Fly scans at HEPS and BSRF

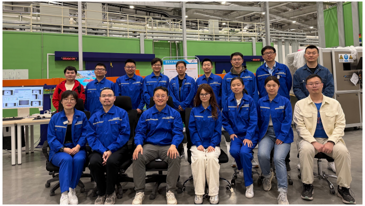
Ai-Yu Zhou
Xiao-Bao Deng
Zong-Yang Yue } Beamline control
Yu Liu
Peng-Cheng Li } Experiment software

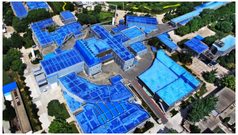
Technical discussions between HEPS and MAX-IV

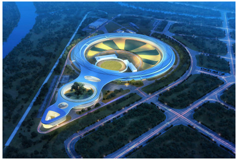
2025.08

Introduction: background information

- Beamline control (left image): EPICS-based control of individual devices.
- Experiment software (middle image): Bluesky-based interlocked action between devices.
- Before deployment at HEPS, a lot of our work has been tested at BSRF (upper right).





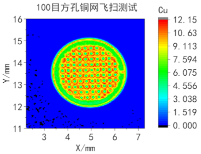


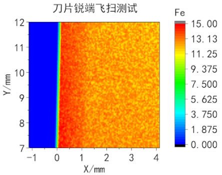
Introduction: a brief timeline of fly scans at HEPS and BSRF

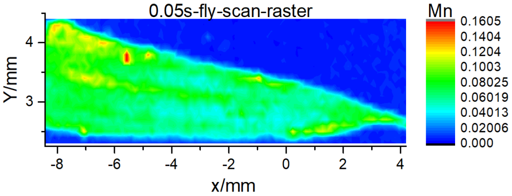
- 2019: first PandABox purchased by beamline control for early research, because of the need to speed up XRF scans at 4W1B of BSRF.
- 2020–2021: after investigation on candidates, comprehensive research on Bluesky was conducted; experiment software group formed, development of Mamba began.
- 2022–2023: our own framework for PandABox-Bluesky fly scans was developed and applied in a few simple cases at BSRF.
- 2024: composite scans and PandABox-based stationary acquisition schemes at HEPS.
- From 2024.09: high-speed fly scans based on trajectory moving, already producing interesting intermediary results now.
- On our horizon: undulator-monochromator fly scans, adaptive fly scans, fly scans that can be paused and resumed, ...

Architecture of fly scans: our early issues with pymalcolm

- To implement fly scans with Mamba (`DOI:10.1107/S1600577522002697`, `codeberg:CasperVector/mamba-ose`), our Bluesky-based software ecosystem, we researched pymalcolm (results shown in the images), PandABox's official middleware.
- We attempted to reuse pymalcolm's code, but found it over-complex. By understanding the role of PandABox, we would be able to identify what we needed to extract from pymalcolm.

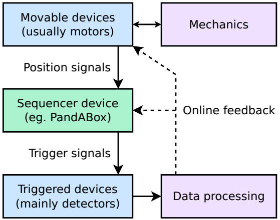


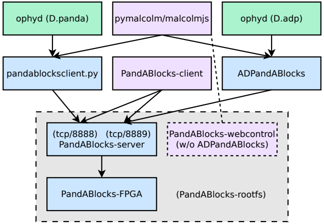




Architecture of fly scans: "P(osition) & A(cquisition)"

