python accelerator middle layer

pyAML

S.White (ESRF) on behalf of the pyAML Collaboration

MIDDLE LAYER

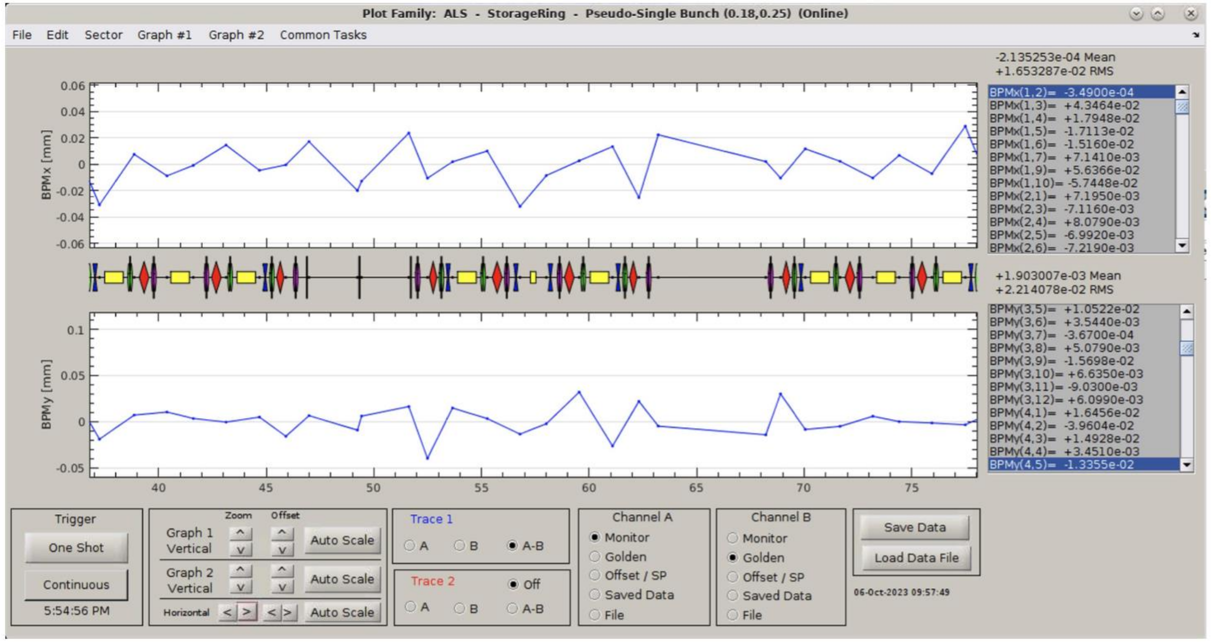
1. Software tools usable by any accelerator:
   1. Control agnostic
   2. Accelerator naming/layout agnostic

2. Easy to use, includes high level graphical application
3. “Easy” to set up
4. Tested in many labs: Robust and Reliable
5. No need of additional resources to set up standard tools
6. Easy scripting language
7. Integration of accelerator model (digital twin)
8. Gives access to not trivial to implement software, such as LOCO (linear optics from closed orbit)

https://indico.desy.de/event/43233/contributions/169040/attachments/90944/122729/OperationML_nadolski.pdf

https://indico.esrf.fr/event/123/contributions/623/attachments/377/750/2023%20-%20MML%20History%20-%20AT%20Workshop.pdf

1994, SLAC + ALS, Matlab



ML config

Facility Choice (SOLEIL, ALS, etc.) Accelerator Choice (Ring, Booster, transfer lines…)

Accelerator Model Config

Data Storage

Matlab Middle Layer

Matlab is a proprietary programming language

Collaborative development very difficult: last update in 2018, diverged in many private versions

Does not benefit from modern computing, not possible to interface with more recent developments

Does not implement scientific open data management

- MML will soon become obsolete
- Operation of many facilities at risk
- Users community looking for long term alternatives

has World wide users (incomplete list)

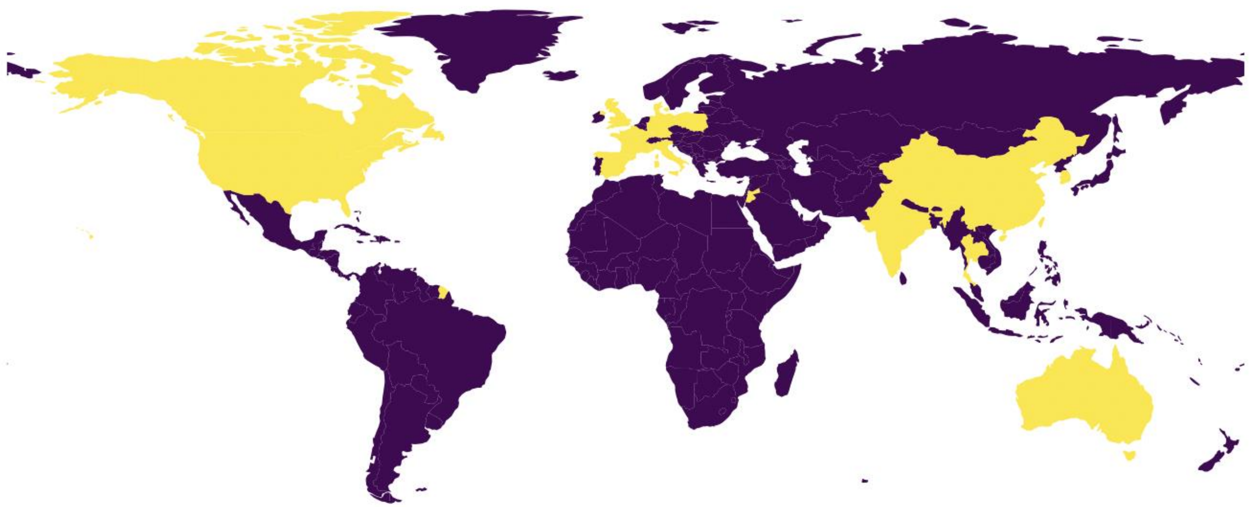
USA: ALS, Stanford (Spear3), Duke FEL, Brookhaven (VUV or X-Ray rings), B-Factory

