ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP113

A FLEXIBLE EPICS FRAMEWORK FOR SAMPLE ALIGNMENT AT NEUTRON BEAMLINES

M. Henderson, J. Edelen*, M. Kilpatrick, RadiaSoft LLC, Boulder, USA
S. Calder, B. Vacaliuc, ORNL RAD, Oak Ridge, USA
R. D. Gregory, G. Guyotte, C. Hoffmann, B. Krishna, ORNL, Oak Ridge, USA

Abstract

RadiaSoft has developed a flexible front-end framework, written in Python, for rapidly developing and testing automated sample alignment IOCs at Oak Ridge National Laboratory. We utilize YAML-formatted configuration files to construct a thin abstraction layer of custom classes which provide an internal representation of the external hardware within a controls system. The abstraction layer employs the PCASPy and PyEpics libraries in order to serve EPICS process variables and respond to read/write requests via Channel Access, with future developments planned for PV Access through the P4P library. Our framework allows users to build a new IOC that has access to information about the sample environment in addition to user-defined machine learning models and data processing methods. The IOC monitors for user inputs, performs user-defined operations on the beamline, and reports its status back to the control system. Our IOCs can be booted from the command line, and we have developed command line tools for rapidly running and testing alignment processes. These tools can also be accessed through EPICS GUIs or separate Python scripts. These proceedings provide an overview of our software structure and showcases its use at two beamlines at ORNL.

INTRODUCTION

Robust, accessible controls software for beamline sample environments are a critical need for operators at neutron science user facilities like the Spallation Neutron Source (SNS) and High-Flux Isotope Reactor (HFIR) at Oak Ridge National Laboratory (ORNL). Existing input-output controller (IOC) software and control workflows allow beamline operators to meet the needs of users, but require operators to expend significant amounts of time and resources on trivial tasks like sample alignment which make poor use of their expertise and are good targets for automation. Additionally, the creation of new workflows or the extension of existing ones typically require appreciable effort from controls experts who already experience heavy workloads unrelated to day-to-day beamline operations at user facilities. To that end, RadiaSoft has developed *rscontrols*: a flexible front-end controls framework, written in Python, for rapidly developing IOCs with embedded machine learning (ML) models and other modern automation tools. *rscontrols* leverages common configuration tools and a Pythonic API to enhance user accessibility and is built on existing Python packages for implementing controls via the EPICS framework.

* jedelen@radiasoft.net

SUPPORTING PACKAGES

In addition to supporting broad user accessibility, our choice to employ Python for the *rscontrols* framework has allowed us to take advantage of pre-existing tools for executing controls operations through EPICS. For client-side access to existing networks of EPICS process variables (PVs) we use the PyEpics package [1], a Python API for the channel access (CA) protocol within EPICS. For serving new PVs associated with the *rscontrols* framework to the network we use the PCASPy [2] package, a Python API for the Portable CA Server (PCAS) support module for EPICS.

USER INPUTS

One of the strengths of the *rscontrols* framework is the reduced workload placed on users in comparison to pure EPICS or the Python APIs employed by *rscontrols*. To create new IOCs using those tools typically requires significant development time and the efforts of expert programmers. Even in the case of Python APIs like PyEpics and PCASPy, these efforts generally produce IOCs dedicated to specific processes that therefore feature low levels of reusability. In contrast, user inputs for *rscontrols* IOCs have been reduced to a single configuration file with a human-readable layout and hooks for control and server processes which take the form of simple Python functions. This allows new IOCs to be developed quickly by operators familiar with local beamline equipment and processes and a basic understanding of Python scripting.

Configuration Files

The primary user input for an IOC created with *rscontrols* is a YAML-formatted configuration file. This file provides a list of all hardware elements to be abstracted by the software, including the PVs associated with each element, as well the available ML models, control processes, and any new PVs to be served by the IOC (see Appendix).

Controls Process Scripts

One of the most important aspects of *rscontrols* is its method of representing user-defined controls processes as Python functions. A single Python module containing these functions is specified by the user via the *path* entry of the *Processes* section of a config file. Only processes listed in the config file are imported for use into the framework, though functions in the processes module can freely depend on one another (including functions which do not correspond to any active processes).