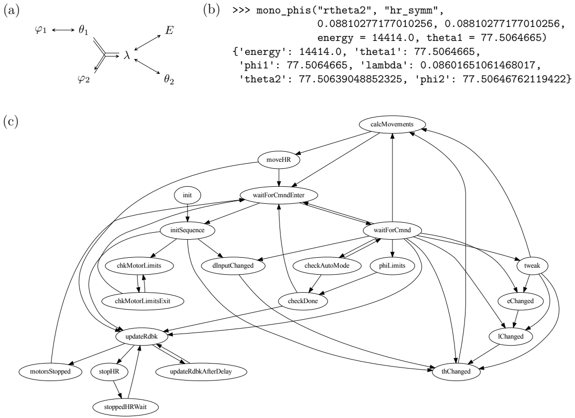
- Based on prior experience, we came up with the "P & A" architecture, inspired by the full name of PandABox, "position and acquisition control system". Mechanical factors also affect motion control. Work on detectors and data processing can be offloaded to other people. Also present is online feedback, a basis for adaptive fly scans.
- We extracted `pandablocksclient.py` from pymalcolm; an ophyd encapsulation of it is used to control PandABox, while ADPandABlocks is used to do the data readout. In our preliminary research for high-speed fly scans, we found ADPandABlocks to have a framerate upper-bound of a few kHz. This was a major motivation for our caproto-based Python IOC framework QueueIOC (`arXiv:2411.01258`, `arXiv:2411.01278`, `codeberg:CasperVector/queue_iocs`).





Architecture of fly scans: from ADPandABlocks to QueueIOC

- Our Python IOC `qdet_panda` is able to saturate the 45 MB/s bandwidth of PandABox's TCP server using its gigabit ethernet and the `XML FRAMED SCALED` format: a 980 kHz framerate when each frame is composed of 6 `double` fields.
- This is about the same performance as Diamond's PandABlocks-ioc, but our code is shorter and more reusable. QueueIOCs have also been written to support devices traditionally done with EPICS StreamDevice and seq; our monochromator IOCs greatly simplify complex state machines from their optics-based counterparts (lower image).



Fly scans at BSRF: time-based triggering with Bluesky

- With our ophyd encapsulation, we can fully control PandABox. Based on it is our first Bluesky plan for regular grid fly scans, `fly_grid()` (`DOI:10.1007/s41605-023-00416-x`). It and `fly_dgrid()` are usable with a wide range of motors with encoders.
- Using time-based triggering, `fly_grid()` is also a simplest fly-scan plan to implement, only needing loops of at most 4 sequencer instructions. Time-based triggering produce uneven scan points with unstable motor speeds, but Voronoi diagrams can be considered.

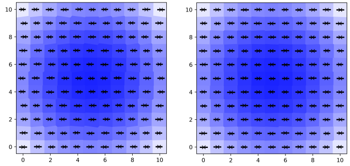
D.panda call:
```
> D.panda.seq1.table.max_length.get()
< 16384
```
*Underlying **pandablocksclient.py** call:*
```
> client.get_field("SEQ1", "TABLE.MAX_LENGTH")
< "16384"
```
PandABlocks-server's on-wire communication:
```
> SEQ1.TABLE.MAX_LENGTH?
< OK =16384
```

D.panda call:
```
> D.panda.ttlout10.val.put("TTLIN1.VAL")
< None
```
(Misuse raises exceptions.)

*Underlying **pandablocksclient.py** call:*
```
> client.set_field("TTLOUT10", "VAL", "TTLIN1.VAL")
< None
```
(Misuse raises exceptions.)

PandABlocks-server's on-wire communication:
```
> TTLOUT10.VAL=TTLIN1.VAL
< OK
```
(Misuse results in other replies.)

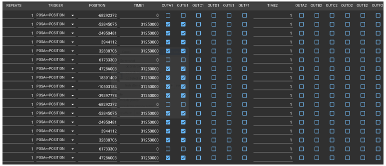
REPEATS	TRIGGER	POSITION	TIME1	OUTA1	OUTB1	OUTC1	OUTD1	OUTE1	OUTF1	TIME2	OUTA2	OUTB2	OUTC2	OUTD2	OUTE2	OUTF2
1	POSA<=POSITION	-68292372	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
4	POSA>=POSITION	-53845075	31250000	☑	☑	☐	☐	☐	☐	31250000	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	61733300	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
4	POSA<=POSITION	47286003	31250000	☑	☑	☐	☐	☐	☐	31250000	☐	☐	☐	☐	☐	☐