Canada: CLS
Europe: SOLEIL (France), DIAMOND (England), ALBA (Spain), Solaris (Poland), MAX-IV (Sweden)

Asia: PLS2 (Korea), SLS (Thailand), SSRF (Shanghai), NSRRC (Taiwan)

Middle East: SESAME (Jordan)

Australia: ASP



```
>>> switch_to_hardware()
```

EPICS

Home made

Operation phase

Python Accelerator Middle Layer (pyAML)

The python digital twin would provide a long term solution in an open source license free environment:

- **Simplified collaborative development, integrate modern CI/CD approach**
- **Clean and simple installation procedure**
- **Easy to interface with others recent developments using modern techniques such as advance commissioning simulations (pySC) or AI/ML (Badger/Xopt)**
- **Clear automatically generated documentation**
- **Works for any accelerator (ring, linac, transfer lines) and control systems**

Python is among the most used software language in the world. Free and open-source. Very large users/developers community Huge number of scientific libraries.

Python Accelerator Toolbox simulations

Development phase

```
>>> switch_to_physics()
```

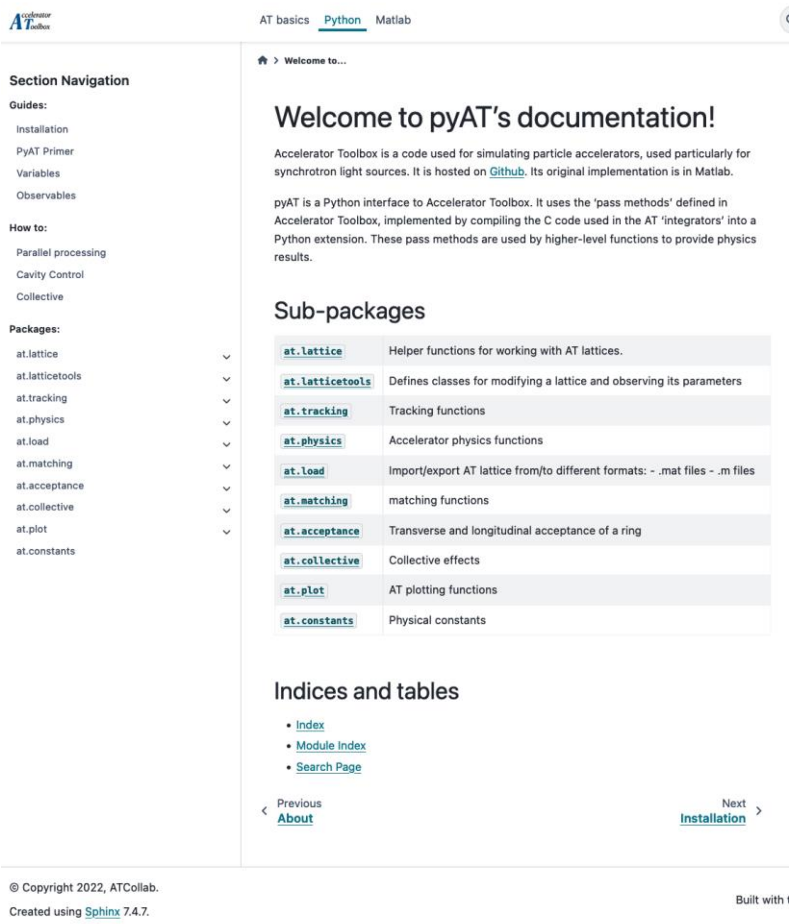
WHERE DO WE START?

A solid and benchmarked python Accelerator Toolbox (pyAT) package for storage ring simulations, with active users and developers

Very wide user experience from MML and (py)AT within a well established collaborative environment

Very strong experiment in development on lattice correction and optimization algorithms:

- Well established and tested methods: ESRF-EBS commissioning
- Python Simulated Commissioning (outcome of EURIZON ~ 1year of collaboration ESRF-DESY)
- Python virtual accelerators ESRF-EBS, Diamond (UK), Sirius (Brasil),...
- Fully benchmarked python optimization algorithms and software (Badger, Xopt developed at SLAC)



AT basics Python Matlab

Welcome to...