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP113

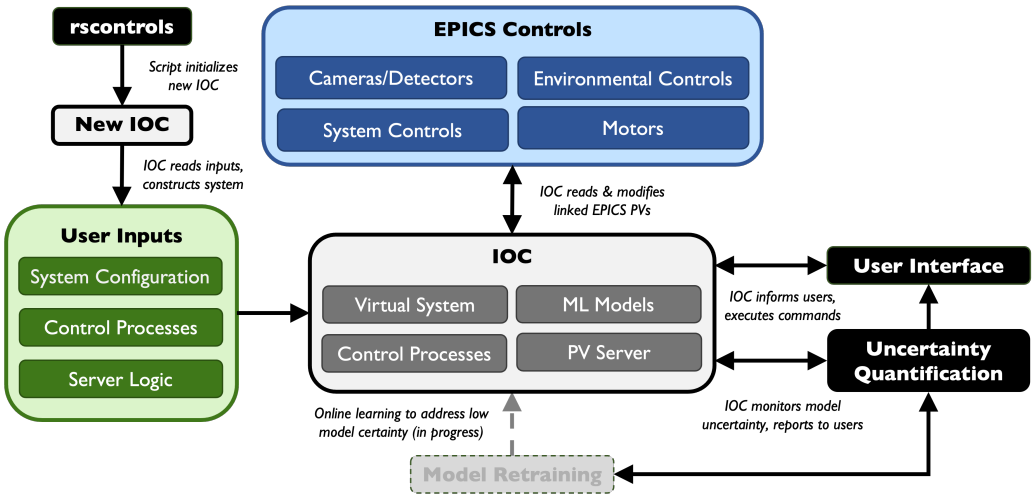

Figure 1: Overview of the *rscontrols* runtime workflow, including ongoing work with online learning.

Server Logic Scripts

Similar to controls process scripts, units of server logic are provided to *rscontrols* by users in the form of Python functions. Unlike process scripts, the functions in server logic scripts are more limited in nature as they generally correspond to EPICS operations for *get* or *put*, or to describe the reaction associated with triggering the PROC field of PV. Server logic modules can also contain support functions that don't get imported but instead are called by primary functions which do, helping to reduce reproduction of code.

Machine Learning Models

In support of process automation, *rscontrols* features tools for deploying trained ML models defined using the *Models* section of a configuration file (see Appendix). The entries needed to define a model are a model type, a path to trained model weights (stored as an hdf5 file), and a path to any model hyper-parameters needed to instantiate a member of the model class (stored as a dictionary of named parameters in a Python "pickle" file). Current development efforts include work to make model definitions more general and require fewer input files, while maintaining support for models implemented with either PyTorch and Tensorflow.

As the first ML-based tasks undertaken by the framework have been in the area of machine vision (see Deployment & Testing), all currently available models are variants of UNet [3] segmentation networks, though support for a broader variety of models is planned. Ultimately, this will also include non-ML models (some of which, e.g. standard denoising filters, are currently implemented as processes).

SYSTEM ABSTRACTION & OPERATION

Hardware & Machine Learning Models

Parameters defined in the *System* section of the user-provided configuration are used to initialize virtual hardware components in the abstraction layer. Similarly, hyper-parameter and network weight files listed in the *Models* section are used to load trained ML models. During operation, elements and models are used in much the same way, being passed as arguments to controls process and server logic functions. The main difference between their uses in these contexts is that hardware elements feature hooks for accessing EPICS PVs, whereas models generally feature methods for processing data (e.g., motor positions or images).

Control Process & Server Logic Hooks

Python scripts provided by users to define control processes and server logic function as hooks in the larger program. Each function defined in the scripts represents a distinct unit of activity in the *rscontrols* framework. Functions defining processes can be executed through any of the user interfaces (see the User Interface Tools), at which point the functions are called and passed their respective input arguments. Server logic functions on the other hand are called any time the network receives a request for the value(s) of a served PV (in the case of *get* functions), or any time a new value is written to one (in the case of *put* functions).

