pyAT, Pytac and pythonSoftIoc: A Pure Python Virtual Accelerator

W. Rogers, T. J. R. Nicholls, A. A. Wilson, Diamond Light Source
Harwell Science and Innovation Campus, Didcot, Oxon OX11 0DE

Motivation

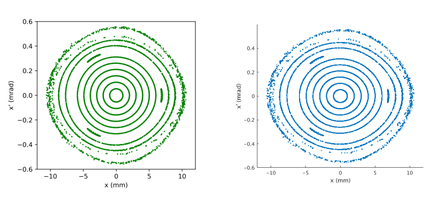
Testing control system software can be difficult because the hardware may be in use much of the time or, in some cases, may not yet exist. *Virtual Accelerators* are used to simulate a control system so that the software that will run against the control system can be realistically tested.

Python is a high-level programming language with many advantages for building a virtual accelerator: it is free and open source, is widely used in science and industry, has many third-party libraries available and is capable of building large scalable applications.

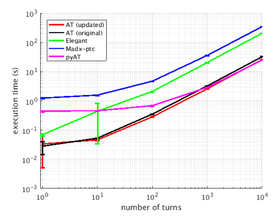
To build a virtual accelerator in Python a number of components are required, including a simulation code and a control system server that can be run from Python. This poster describes assembling those components into a fully-featured virtual accelerator.

pyAT

Accelerator Toolbox (AT) is a simulator for synchrotron light sources written in Matlab. It uses a numerical engine coded in C for efficiency. That design allowed recompiling the same C code into a Python extension so that the same engine could be used in Python, now available as pyAT.
Since the underlying code is the same, pyAT gives numerically identical results to AT, and it provides the simulation for our virtual accelerator.



Identical results obtained and plotted using the Python (left) and Matlab (right) builds of Accelerator Toolbox



Comparison of speeds of equivalent simulations of the ESRF EBS lattice using different codes

Pytac

Python Toolkit for Accelerator Controls (Pytac) is a Python library designed to enable working with different particle accelerator components. Each element in an accelerator is represented by an object that may give access to a number of physical parameters. Elements may belong to 'families', groupings that are often used in accelerator physics. Pytac allows requesting live data from the control system or (via ATIP) the equivalent data from a pyAT simulation.

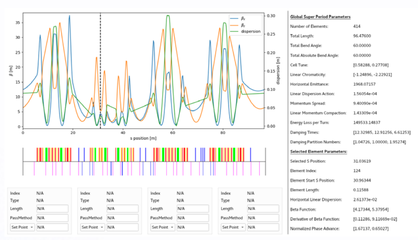
Unit Conversion

A control system typically works in engineering units, whereas simulation codes use physics units. Pytac provides a mechanism for converting between these two unit systems, with extensibility for different conversion methods.

```
>>> quad.get_value('b1', units=pytac.ENG)
103.18108367919922
>>> quad.get_value('b1', units=pytac.PHYS)
-1.0192934647760261
```

Volo

The different components described on this poster lend themselves to building other applications. Visualiser and Optimiser for Linear Optics (Volo) is now under development. It uses the pyAT simulator and the ATIP threading model to provide an interactive view of accelerator parameters. The next development is using SciPy numerical routines to provide an optimiser.



A screenshot of an early version of Volo

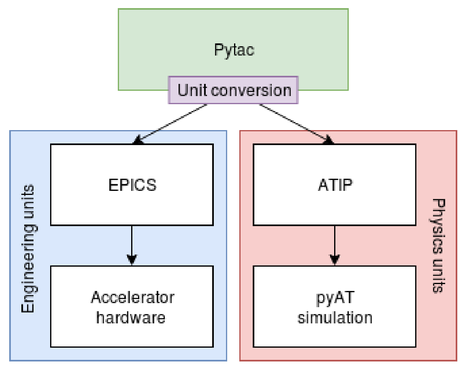
For more information please contact Will Rogers at will.rogers@diamond.ac.uk

ATIP

Accelerator Toolbox Interface for Pytac (ATIP) is the plugin that allows Pytac to use pyAT as its simulation.

```
>>> lattice.set_default_data_source(pytac.SIM)
>>> lattice.get_element_values('BPM', 'x')
[0.0, 0.0, 0.0, ...]
>>> h_corr = lattice.get_elements('HSTR')[0]
>>> h_corr.set_value('x_kick', 0.1, units=pytac.ENG)
>>> lattice.get_element_values('BPM', 'x')
[0.24630504031808942, 0.12495575893699563,
-0.1257213016476168, ...]
```

ATIP uses a simple asynchronous threading model to keep the simulation up to date. Any changes that would require a recalculation using pyAT are placed on a queue. A dedicated simulation thread loops continuously checking whether there are any items in the queue. If so, it empties the queue, applies the changes and recalculates. When a request for data is received, ATIP will check whether an update is pending and if so will wait until the recalculation is complete before returning that data.



How ATIP integrates into Pytac

Virtual accelerator

The virtual accelerator uses the components described on this page and pythonSoftIoc (see below) to construct the virtual accelerator.

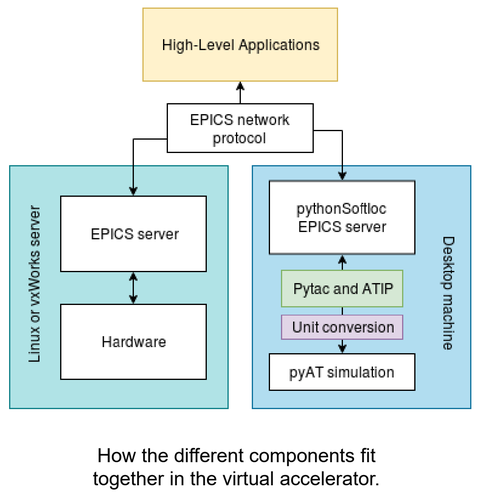
High-Level Applications

The following high-level applications can be tested using the virtual accelerator:

Slow orbit feedback uses the orbit response matrix and the beam position measurements to calculate corrections that are applied to corrector magnets.
RF feedback corrects any drifts in the RF frequency that are accumulated while orbit feedback is running.
Tune feedback uses some of the families of quadrupoles in Diamond's storage ring to keep the horizontal and vertical tunes stable.
Vertical emittance feedback uses Diamond's skew quadrupoles to maintain the vertical size of the electron beam.
Pyburt a new version of the EPICS Back Up and Restore Tool, under development and also using pure Python code.

Challenges

Control system complexities
Often the virtual accelerator presents a simpler view than the real control system. A number of techniques are used to mitigate this.
Simulation speed
The simulation can do a full recalculation in less than one second. In most cases this is adequate but in one case an update rate check needed disabling for a client application to run. It is also possible to calculate fewer parameters and update more quickly.



How the different components fit together in the virtual accelerator.

pythonSoftIoc
pythonSoftIoc allows creating EPICS servers using just Python code. It provides the control system server for the virtual accelerator.

Software

Python version 2.7 and 3.5+ are supported.
All code is open source and hosted on Github at https://github.com/dls-controls.
pyAT, Pytac and ATIP are available from PyPI: https://pypi.org.
Any interest in collaboration would be welcomed: email will.rogers@diamond.ac.uk.