Section Navigation

Guides:
- Installation
- PyAT Primer
- Variables
- Observables

How to:
- Parallel processing
- Cavity Control
- Collective

Packages:
- at.lattice
- at.latticetools
- at.tracking
- at.physics
- at.load
- at.matching
- at.acceptance
- at.collective
- at.plot
- at.constants

Welcome to pyAT's documentation!

Accelerator Toolbox is a code used for simulating particle accelerators, used particularly for synchrotron light sources. It is hosted on Github. Its original implementation is in Matlab.

pyAT is a Python interface to Accelerator Toolbox. It uses the 'pass methods' defined in Accelerator Toolbox, implemented by compiling the C code used in the AT 'integrators' into a Python extension. These pass methods are used by higher-level functions to provide physics results.

Sub-packages

at.lattice	Helper functions for working with AT lattices.
at.latticetools	Defines classes for modifying a lattice and observing its parameters
at.tracking	Tracking functions
at.physics	Accelerator physics functions
at.load	Import/export AT lattice from/to different formats: - .mat files - .m files
at.matching	matching functions
at.acceptance	Transverse and longitudinal acceptance of a ring
at.collective	Collective effects
at.plot	AT plotting functions
at.constants	Physical constants

Indices and tables

- Index
- Module Index
- Search Page

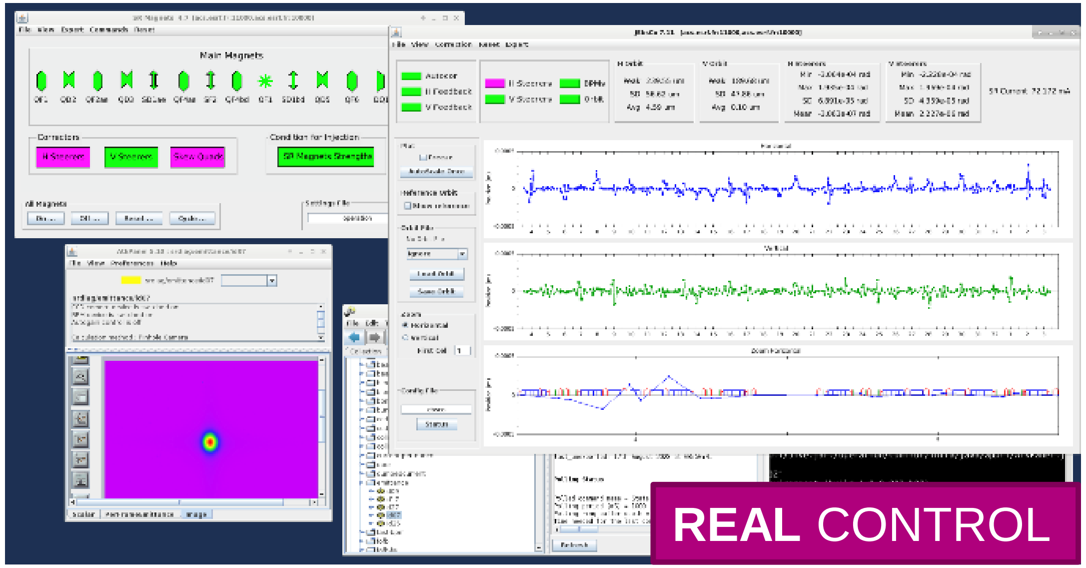
Previous About — Next Installation

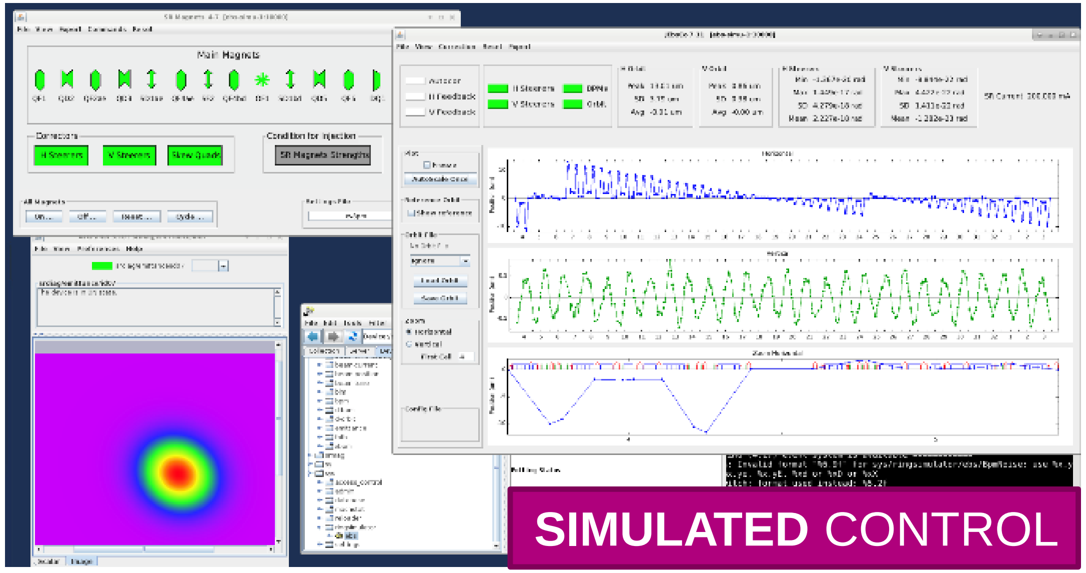
© Copyright 2022, ATCollab.
Created using Sphinx 7.4.7.