Workflow

Given the dynamic and interactive nature of controls software, *rscontrols* was designed with a well-defined workflow in mind (see Fig. 1). At startup, a new IOC object is initialized and proceeds to read the supplied configuration file and other user inputs. These inputs are parsed and used to construct the abstraction layer components of the IOC, which then enters into its main operational loop. Once in operation, the IOC continuously and asynchronously scans for inputs from the user via the CLI (if one is running) or the EPICS interface, as well as changes to the PVs associated with hardware elements. When processes or server operations are executed, the IOC forwards their inputs (hardware elements, machine learning models, or combinations of both) to the appropriate Python functions imported during startup.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP113

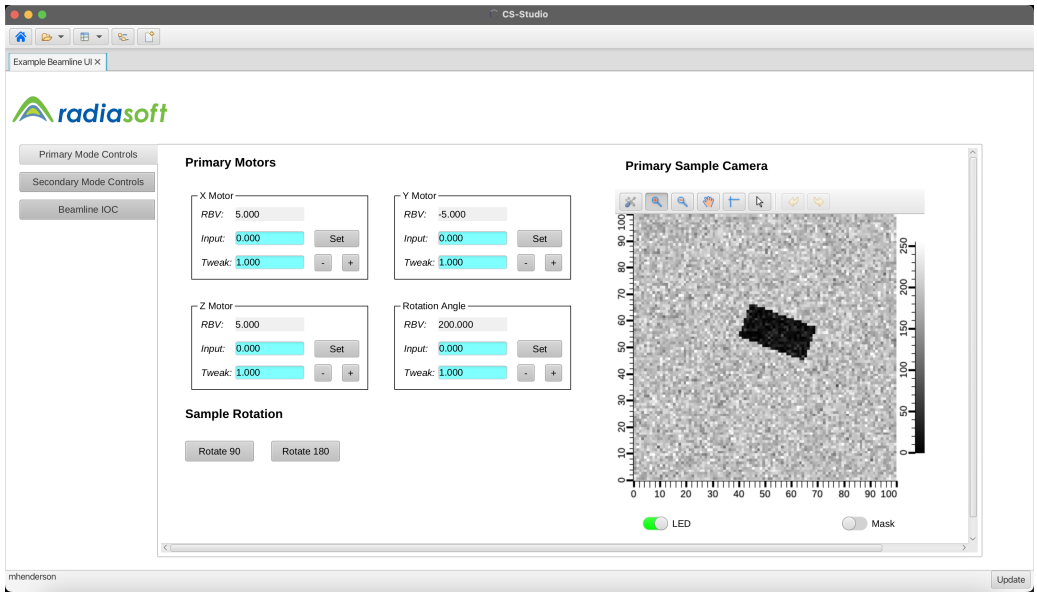

Figure 2: An example GUI interface using CS-Studio for a control system simulated entirely with *rscontrols*.

USER INTERFACE TOOLS

Command Line Interface

The *rscontrols* framework is distributed with a command line interface (CLI) for quick and easy access. The CLI is initiated using the *rscontrols* executable installed with the package and provides direct access to a lists of hardware elements and control processes, and inputs for process execution. While many users will prefer to embed *rscontrols* functionality into a GUI (or interact directly through EPICS, for low-level systems), the CLI provides a simple access point that is always available "out of the box".

EPICS Interface

Although one purpose of *rscontrols* is to facilitate the operation and extension of controls systems by providing an abstraction layer and UI tools, other purposes include the automation of controls processes and exposure of processed data on-network. Since processes and results can also be exposed by *rscontrols* as EPICS PVs, users who prefer to operate strictly through EPICS (e.g., using *caget* and *caput*) can do so. This helps to maintain the flexibility of the framework and extends its usefulness to systems for which only direct command line access is available but automated controls are still desirable (such as low memory systems).

GUI Compatability & Support

Since *rscontrols* operates using EPICS, integration with GUI applications like Control System Studio (CSS) is straightforward. The names and properties of controls equipment as well as the names of implemented processes can be read directly from PVs and used to populate controls screens. Additionally, tools capable of embedding Python scripts directly are free to employ *rscontrols* through the Python API, allowing entire IOCs to be embedded into displays.

