ECROC: Integrated run control

Hussain Al Saleh, Miles Armour, **Daniel Foulds-Holt**, Jingyuan Shi†

Detector Informal Meeting PPE
29th October 2026





THE UNIVERSITY of EDINBURGH



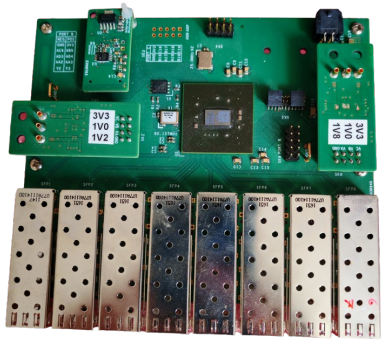
UNIVERSITY OF CAMBRIDGE †

The premise

Taking data in the lab is rarely a synchronized process, there are many systems that do not always talk to each other well.

- HV
- Position stages
- Chip configuration
- DAQ
- Laser / Monochromator

A reliance on manually changing setting limits our ability to do detailed scans of experimental conditions! Therefore, we wanted to create a centralised control software that handles data taking alongside changing the experimental configuration to enable easy automated scans across multiple systems.











***The* Edinburgh-Cambridge Run Operator *and* Controller**

The repository is here: ecroc

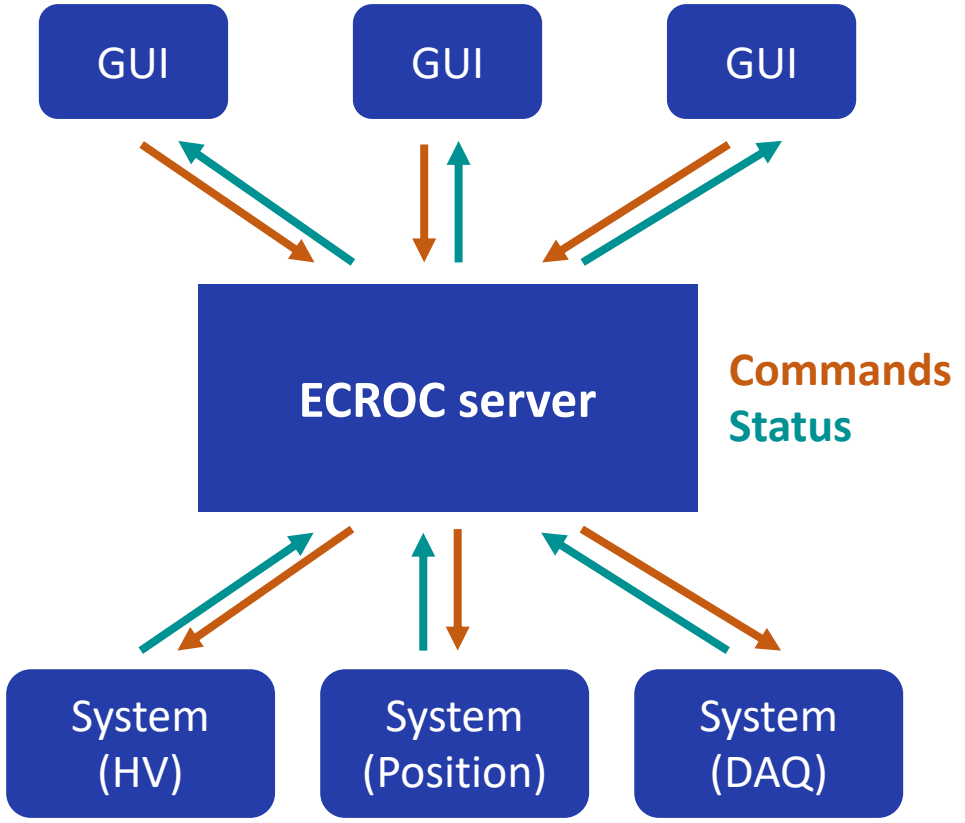
Further reading on Kapitsa's crocodile.

ECROC

ECROC is a control system inspired by the the LHCb WinCC control system, and the telescope run control software EUDAQ.