Standalone laboratory specific applications.

Building a joint technology platform to merge and integrate all these developments and coordinating the effort in a collaborative manner would bring strong benefits for the whole community

EXAMPLE: ESRF-EBS VIRTUAL ACCELERATOR





acs.esrf.fr:10000 real control system	**ebs-simu:10000** simulated control system
Magnets:	**Magnets**:
PS (currents)	PS (currents) no hardware
Magnet control (strengths)	Magnet control (strengths)
Calibration DS	Calibration DS
Vectorized DS	Vectorized DS
Hot Swap management	Hot Swap management
Cycling devices	Cycling devices
RF:	**RF**:
Master source device	Master source device
Diagnostics:	**Diagnostics**:
Beam Position Monitor	Beam Position Monitor
(Slow and Turn-by-turn)	(Slow and Turn-by-turn)
Tune Monitor	Tune Monitor
Emittance monitors	Emittance monitors
Beam current and lifetime	Beam current and lifetime
Physics:	**Physics**:
Electron beam	PyRingSimulator
High-level applications:	**High-level applications**:
Magnets	Magnets
Orbit, Tune, Optics	Orbit, Tune, Optics
Chromaticity	Chromaticity
etc..	etc..

= (magnets: real = simulated)

simulated (RF and diagnostics)

update

> Events on change >

= (high-level applications: real = simulated)

PYTHON ACCELERATOR MIDDLE LAYER

Will provide as MML:

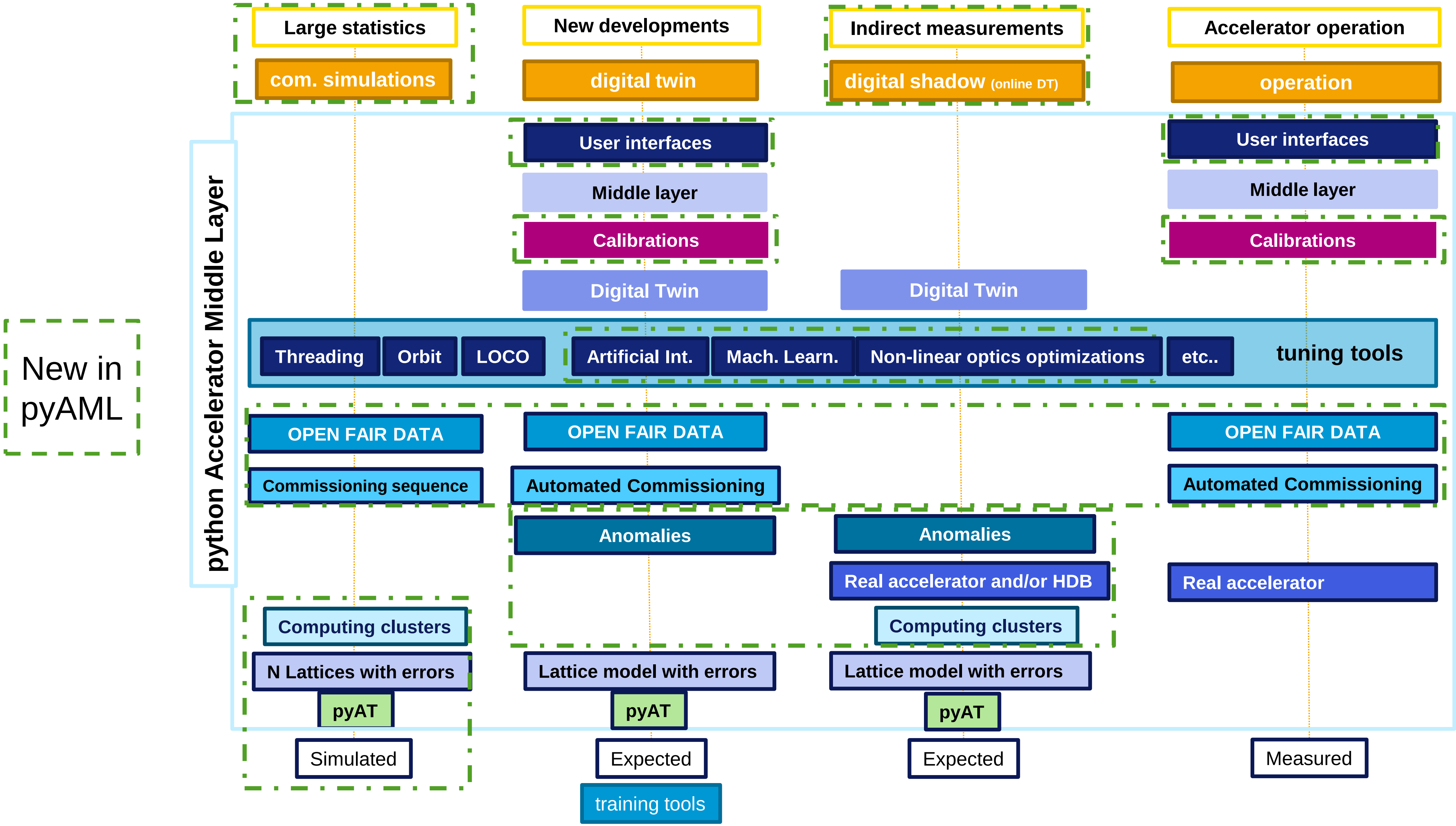
- abstracted interaction with different control systems
- abstracted naming conventions for all devices
- Work for transfer-lines, linear and circular accelerators, ramped accelerators
- Digital twin
- Tools for orbit, trajectory, linear and non-linear optics corrections

