Bluesky at BESSY II
A measurement script metamorphosis

Pierre Schnizer,
Luis Vera Ramirez, Thomas Birke, Tom Mertens, Markus Ries

Helmholtzzentrum Berlin
pierre.schnizer@helmholtz-berlin.de

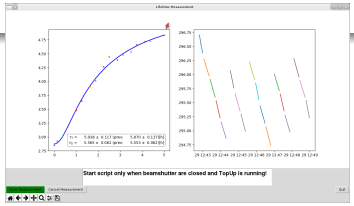
EPICS fall meeting 19^{th} – 23^{rd} September 2020

Contents

Introduction

Control system newbie

- Light optics, magnets, test station experience
- Joined BESSY II/VSR 2017
- First contact → synchrotron light source machine, BESSY II
- Control system newbie: → lifetime tool



Bluesky: Learning

- Jan 2019 ← info from Roland
- Start using FakeEpicsSignals
- Learned from:
  - Docs
  - gitter:
    - Listening
    - Questions (Thanks for the quick responses!)

BESSY II Control room environment

Environment

- Debian stretch
- Singularity container for python3.7
- (py)mongodb for data storage
- Lots of epics devices and tools

A big thanks to BESSY II operation / control system responsibles

Accelerator tools

Transverse optics

- Response matrix measurement
- Beam based alignment measurement
- Simple scripts
  - "synchronous" ramp down of sextupoles
  - Life time device

Encapsulated devices (Ophyd)

- Power supplies
- Beam position monitors,
- RF system
- Tune measurement, regulation
- Topup engine (beam current control)
- master clock

Dedicated plans

- per step plans for switching multiplexers, tune correction, topping up beam current

Plan stub example: beam based alignment

Beam based alignment

- each quadrupole $\rightarrow \pm\Delta I \rightarrow$ axis
- measurement:
  - multiplexer: $\rightarrow$ connects to each quadrupole
  - excites some current, up, down
  - BPM readings stored
  - analysis $\rightarrow$ ideal, golden orbit

BBA: Measurement implementation

Per step plan

```
import bluesky.plan_stubs as bps
import bluesky.plans as bp

def per_step(detectors, step, pos_cache):
    motors = step.keys()
    yield from bps.move_per_step(step, pos_cache)
    yield from bps.trigger_and_read(list(detectors) + list(motors))

    current = step.get(mux.power_converter, None)
    dIa = np.absolute(current)
    check_tune = False
    if dIa > 1e-3:
        # Power supplied powered ... no tune checking
        return

    # Readjusting tune
    yield from bps.mv(tune_fb.hor, tune_h, tune_fb.vert, tune_v)

    # Reinjecting current if required
    yield from bps.mv(topup, True)

# Outer product: spans "3D" space
loop_I = cycler(mux.power_converter, [0, dI, 0, -dI, 0])
quads = cycler(mux.selector, quadrupole_names)
repeat = cycler(CounterSink, range(n_meas))

RE(bp.scan_nd, det, loop_I * quads * repeat, per_step=per_step)
```

Plan stub example: beam based alignment

Beam **b**ased **a**lignment

BBA: measurement implementation

- ophyd devices: Multiplexer, piggy pack power converter: $\rightarrow$ devices in a tree
- measurement: bluesky plan to loop over quadrupoles, currents
- data storage: data broker $\rightarrow$ monogodb, uid in run book

Per step plan

```python
import bluesky.plan_stubs as bps
import bluesky.plans as bp

def per_step(detectors, step, pos_cache):
    motors = step.keys()
    yield from bps.move_per_step(step, pos_cache)
    yield from bps.trigger_and_read(list(detectors) + list(motors))

    current = step.get(mux.power_converter, None)
    dIa = np.absolute(current)
    check_tune = False
    if dIa > 1e-3:
        # Power supplied powered ... no tune checking
        return

    # Readjusting tune
    yield from bps.mv(tune_fb.hor, tune_h, tune_fb.vert, tune_v)

    # Reinjecting current if required
    yield from bps.mv(topup, True)

# Outer product: spans "3D" space
loop_I = cycler(mux.power_converter, [0, dI, 0, -dI, 0])
quads = cycler(mux.selector, quadrupole_names)
repeat = cycler(CounterSink, range(n_meas))

RE(bp.scan_nd, det, loop_I * quads * repeat, per_step=per_step)
```