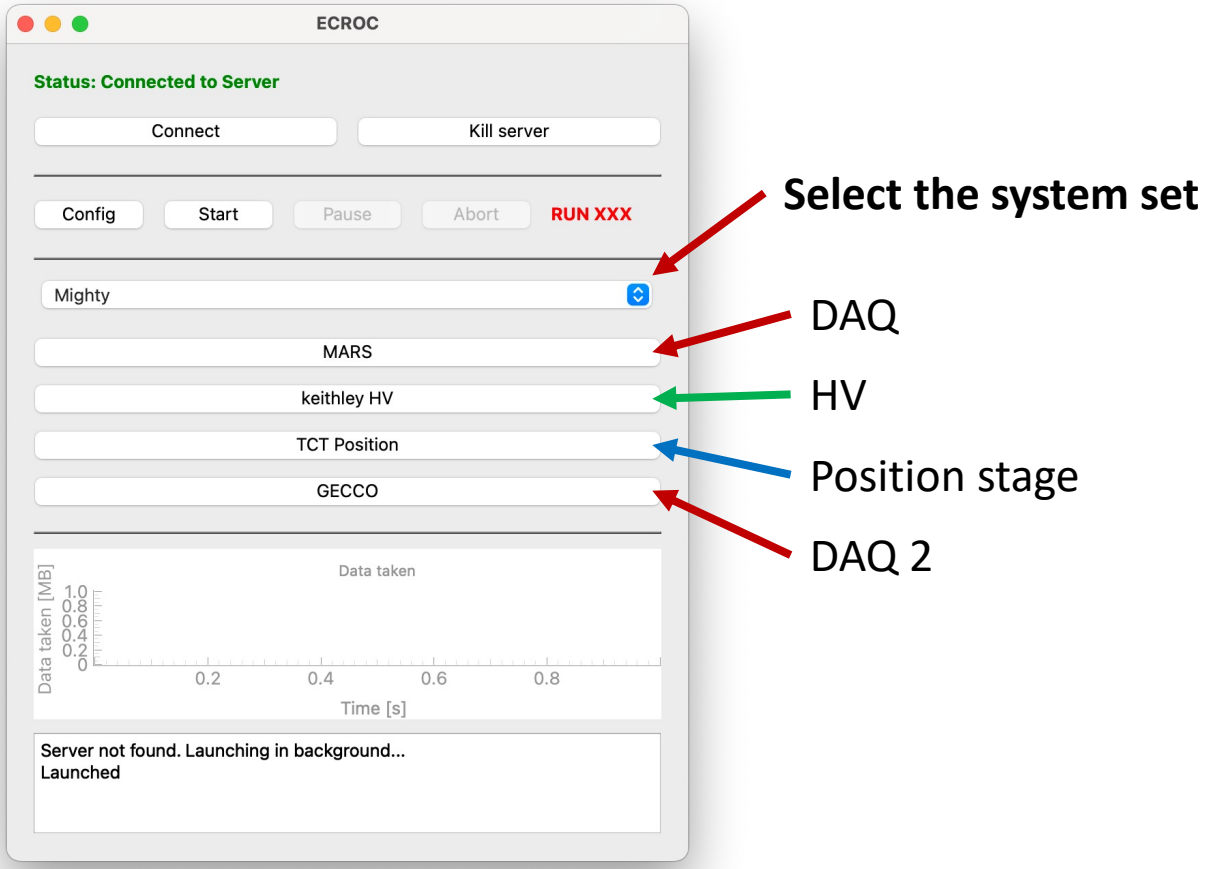
It runs a single server as an FSM *(-ish)* that holds objects that communicate with the various systems (HV, position, etc) in their own threads. The server can be communicated with using a GUI connection.

This has an advantage of enabling multiple GUIs to see data and give inputs (e.g. you can ssh into the PC and input commands without restarting any software)



The main window

The main window is the launch pad for various GUIs and holds the top-level / system agnostic control.