DEPLOYMENT & TESTING

The origin and initial test setting for the *rscontrols* framework is a collaborative project between RadiaSoft and Oak Ridge National Laboratory (ORNL) for ML-based stabilization of neutron scattering sample environments. In the context of that project, *rscontrols* has so far been deployed to and tested at two different beamlines: the TOPAZ beamline at the Spallation Neutron Source (SNS) facility [5] and the HB2A beamline at the High-Flux Isotop Reactor (HFIR) facility [4]. Though improvements to these individual deployments are ongoing, early test results for the processes already implemented have been promising.

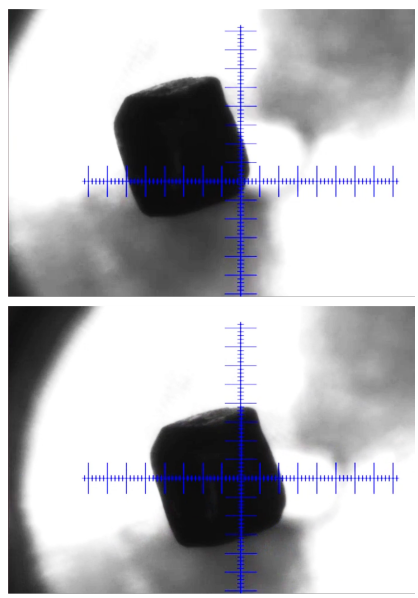

Figure 3: Sample images from the TOPAZ beamline at ORNL before (top) and after (bottom) conducting an automated alignment process with *rscontrols*.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP113

The first live tests of our controls framework were undertaken at the TOPAZ beamline, where we successfully ran an automated alignment process (see Fig. 3), in addition to several preliminary tests with more basic controls. One challenging aspect of these tests was cooperating with the existing controls framework at TOPAZ to replace previously human-in-the-loop portions of the alignment workflow. Given the well-established nature of controls systems at TOPAZ and other beamline facilities, achieving this level of flexible modularity with *rscontrols* was one of key goals for future deployments of the framework.

Unlike TOPAZ, the HB2A beamline previously lacked any level of controls automation, with all processes being carried out manually by operators. Controls on the HB2A beamline are also implemented using an in-house software called SPICE, rather than EPICS. To bridge the system with our EPICS-based software, our collaborators among the controls group at ORNL developed an interface software for mediating between network variables used by SPICE and EPICS PVs. In addition to providing us a means of connecting to the SPICE-based controls at HB2A using *rscontrols*, this bridge software has greatly improved the existing manual controls workflow at HB2A, including facilitating integration with CS-Studio GUIs, and is now being deployed on other instruments at various ORNL facilities.

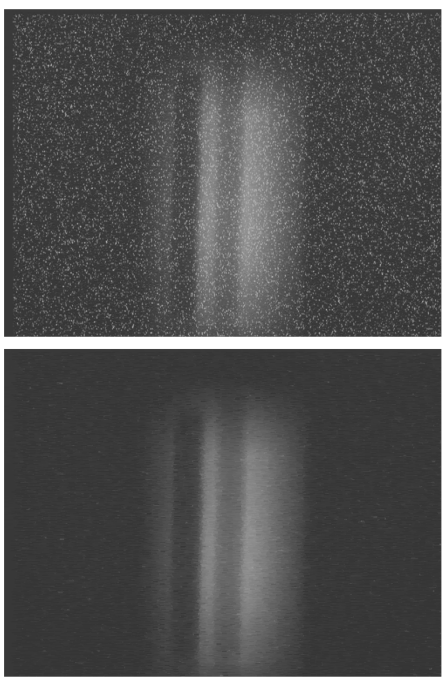

Figure 4: Neutron camera images of a sample at HB2A before (top) and after (bottom) denoising with *rscontrols*.