See in the next slide a schematic view of the **pyAML** joint technology platform

Will provide ADDITIONALLY:

- Fully open source
- High Performance Computing (HPC) clusters for commissioning simulations
- GPUs and analytic Jacobians)
- Enable easy and user-friendly connection to any control system including commands, properties and timing features
- OPENDATA and FAIR data
- data labelling for machine learning and artificial intelligence algorithms
- off-line digital twin for tuning tools development
- on-line digital shadow for monitoring of indirect observables and anomaly detection
- Python based Artificial Intelligence and Machine Learning tools (e.g.: pytorch , Badger/Xopt)
- Enforce thorough documentation
- Provide a virtual accelerator environment to train students or newcomers to the field

WHAT WE WANT TO OBTAIN: PYAML JOINT TECHNOLOGY PLATFORM

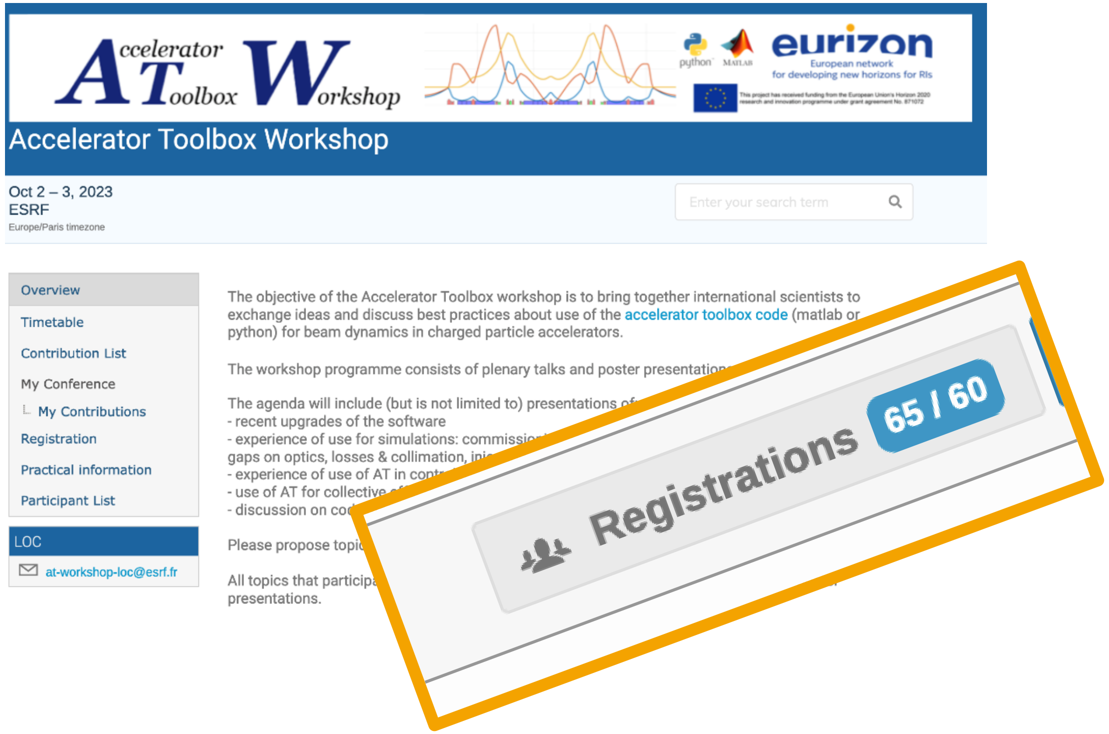


CONSTRUCTION AND CONSOLIDATION OF THE AT COLLABORATION

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072

eurizon
European network
for developing new horizons for RIs

25 world-wide laboratories



Extremely positive, active and fruitful collaboration within EU project. **TO BE CONTINUED and EXTENDED.**

Expression of interest from many institutes, contact person identified:

ESRF: Simone Liuzzo, S.White, L.Farvacque
DESY: Ilya Agapov, Lukas Malina, Yong-Chul Chae
SOLEIL: Laurent Nadolski, Alexis Gamelin
ALBA: Zeus Marti, Gabriele Benedetti
MAXIV: Marco Apollonio, Magnus Sjostrom
HZB: Teresia Olsson, Pierre Schnizer
IJClab-ThomX: Vyacheslav Kubytskyi
DIAMOND: Hung-Chun Chao, Tobyn Nicolls, Martin Gaughran, Richard Fielder
ELETTRA: Stefano Krecic
SOLARIS: Jacek Biernat
SESAME (Jordan): Samira kasaei
ALS LBNL (USA): Thorsten Hellert
IHEP (China): Daheng Ji, Mengyu Su
ANSTO (Australia): Paul Bennetto
NSLS-2 (USA): Xi Yang
PSI (Switzerland): Jonas Kallenstrup
CLS (Canada): Michael Bree
Korea 4GSR (Korea): Chong Shin Park

Signed project summary shared with LEAPS RDB

Contributions from many labs.