Select the system set

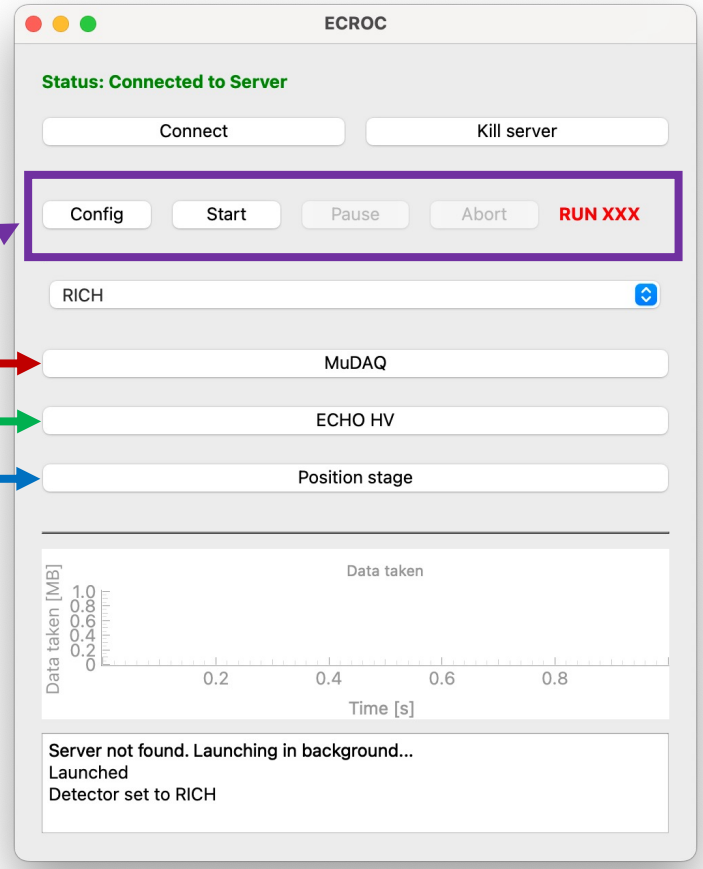
DAQ

HV

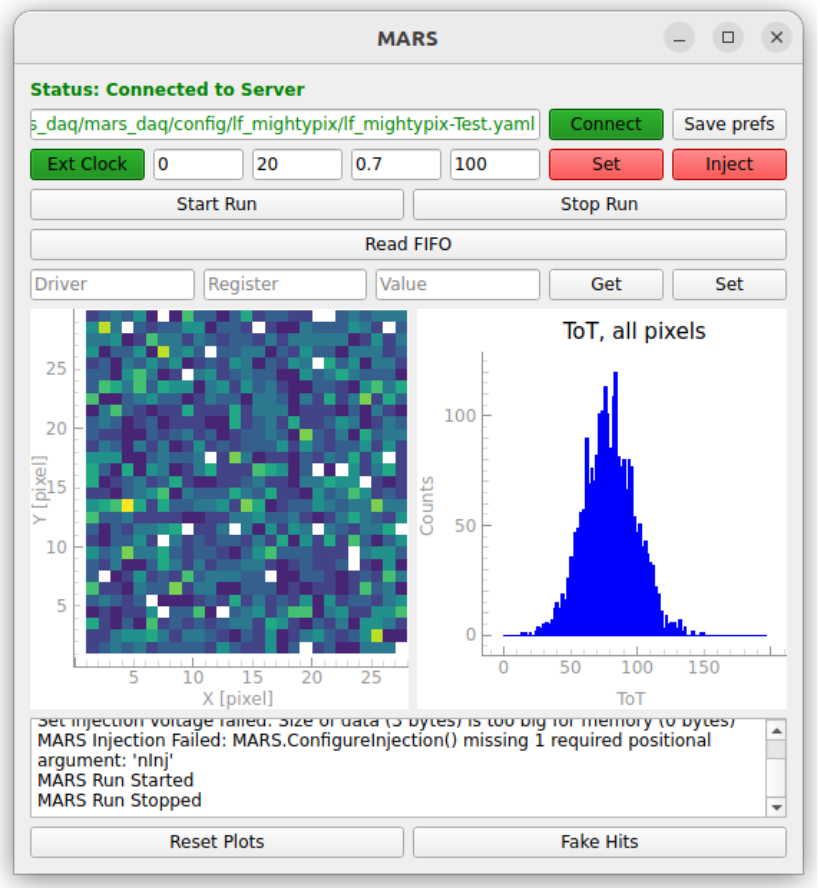
Position stage

DAQ 2

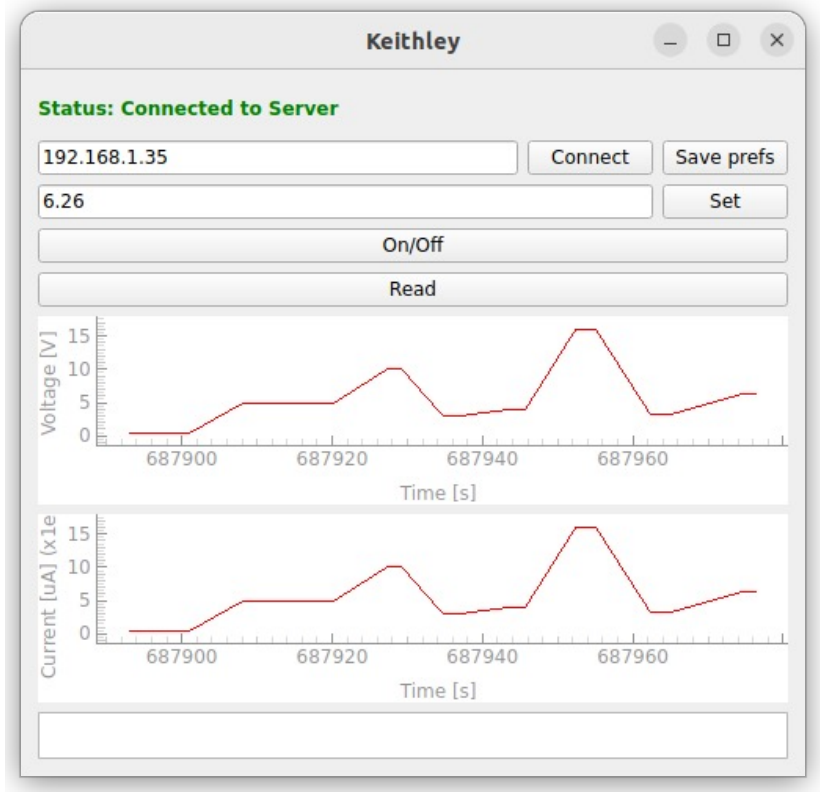
Run control...
We will return to this



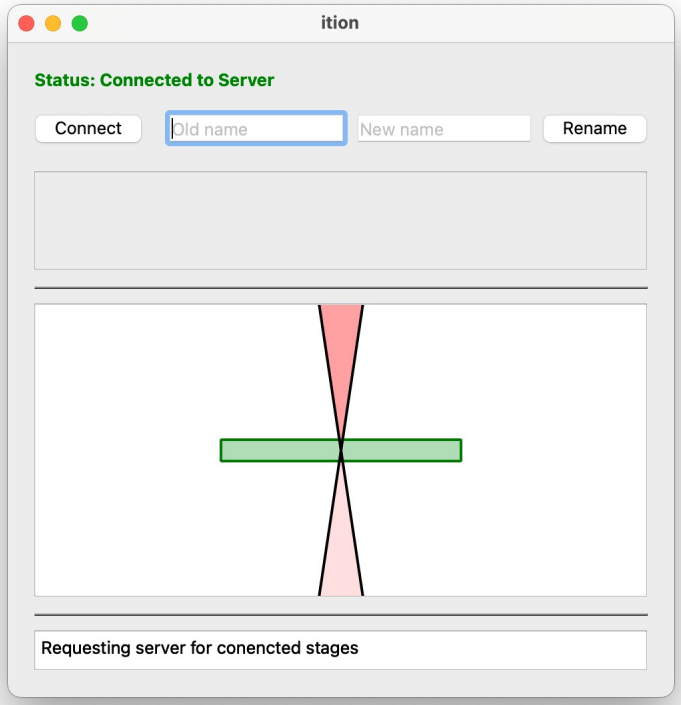
Systems



DAQ



HV



Position stage

Systems are the various bits of equipment that you want to control and coordinate.
Each one has its own GUI and backend implementation.

Run control

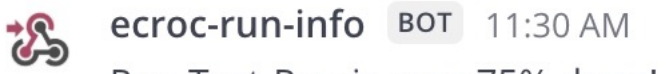
Scans can be easily configured to run over a **range** of **system** **commands**.