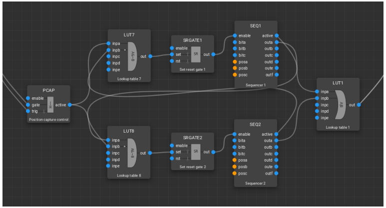
(a) (b)

Fly scans at BSRF: double SEQ and position-based triggering

- Position-based triggering may need many sequencer instructions, but one PandABox SEQ block can only contain 4096 instructions. We implemented automatic switching between the 2 SEQ blocks in a PandABox, now allowing a 16–20 kHz instruction rate. After one block runs out of instructions, new instructions are filled in while the other block starts running.
- Based on this, we implemented the plan `fly_dgrid()` that supports position-based triggering. Our PMAC-based plans are also based on this double-SEQ scheme, as will be our planned adaptive fly scans, because it can support infinite streams with online feedback.



REPEATS	TRIGGER	POSITION	TIME1	OUTA1	OUTB1	OUTC1	OUTD1	OUTE1	OUTF1	TIME2	OUTA2	OUTB2	OUTC2	OUTD2	OUTE2	OUTF2
1	POSA<=POSITION	-68292372	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	-53845075	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	-24950481	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	3944112	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	32838706	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	61733300	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	47286003	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	18391409	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	-10503184	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	-39397778	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	-68292372	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	-53845075	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	-24950481	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	3944112	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	32838706	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA>=POSITION	61733300	0	☐	☐	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐
1	POSA<=POSITION	47286003	31250000	☑	☑	☐	☐	☐	☐	1	☐	☐	☐	☐	☐	☐



Fly scans at BSRF: scan fragmentation, MambaPlanner and Bubo

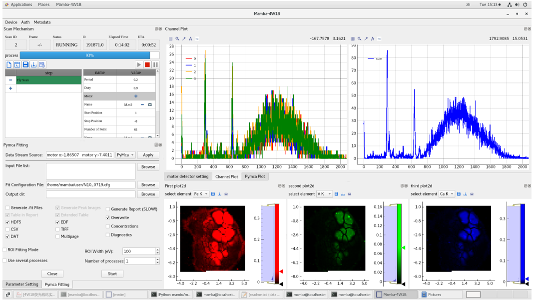
- Devices like Xspress3 have limits on the number of frames that can be acquired one time, and our fly-scan plans can split scans into fragments within the limits (set by the parameter `div`).
- Parameters like `div` make the raw command for a fly scan quite verbose. So MambaPlanner was developed to reduce duplication, abstract details and do checks. Also developed was Bubo, a mechanism for software-based fly-scans (`DOI:10.1080/08940886.2023.2277639`).

```
U.planner = ImagePlanner(U)
U.planner.extend(PandaPlanner(
    [D.panda], divs = {D.xsp3: 12216}, h5_tols = {D.xsp3: 0},
    enc_tols = {m: 25 for m in M.values()},
    vbas_ratios = {m: 2.0 for m in M.values()},
    configs = {D.xsp3: {"cam.trigger_mode": 3}}
))
U.planner.extend(BuboPlanner(D.bubo,
    divs = {D.xsp3: 12216}, h5_tols = {D.xsp3: 0}))
P = U.planner.make_plans()

#RE(fly_dgrid([D.xsp3], M.m2, -1, 1, 3, M.m1, -4, 4, 5, duty = 0.5,
#    period = 0.5, div = 12216), cb_gen(...), md = U.mdg.read_advance())
P.fly_dgrid([D.xsp3], M.m2, -1, 1, 3, M.m1, -4, 4, 5, duty = 0.5, period = 0.5)
P.sfly_grid([D.xsp3], M.m2, -1, 1, 3, M.m1, -4, 4, 5, pad = 2)
```

Fly scans at HEPS: composite fly scans based on `fly_dgrid()`

- Based on the backend mechanisms above, we developed Mamba GUIs for simple fly scans at BSRF (4W1B example shown in the image), and produced satisfactory results. Based on all of these, we implemented plans for composite scans.
- Examples include 1D, 2D energy and 3D mosaic tomography scans at imaging beamlines of HEPS; corresponding GUIs are also in the process of development or deployment. Other fly-scan requirements based on `fly_dgrid()` have also been implemented, eg. 3D XRD fly scans.