Discussion about
- AGILE developments
- Definition of specifications
- Definition of use-cases
- New software architectures
- Fully python based facilities (such as Sirius in Brasil)
- EU grant possibilities

Outcome:
- Submit proposal for COST action (DONE)
- Proceed with exploratory tests (DONE, continues)
- Define specifications/use-cases documents (to start in November)

Next workshop in HZB (Berlin) February-March 2025

FUNDS needed. COULD LEAPS SUPPORT US?

COST: COOPERATION FOR SCIENCE AND TECHNOLOGY

~ 150kEuro / year for 4 years for "networking"

Must involve 50% ITC countries

All country/lab gets payed trips to conferences, trainings, exchanges, summer schools, coding workshops.

Non-EU countries (USA, Brazil, etc.) collaborators can be paid as expert giving a presentation or a training.

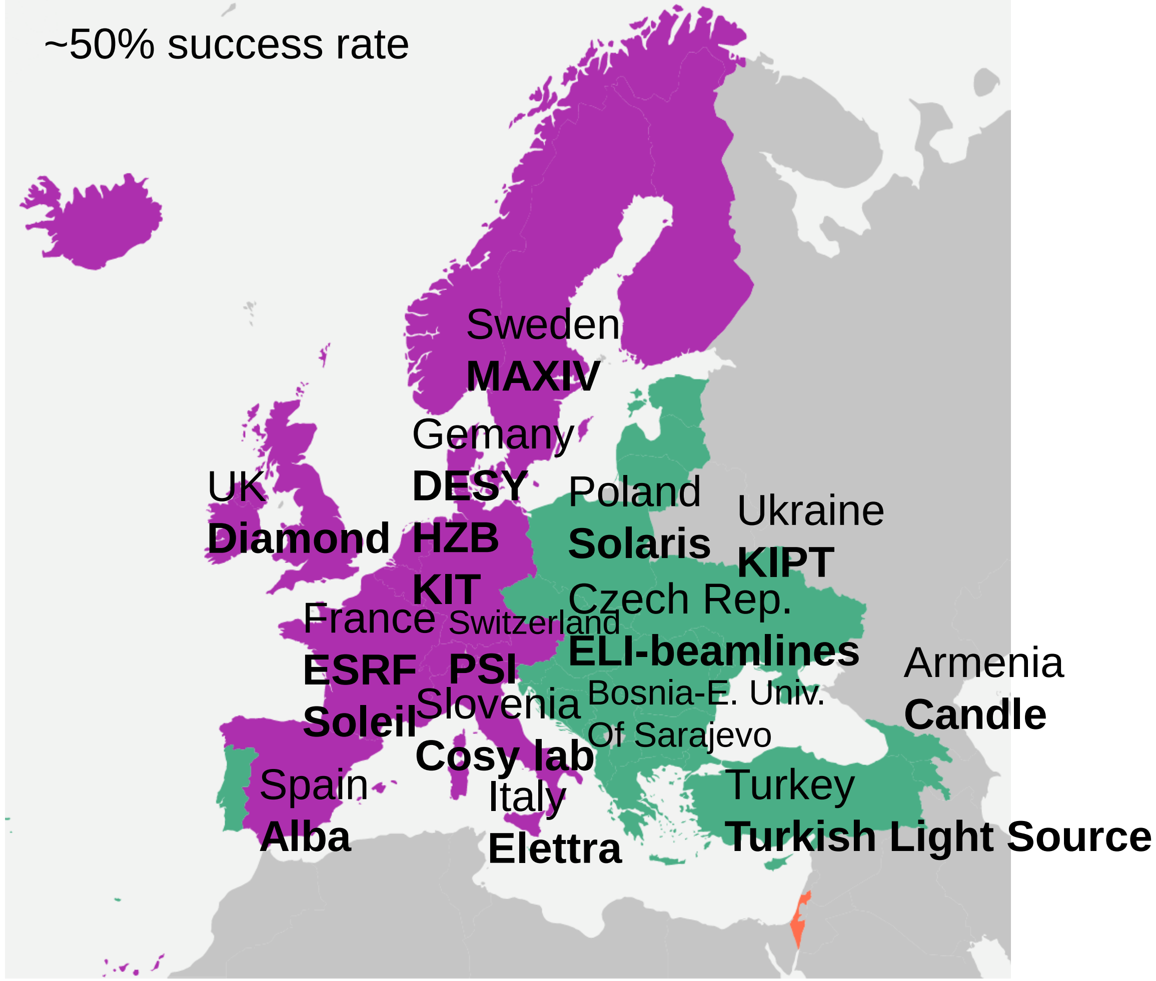
pyAML-Net COST Action Proposal
submitted

Reply expected June 2025
If approved, start activity in October 2025

Once approved anyone will be able to join the network!

Green == ICT (inclusiveness target country)

~50% success rate



www.cost.eu

ONGOING EXPLORATORY WORK: TEST AND EVALUATE EXISTING SOLUTIONS

pyTAC : Diamond light source, UK, in use, EPICS based

bluesky : NSLS-II, US, in use by many user beamlines world wide, EPICS based

pyacal : SIRIUS, Brazil, in use (adapted from existing tools to be used by other labs), EPICS based

→ Run an orbit response matrix measurement

The above tools have been tested at ESRF (TANGO), HZB (EPICS), MAXIV (TANGO)

Easy to set up?