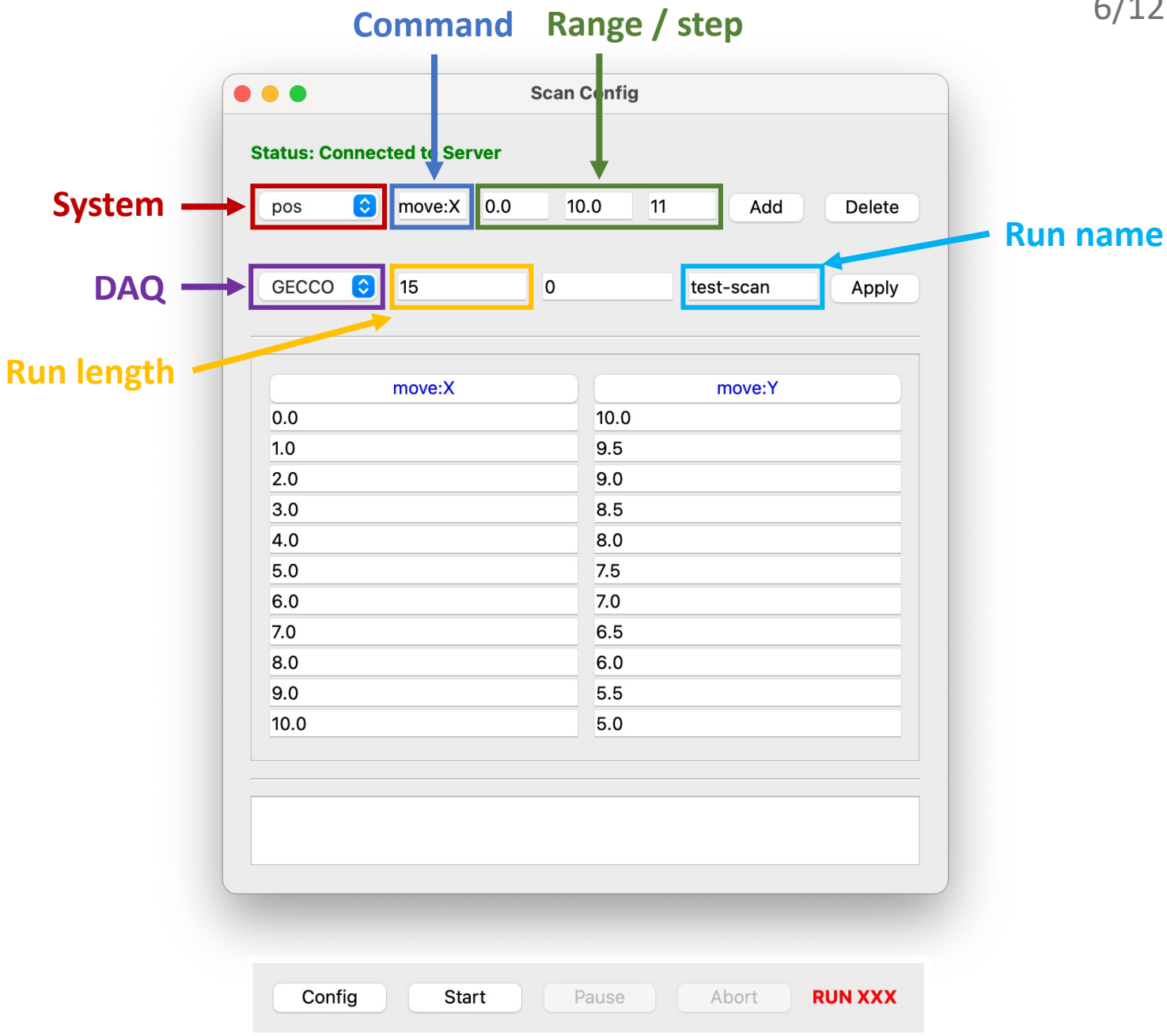
e.g. move a position stage or set a HV voltage.

The run control will set the system settings according to the scan, then start a run using the selected **DAQ** and wait a **user defined length** before stopping the run and moving to the next step.

Data and a log file is saved in a new run folder named after the **run name**.

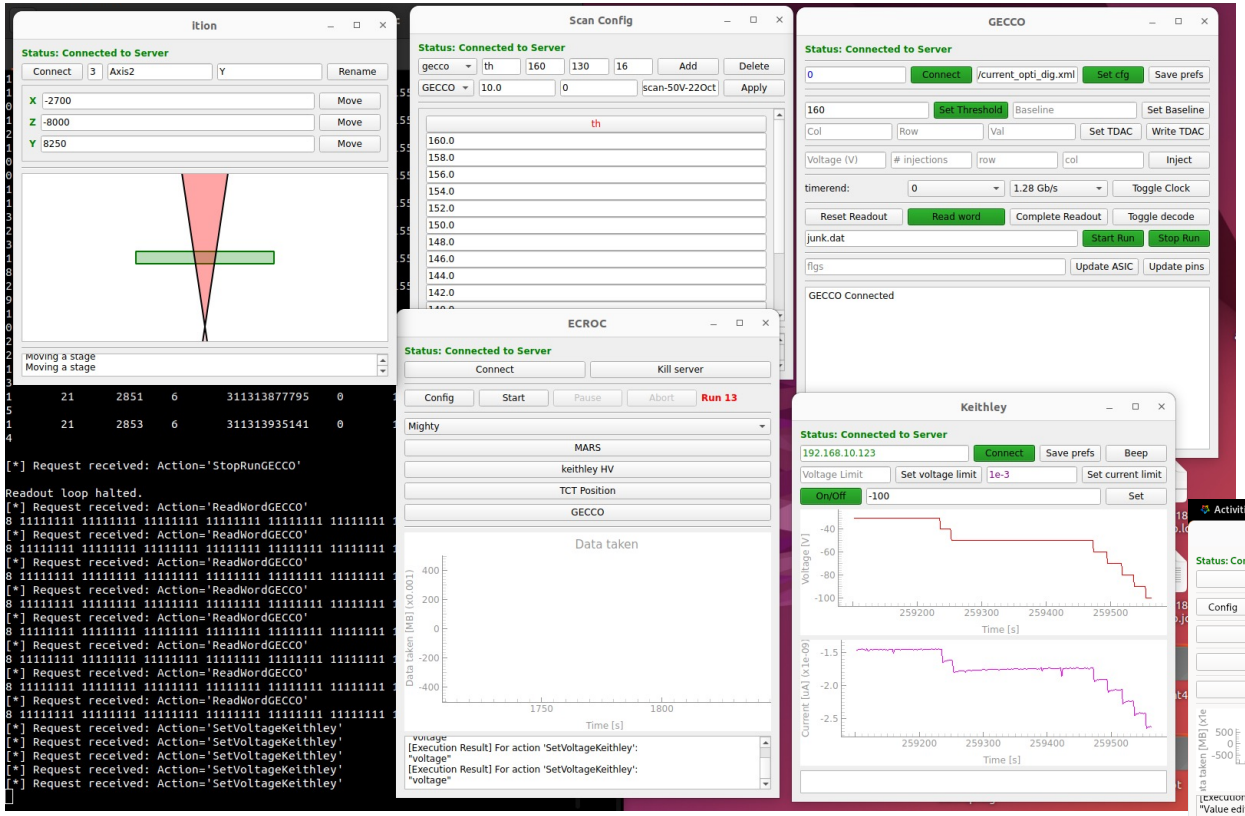
It can be configured to send a mattermost message to update you on the run progress:





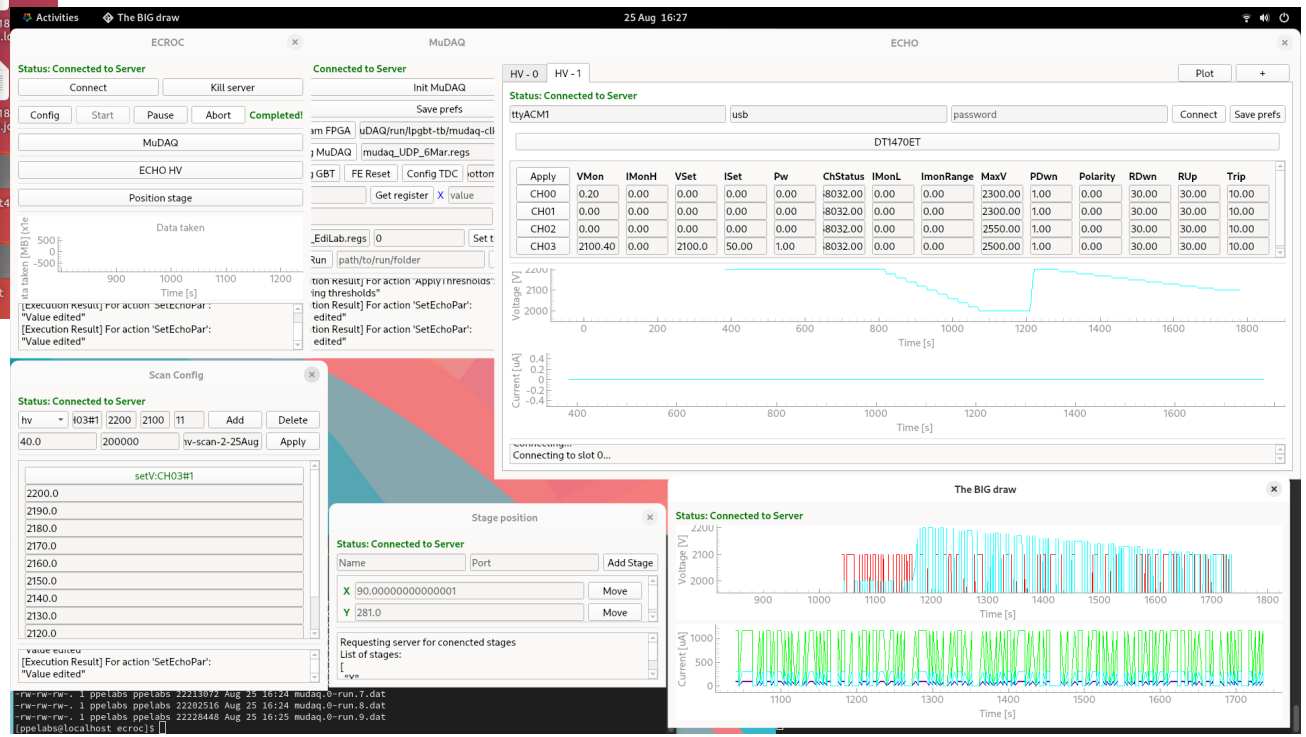
Config, start, pause, and abort on the main panel