Plan stub example: beam based alignment

Beam **b**ased **a**lignment

BBA: Measurement implementation

Per step plan

- called at each step
- thus possible:
- adjust tune at 0 current $\rightarrow$ mitigate hysteresis effect
  - $\rightarrow$ no beam loss during first run
  - executed at full current

```
import bluesky.plan_stubs as bps
import bluesky.plans as bp

def per_step(detectors, step, pos_cache):
    motors = step.keys()
    yield from bps.move_per_step(step, pos_cache)
    yield from bps.trigger_and_read(list(detectors) + list(motors))

    current = step.get(mux.power_converter, None)
    dIa = np.absolute(current)
    check_tune = False
    if dIa > 1e-3:
        # Power supplied powered ... no tune checking
        return

    # Readjusting tune
    yield from bps.mv(tune_fb.hor, tune_h, tune_fb.vert, tune_v)

    # Reinjecting current if required
    yield from bps.mv(topup, True)

# Outer product: spans "3D" space
loop_I = cycler(mux.power_converter, [0, dI, 0, -dI, 0])
quads = cycler(mux.selector, quadrupole_names)
repeat = cycler(CounterSink, range(n_meas))

RE(bp.scan_nd, det, loop_I * quads * repeat, per_step=per_step)
```

Callback based solvers

Numerical minimisers

- call back orientied implementation
- "full control"

Bluesky plans

- generator based
- yielding messages → executed by RunEngine
- result available afterwards

Possible solution

Solver and RunEngine in different threads, bridged by a bcib bridge

Software status: alpha

Example

- Callback:

```
def _stub(detectors, motor, val):
      yield from bps.mv(motor, val)
      r = (yield from
           bps.trigger_and_read(detectors + [motor])
      return r
def cb(val):
      # Typically some move command required
      cmd = partial(_stub, dets,  mot, val)
      r_dic = bridge.submit(cmd)
      # Extract return values from dictionary
      return r
```

- plan stub

```
yield from bcib.bridge_plan_stub(bridge)
```

- bridge creation:

```
bridge = setup_threaded_callback_iterator_bridge()
```

Naus: an OpenAI compatible environment

Implementation

- Actuators for state and actions
- Typically: Bluesky returns more data than required
- Dedicated methods for selecting required data

Design

- Bluesky plans:
  - Stepping
  - Resetting environment
- Dedicated methods: data extraction
  - state
  - reward

API

storeInitialState default: executes plan to retrieve current state of motors, extract "state at start of epoch"

getStateToResetTo returns state of "start" of epoch

computeState : compute state from detector readings

computeRewardTerminal compute reward, epoch finished

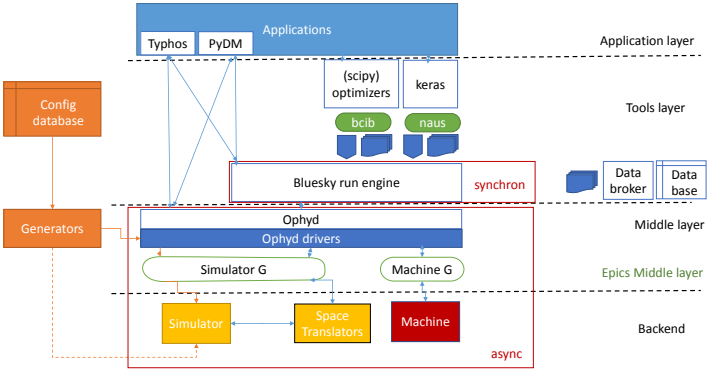
First use

See Luis talk!