Handles calibrations?

How is data stored/managed?

User friendly?

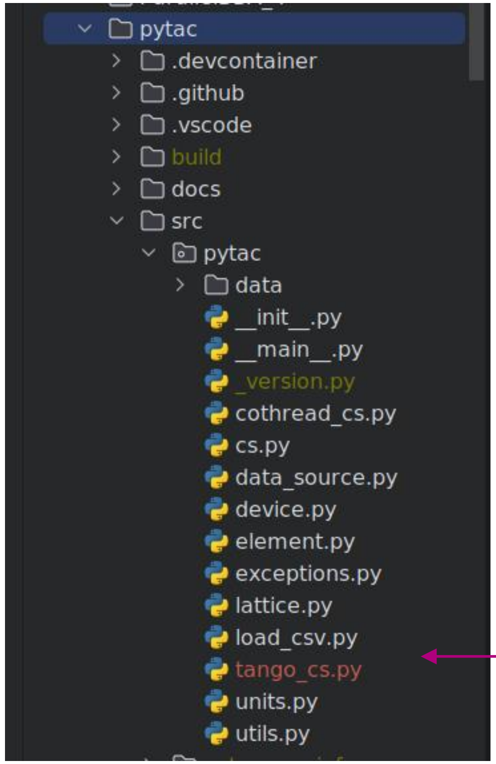
Graphical interfaces are available?

What measurements can be done?

Digital twin/shadow?

PYTAC TANGO INTERFACE: 1 FILE

Jean-Luc Pons, ESRF




```python
from pytac.load_csv import load
from pytac.tango_cs import TangoControlSystem
```

```python
lattice = load('EBS-vec',
               directory=Path('/machfs/liuzzo/EBS/beamdyn/MDT/2024/2024_06_01/pytac/src/pytac/data'),
               control_system=TangoControlSystem(wait=True, timeout=2.0))
```

pyTAC already ready for other CS.

set_single and read_single methods are the only control system dependent methods.

CODE COMPARISON (IMPORTS, MEASUREMENT, DATA, PLOT)

pyTAC ORM

```python
import os
import time

from pytac.load_csv import load
from pytac.tango_cs import TangoControlSystem
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import pickle

# os.environ['TANGO_HOST'] = 'ebs-simu-1:10000'

lattice = load('EBS-vec',
            directory=Path('/machfs/liuzzo/EBS/beamdyn/MDT/2024/2024_06_01/pytac/src/pytac/data'),
            control_system=TangoControlSystem(wait=True, timeout=2.0))

dk = 1e-5
df = 100

horStrs = lattice.get_elements("SF2A")
verStrs = lattice.get_elements("SF2A")
oh = lattice.get_value('orbit_h')
nbpm = len(oh)
NHst = len(horStrs)
NVst = len(verStrs)

orm = dict(HorSteererHorBPMs=np.empty((NHst, nbpm)),
         VerSteererHorBPMs=np.empty((NVst, nbpm)),
         HorSteererVerBPMs=np.empty((NHst, nbpm)),
         VerSteererVerBPMs=np.empty((NVst, nbpm)),
         RFHorBPM=np.empty((1, nbpm)),
         RFVerBPM=np.empty((1, nbpm))
         )

def get_orbit():
   return lattice.get_value('orbit_h'), lattice.get_value("orbit_v")

def get_resp(s, kick='y_kick', sleeptime = 3, dk=dk):
   n = s.get_pv_name(kick, 'readback')
   print(f'response for {n}')
   s0 = s.get_value(kick)
   s.set_value(kick, s0 + dk)
   time.sleep(sleeptime)
   (horb_stp, vorb_stp) = get_orbit()
   s.set_value(kick, s0 - dk)
   time.sleep(sleeptime)
   (horb_stm, vorb_stm) = get_orbit()
   s.set_value(kick, s0)
   time.sleep(sleeptime)
   hr = [(p - m) / (2*dk) for p, m in zip(horb_stp, horb_stm)]
   vr = [(p - m) / (2*dk) for p, m in zip(vorb_stp, vorb_stm)]
   return hr, vr


for i, h in enumerate(horStrs):
   hr, vr = get_resp(h, kick='x_kick')

   orm['HorSteererHorBPMs'][i, :] = hr
   orm['HorSteererVerBPMs'][i, :] = vr


for i, h in enumerate(verStrs):
   hr, vr = get_resp(h, kick='y_kick')

   orm['VerSteererHorBPMs'][i, :] = hr
   orm['VerSteererVerBPMs'][i, :] = vr


# # dispersion
freq = lattice.get_elements("RFCavity")
hr, vr = get_resp(freq[0], kick='f', dk=df)  # all cavities point to the same master source device

orm['RFHorBPM'][0, :] = hr
orm['RFVerBPM'][0, :] = vr


# store in file all information required for optics FIT
orm_for_fit = dict(
          H2H=orm['HorSteererHorBPMs'],
          V2H=orm['VerSteererHorBPMs'],
          H2V=orm['HorSteererVerBPMs'],
          V2V=orm['VerSteererVerBPMs'],
          dh=orm['RFHorBPM'],
          dv=orm['RFVerBPM'],
          tunes=[lattice.get_value('tune_x'), lattice.get_value('tune_y')],
          hor_steerers=[h.get_pv_name('x_kick', 'readback') for h in horStrs],
          ver_steerers=[v.get_pv_name('y_kick', 'readback') for v in verStrs],
          bpms=lattice.get_element_pv_names('BPM', 'y', 'readback'))

pickle.dump({'orm': orm}, open('ORMdata_pyTAC.pkl', 'wb'))

# Set up grid and test data
S=orm['HorSteererHorBPMs'].shape

nx, ny = S[0], S[1]
x = range(nx)
y = range(ny)
X, Y = np.meshgrid(x, y)  # `plot_surface` expects `x` and `y` data to be 2D

plt.style.use('_mpl-gallery')

fig, ax = plt.subplots(nrows=2, ncols=2, subplot_kw={"projection": "3d"}, figsize=(10, 10))
ax[0, 0].plot_surface(X, Y, orm['HorSteererHorBPMs'].T)
ax[0, 0].set_ylabel('Hor BPM')

ax[0, 1].plot_surface(X, Y, orm['VerSteererHorBPMs'].T)

ax[1, 0].plot_surface(X, Y, orm['HorSteererVerBPMs'].T)
ax[1, 0].set_xlabel('Hor steerers')
ax[1, 0].set_ylabel('Ver BPM')

ax[1, 1].plot_surface(X, Y, orm['VerSteererVerBPMs'].T)
ax[1, 1].set_xlabel('Ver steerers')

plt.savefig('ORM_pyTAC.png')
plt.show()
```