Proof of life

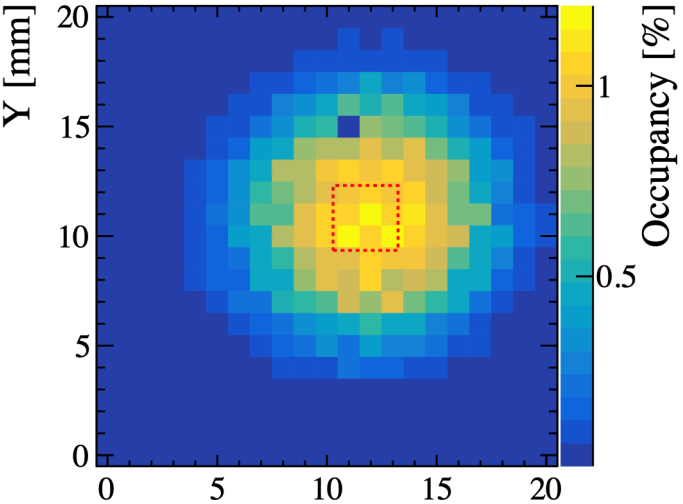


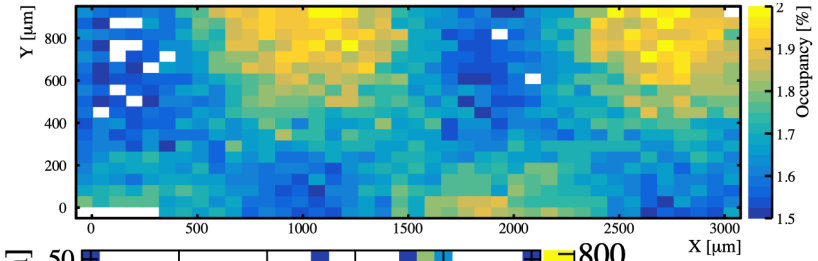
MightyPix communicates with the sensor readout (see Hussian's talk), Keithley HV, and TCT position stage

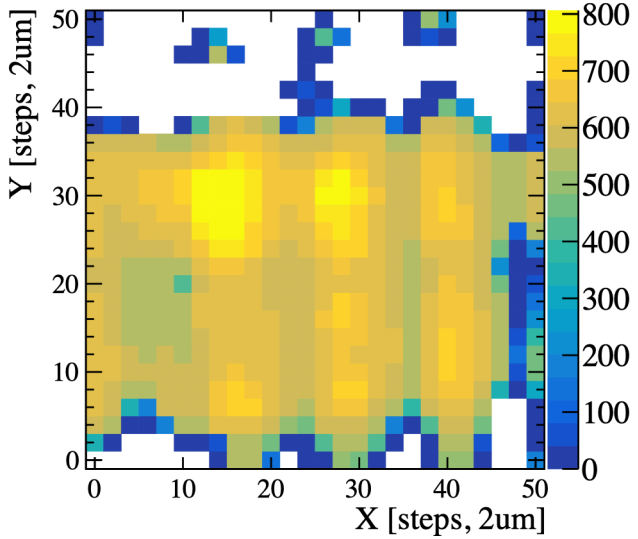
RICH communicates with the MuDAQ readout, CAEN HV, and ThorLabs position stage



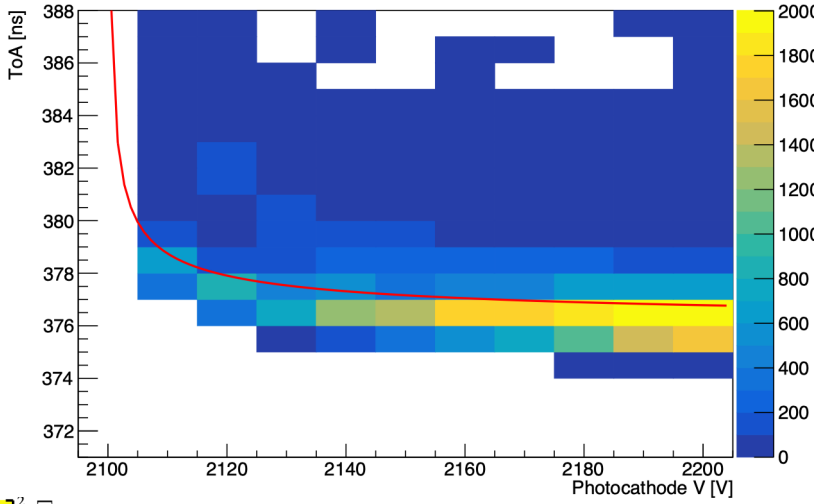
What can this do?