Enabled by the SPICE-to-EPICS interface software, several live tests of *rscontrols* have now been conducted at the HB2A beamline. These have included simple tests for camera connectivity and readout, basic motor controls, and the deployment of a real-time image denoising process for the beamline's diagnostic neutron camera. This denoising process creates a new PV for denoised image data usable by both operators or automated controls process, and can employ either ML- or filter-based algorithms to process the raw neutron camera images. Due the reliance of alignment workflows (manual or otherwise) on the diagnostic camera, and the tendency for the signal-to-noise ratios of neutron cameras to worsen over time, we anticipate that this functionality will be of continued benefit to operators at HB2A. We are currently exploring the possibility of deploying similar denoising protocols for other instruments at ORNL facilities which rely on neutron cameras for diagnostics and controls.

CONCLUSION

We have produced a flexible new framework, *rscontrols*, for the rapid production and deployment of EPICS IOCs with a focus on controls automation. The *rscontrols* framework achieves a high level of abstraction for maximizing user accessibility and was developed through live testing on the TOPAZ and HB2A neutron beamlines at the SNS and HFIR facilities (respectively) at ORNL. Engaging with operators at these facilities, we have successfully demonstrated the automation and streamlining of controls processes like sample alignment, and have provided extensions to existing controls and diagnostic data including continuous, live denoising for neutron camera images.

Although the core source code for *rscontrols* has reached a relatively stable state of development, incremental improvements to the framework are ongoing. Whereas previous development has been focused on the EPICS CA protocol as it is generally the protocol used at ORNL, the addition of the PV Access (PVA) protocol to our framework using the P4P (PVA for Python) library [6] is already well underway. One of the most important planned improvements to *rscontrols* is the integration of uncertainty quantification metrics like segmentation uncertainty maps, which have already been tested and served as PVs, with automated alignment and other controls procedures. Another planned improvement is the addition of ML models other than UNets, including non-network ML models like Bayesian processes, as well as non-ML models for controls such as simple PID controllers.

ACKNOWLEDGEMENTS

This work is supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, SBIR and STTR Program under Award Number(s) DE-SC0021555.

APPENDIX

The listing below provides excerpts from an example *rscontrols* configuration file. The listing includes each of the possible sections of a configuration file, though in practice not necessarily all sections must exist. For example, only a *Processes* or *Server* section must exist, not necessarily both.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP113


```
IOC :
  protocol : CA
  modes : [primary, secondary]

System :
  name : Example Beamline
  prefix : EXBL

  PrimaryCam :
    type : Detector
    prefix : PCam
    modes : [primary]
    dimensions : [100, 100]
    data_pv : Image

#--------------(break)----------------

Server :
  path : server_functions.py
  prefix: EXBL
  pvs:

  PCam:CleanIm :
    count: 10000
    get:
      function: denoise_cam
        args:
          model: DenoiseUNet
          cam: PrimaryCam
    put: None

#--------------(break)----------------

Models :

  DenoiseUNet:
    type : UNet
    weights: model.h5
    architecture: parameters.pkl

#--------------(break)----------------

Processes :
  path : exbl_processes.py

  align_sample :
    primary:
        function : auto_align
        args :
          model: MaskUNet
          cam : PrimaryCam
          controls: PrimaryControls
    secondary:
        function : auto_align
        args :
          model: MaskUNet
          cam : SecondaryCam
          controls: SecondaryControls
```

REFERENCES

[1] M. Newville, TyEpics Documentation, https://pyepics.github.io/pyepics/

[2] X. Wang, PCASPy Documentation, https://pcaspy.readthedocs.io/en/latest/

[3] O. Ronneberger, P. Fischer, T. Brox, "U-net: Convolutional networks for biomedical image segmentation", *MICCAI'15*, vol. 18, no. 3, 2015. doi:10.48550/arXiv:1505.04597v1

[4] S. Calder *et al.*, "A suite-level review of the neutron powder diffraction instruments at Oak Ridge National Laboratory", *Rev. Sci. Instrum.*, vol. 89,no. 9, p. 092701, Sep. 2018. doi:10.1063/1.5033906

[5] L. Coates *et al.*, "A suite-level review of the neutron single-crystal diffraction instruments at Oak Ridge National Laboratory", *Rev. Sci. Instrum.*, vol. 89, no. 9, p. 092802, Sep. 2018. doi:10.1063/1.5030896

[6] M. Davidsaver, P4P Documentation, https://mdavidsaver.github.io/p4p/index.html