Naus: Open points

- Gain experience with API
- implement callbacks: Keras compatible $\rightarrow$ available for bluesky liveplots
- Gain operation experiences
- Decrease latencies:
  - 50 ms reached up to now in "real world" reinforcement learning application
  - Latency estimates (yappi; "time.time", debian stable, vanilla kernel, intel i7 3.4 GHz cpu):
    - RunEngine: 1 ms
    - Naus, logging disabled: 5 ms
    - Ømq communication: 0.2 ms
- Next steps: latency reduction, EPICS as communication layer

Digital twin development



Similar implementations at other labs (e.g. ESRF, ALS-(U)),
Implementation started at BESSY II $\rightarrow$ tool for BESSY III
Interest on collaborations

Lessons learned

Learning curve

- Device programmers
  - python experience: object oriented programming
  - event / call back patterns
  - personal recommendation: implement stop method (first)
- Plan programmers
  - python experience: generators
- Users
  - python experience: coding scripts

A little obstacle

- "epics.PV": for simple measurements → follow "manual work flow"
- Bluesky → "event" based → requires adaption

"Selling" Arguments

- Document structure → automatic storage
- Replay → life plot development
- Device drivers → make available
- Pay back:
  - Complex devices → details abstracted by standard interface
  - Large number of variables → device tree
  - Sophistcated plan stubs → reuse in different scripts

Conclusion

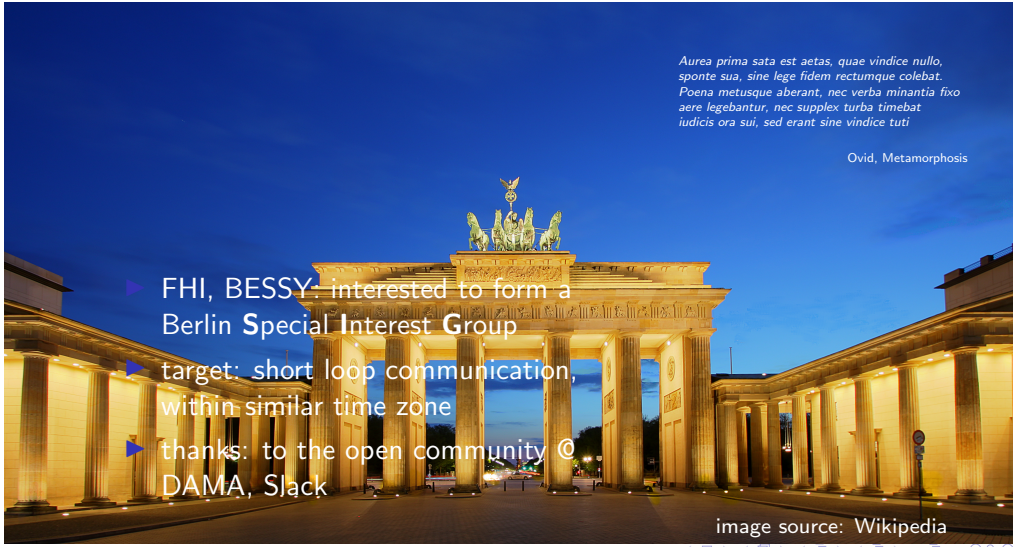
- Bluesky / Ophyd
  - Higher level of abstraction
  - Simplify measurement scripts
- BESSY II: Bluesky firm place for machine commissioning
  - Building device library
  - Measurement scripts
- Looking into:
  - Integrating solvers
  - Machine learning (see Luis talk)
  - Digital twin development

Thanks to

- bluesky / ophyd team for support, direct question answering
- All for introduction into machine, EPICS,. . . , resetting the machine, handling alarms
- All keeping BESSY II running and serviced!

Outlook

Outlook



- FHI, BESSY: interested to form a Berlin **S**pecial **I**nterest **G**roup
- target: short loop communication, within similar time zone
- thanks: to the open community @ DAMA, Slack

image source: Wikipedia