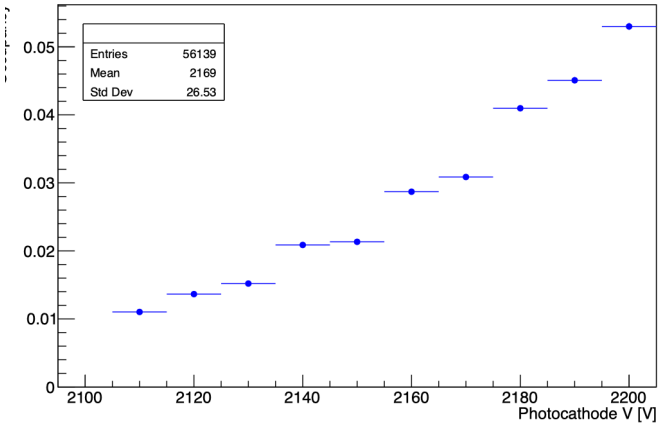




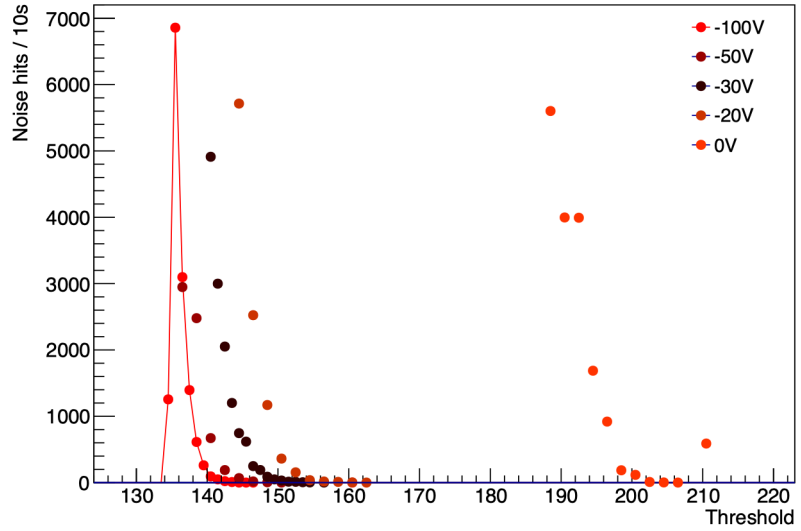


Position scans





HV scans



Threshold scans

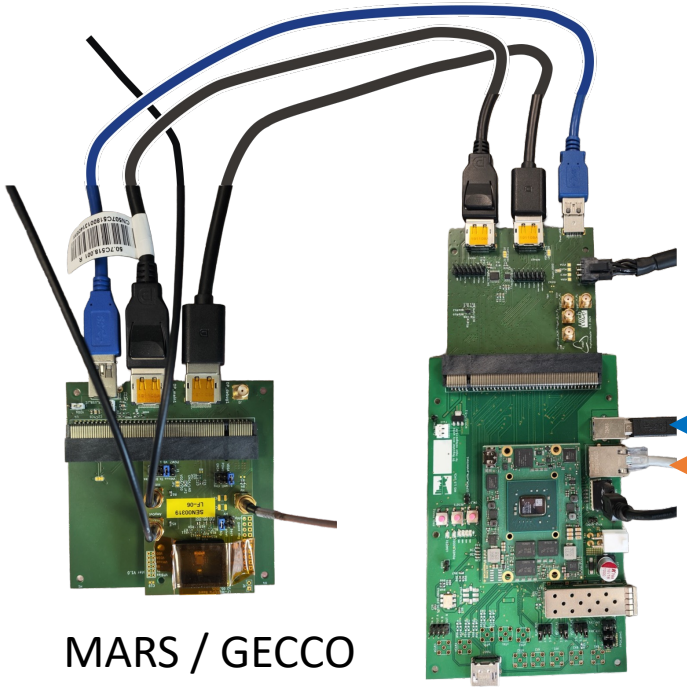
Here are a few of the studies we have used it for, but in general…

Lots!

More details in later talks!

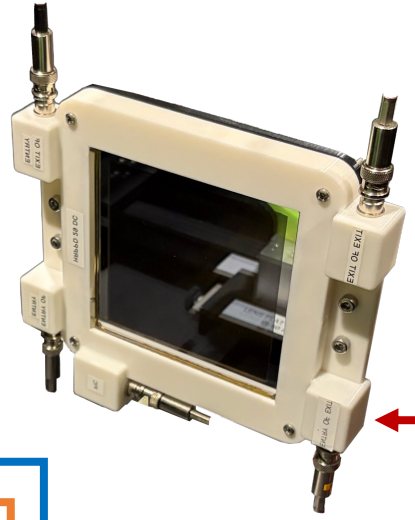
Jack, *Miles*

Possible use as a TB control

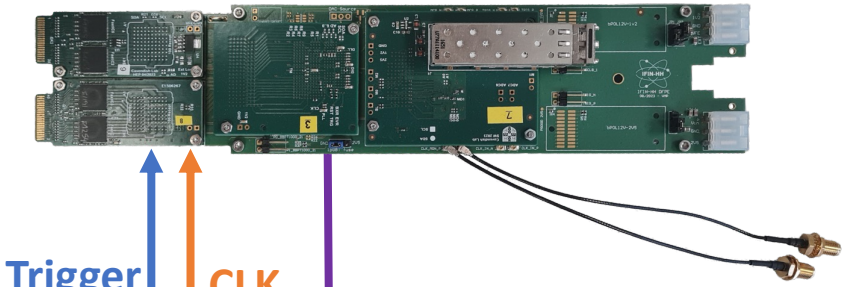


MARS / GECCO

Offline alignment using trigger number.



PicoTDC



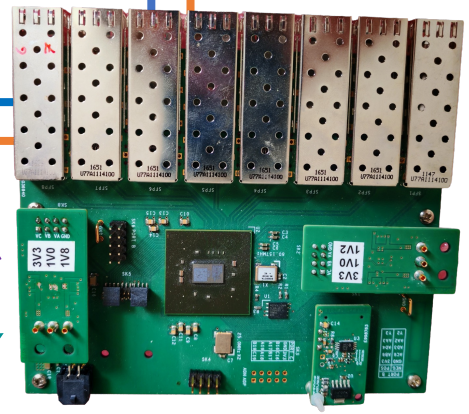
Trigger

CLK

Data

Trigger

CLK



MuDAQ

HV
200V

Keithley HV

Data
Commands
Status

GUI

ECROC

CAEN HV

HV
2000V

Adding a system

Adding a new system is not entirely trivial, but easy enough once you understand how.