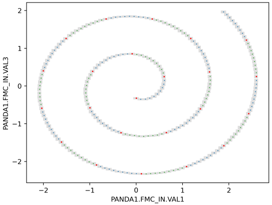


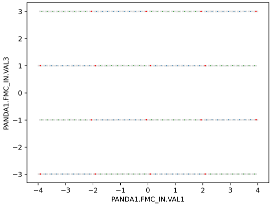
Fly scans at HEPS: position feedbacks and stationary acquisition

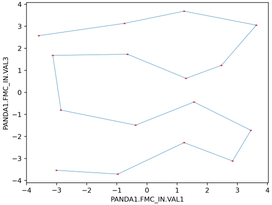
- Another major type of complication arises from the diversity of position feedbacks:
  - BISS encoders etc require more delicate configuration than quadrature encoders.
  - Absolute encoders and ADC feedbacks have no hardware zero points.
  - Laser interferometers & ADC feedbacks have scales & offsets different from the motor IOCs.
  - The Huber controller will lose connection to its BISS encoders when configured wirings between the corresponding "PandA blocks" are reset.
- To fully exploit its potential, we also use PandABox to do other tasks:
  - Beamlines like BB need multi-framed stationary acquisition from detectors, coupled with automatic hardware-timed opening and closing of shutters to minimise radiation damage.
  - The XPCS experiments at B4 are similar, just wrapping the above inside step scans.
  - Mamba GUIs have been developed for both above.

Advanced fly scans: scans based on trajectory moving of motors

- We are actively developing high-speed fly scans based on trajectory moving with eg. PMAC at B4 and B2. We have developed PMAC-based fly-scan plans for archimedean spirals, regular 2D grids, and a plan for pseudo-step scans. In pseudo-step scans, the motion and triggers are controlled by PMAC PVTs and PandABox sequencer tables, allowing for high-speed acquisition.
- The Bluesky/Mamba layer has passed early tests, but we are yet to test how the entire hardware-software system behaves in real high-speed conditions with large numbers of scan points.

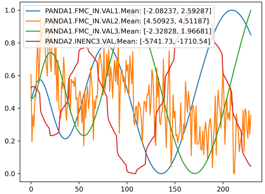


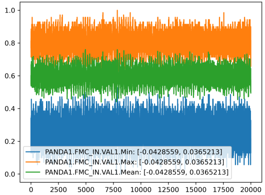




Advanced fly scans: closed-loop feedback and ADC noise issue

- Obvious distortion can be seen on the trajectories measured with laser interferometers; this is a most important issue we need to resolve.
- For now we rely on ADC feedbacks, but another issue is ADC noise: the peak-peak noise of one axis at B4 can be up to 140 nm, while the RMS noise is ~ 5 nm (both converted to the engineering unit).
- While results on the previous page shows the noise does not seem to severely affect our basic triggering logic, the B4 beamline wants better ADCs with no more than ~ 1 nm noise.
- We wish to be able to combine the software layer above with hardware and control bases comparable with APS' velociprobe (`DOI:10.1063/1.5103173`).





Advanced fly scans: subjects on our horizon

- Currently under research – undulator-monochromator fly scans: we are following eg. the HD-DCM at Sirius (`DOI:10.1107/S1600577522010724`).
- Currently under research – adaptive fly scans: we are following eg. the boundary-guided ptychography at APS (`DOI:10.1107/S1600577523009657`); Diamond's PMAC IOC (`github:DiamondLightSource/pmac`) lacks full support for ring buffers, and we plan to add this support in our refactored IOC (`codeberg:CasperVector/motorPmac`).
- Future subject – fly scans that can be paused and resumed: this may require deep changes inside Bluesky's RunEngine.
- In conclusion: plenty of fun is waiting for us!

Thanks!