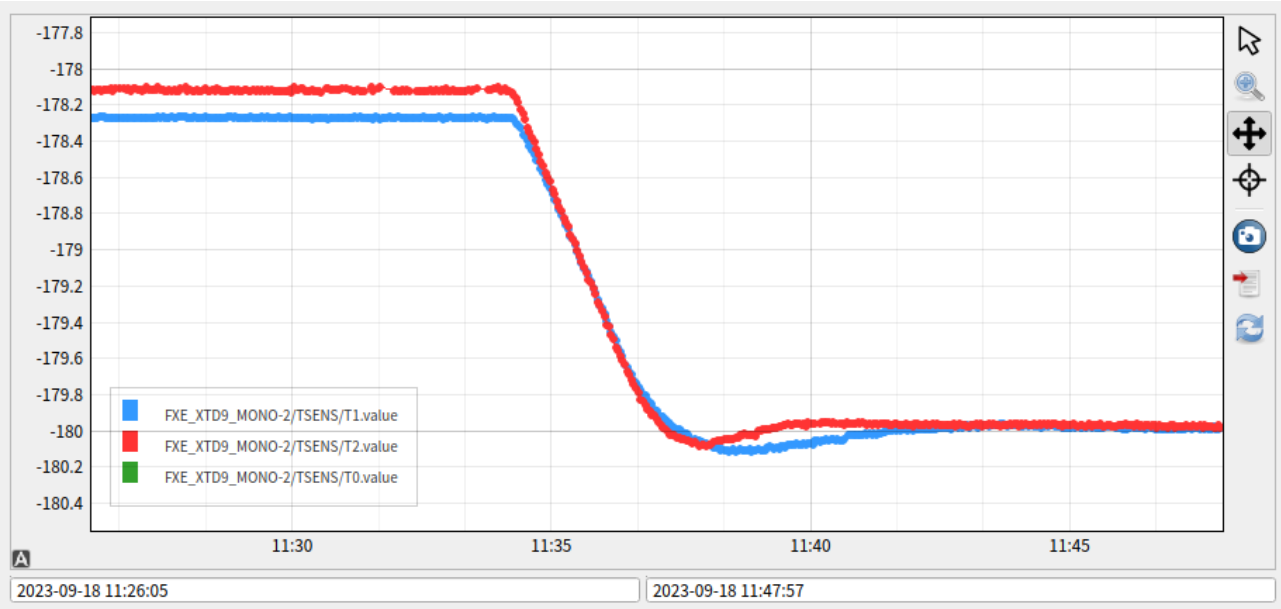

Figure 3: Temperature plot of both monochromator crystals in °C, showing the transition with only a minor overshoot of less than 10%.

Once the model has been synthesized, the effort of writing a control system device is reduced to defining device configuration for the model's constant parameters and writing code to read and write attributes from other remote control system devices. Additionally, the model is fully reusable for unit testing and simulation so there is no duplicated effort in writing code to model the system behavior for the purposes of simulation or unit testing.

The resulting software device now serves as a template for implementing a Karabo device to realize model predictive control.

REFERENCES

[1] S. Hauf *et al.*, "The Karabo distributed control system", *J. Synchrotron Radiat.*, vol. 26, no. 5, pp. 1448–1461, Sep. 2019. doi:10.1107/S1600577519006696

[2] The Python Model Predictive Control Toolbox, 2023, https://www.do-mpc.com

[3] T. Middelmann *et al.*, "Thermal expansion coefficient of single crystal silicon from 7K to 293K", *Phys. Rev. B*, vol. 92, p. 174113, Nov. 2015. doi:10.1103/PhysRevB.92.174113

[4] H. Sinn *et al.*, "X-Ray Optics and Beam Transport", EuXFEL, Hamburg, Germany, Rep. XFEL.EU TR-2012-006, Dec. 2012.