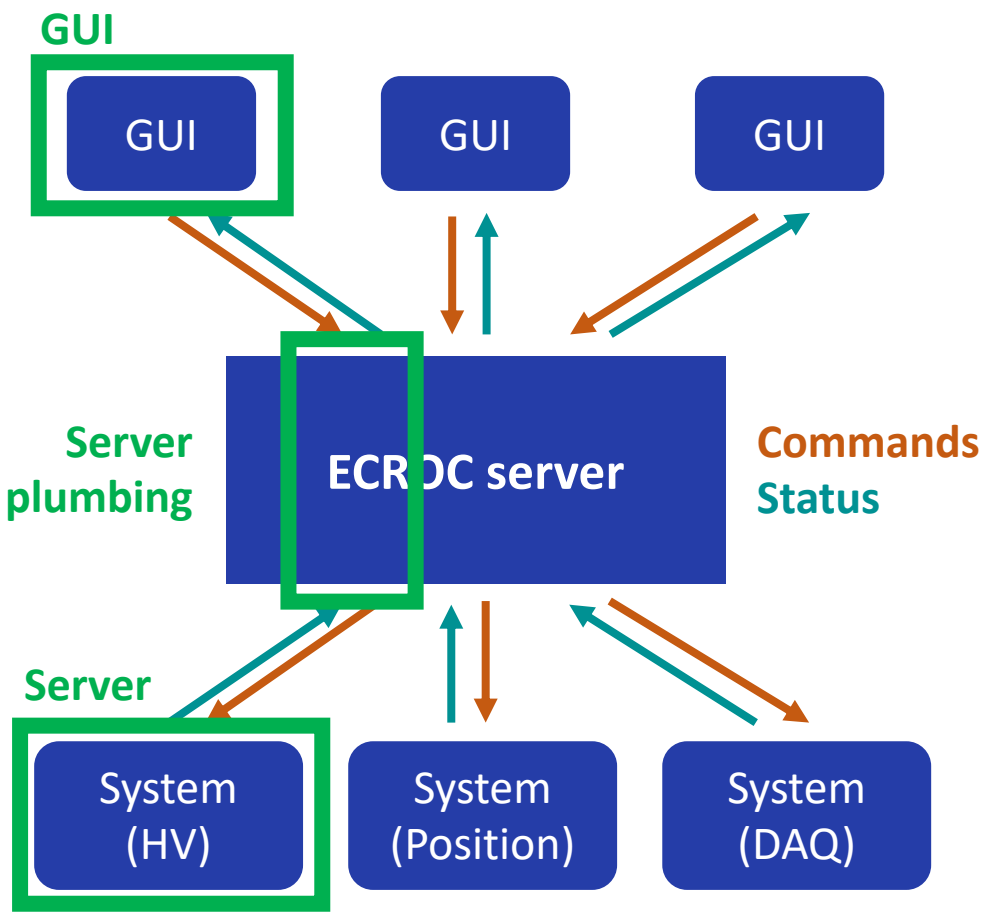
It requires three main changes. A GUI, a python module that executes the control code, and some plumbing to pass information between these two through the server.

You only need to add two files:

ecroc / ECROC_GUI / **Keithley.py**
ecroc / ECROC_SERVER / **Keithley.py**

Importantly! If you are interested in using ECROC but doing this sounds like too much effort. Worry no more! ***We will do the GUI / server wiring for you!***

Just provide us with a python module to control the equipment and we will do the rest!



But my equipment doesn't have a python control module...

Don't panic! This is likely a solved problem.

We have compiled C++ classes into python modules. We have called compiled Java functions in python, we have used random sketchy stuff from online.

It all works well.

```python
class MuDAQ:
    def __init__(self):
        ## Start the java virtual machine
        if not jpype.isJVMStarted():
            jpype.startJVM(
                ## This is again hardcoded!
                "/home/ppelabs/Desktop/MuDAQ/install/PDMDBKIT/pdmdb.lnx/lib/server/libjvm.so",
                "-Djavax.xml.bind.JAXBContextFactory=com.sun.xml.bind.v2.ContextFactory",
                f"-Djava.class.path={classpath_jars}",
                convertStrings=True
            )

        ## Init various CBSW java classes
        self.JPDMDBDepp     = jpype.JClass("cbsw.lhcb.pdmdb.JPDMDBDepp")
        self.JniAdeptDepp   = jpype.JClass("cbsw.adept.JniAdeptDepp")
        self.LpGbt          = jpype.JClass("cbsw.lhcb.pdmdb.LpGbt")
        self.MuDaq          = jpype.JClass("cbsw.lhcb.pdmdb.MuDaq")
```

Calling Java classes in python.

```
BOOST_PYTHON_MODULE(gecco_core)
{
    namespace bp = boost::python;

    bp::class_<GECCO_Python, boost::noncopyable>("GECCO_Python", bp::init<>())
        // Execution Controls (Threaded Readout Loop)
        .def("StartRun", &GECCO_Python::StartRun)
        .def("StopRun", &GECCO_Python::StopRun)
        .def("IsRunning", &GECCO_Python::IsRunning)

        // Device Connection Lifecycle Hooks
        .def("ConnectDevice", &GECCO_Python::ConnectDevice)
        .def("CloseDevice", &GECCO_Python::CloseDevice)

        // Hardware Controls & Configuration Mappings
        .def("UpdateASIC",         &GECCO_Python::UpdateASIC)
        .def("LoadConfigFile",     &GECCO_Python::LoadConfigFile)
        .def("ResetFastROFifo",    &GECCO_Python::ResetFastROFifo)
        .def("FastRO_Clk_toggle", &GECCO_Python::FastRO_Clk_toggle)
        .def("AutoPhaseAlign",     &GECCO_Python::AutoPhaseAlign)
        .def("ReadWord",           &GECCO_Python::ReadWord)
        .def("ToggleDecode",       &GECCO_Python::ToggleDecode)
        .def("WriteTDACs",         &GECCO_Python::WriteTDACs)

        // Data Utility Helpers
        .def("WriteToFile", &GECCO_Python::WriteToFile)
        .def("CharToHex", &GECCO_Python::CharToHex)

        // Expose critical internal string variables for live Python inspection
        .def_readwrite("fastro_lastread", &GECCO_Python::fastro_lastread)
    ;
}

#endif // GECCO_PYTHON_H
```

Python integration of C++ using boost.

Summary

- ECROC has been proven to work well performing scans across experimental conditions.
- Lots of systems have been added (HV, position, DAQ).
- Has been used for both MightyTracker and RICH work.

Next steps

- We would like to add **more systems**! *(Monochromator, rotation stages, oscilloscopes, etc).*
- We have ideas of how to use this to take measurements, but perhaps you do too?

Stay tuned for some nice results using ECROC!