Bluesky + pyTAC ORM

BPM ophyd

```python
"""

Todo:
   Review if ophyd tango should be used
"""
import time
from typing import List, Dict, Any
from ophyd.status import Status
from ophyd import BlueskyInterface
import numpy as np
from collections import OrderedDict
from ophyd.device import OrderedDictType

# temporary for safe-test purposes: force commands to simulator only.
import os
os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'

__author__ = 'Pierre Schnizer (HZB), Simone Liuzzo (ESRF)'
__data__ = 'March 2024'

class BPMs(BlueskyInterface):
   """
   Read BPM values

   Parameters
   ----------
   name : string

   Example
   -------

   """
   def stage(self, **kwargs):
      super().__init__(**kwargs)
      pass

   def unstage(self) -> List[object]:
      pass

   def describe(self):
      """Return an OrderedDict with exactly the same keys as the ``read``
      method, here mapped to per-scan metadata about each field.
      """
      return dict(positions={'source': 'bpms', 'dtype': 'array', 'shape': (320,)})

   def describe_configuration(self) -> OrderedDictType[str, Dict[str, Any]]:
      res = OrderedDict()
      res.update({'position': {'source': 'bpms', 'dtype': 'array', 'shape': (320,)}})
      return res

   def trigger(self):
      return Status(success=True, done=True)

   def read(self):
      """Return an OrderedDict mapping string field name(s) to dictionaries
      of values and timestamps and optional per-point metadata.
      """
      vals = np.vstack([
         self.lattice.get_value('orbit_h'),
         self.lattice.get_value('orbit_v')
      ])
      return dict(positions=dict(value=vals, timestamp=time.time()))

   def read_configuration(self) -> OrderedDictType[str, Dict[str, Any]]:
      res = OrderedDict()
      res.update({'positions': {'value': [], 'timestamp': []}})
      return res

   def __init__(self, name, lattice, **kwargs):
      """
      Args:
         name: either a string attribute for tango or a pytac lattice
         random_state:
         **kwargs:
      """
      self.name = name
      self.lattice = lattice  # pyTAC lattice
      self.parent = None


if __name__ == "__main__":


   from pytac.load_csv import load
   from pytac.tango_cs import TangoControlSystem
   from pathlib import Path
   import logging
   import os

   os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'

   machine = 'ebs'

   logger = logging.getLogger(f"{machine}-bluesky-orm")

   lattice = load('EBS-vec',
               directory=Path('/machfs/liuzzo/EBS/beamdyn/python_LOCO_for_CTRM/pytac/src/pytac/data'),
               control_system=TangoControlSystem(wait=True, timeout=2.0))

   # test steerer directly with pyTAC
   h_orbit = lattice.get_value("orbit_h")
   v_orbit = lattice.get_value("orbit_v")

   bpms = BPMs(name="bpms", lattice=lattice)
   status = bpms.trigger()
   # I guess your bpms are much faster ...
   status.wait(1.0)
   data = bpms.read()
   print(data)

   print('hor bpm reading')
   print(data['positions']['value'][0])

   print('ver bpm reading')
   print(data['positions']['value'][1])

   # print(f'time: {timestamp}')
```

ORM measurement

```python
from bluesky.utils import ProgressBarManager

import time

from bluesky import RunEngine
from bluesky.callbacks.best_effort import BestEffortCallback
from databroker import Broker
from bluesky.plans import scan, rel_scan
from bluesky.simulators import summarize_plan

import numpy as np
import logging

from pytac.load_csv import load
from pytac.tango_cs import TangoControlSystem
from pathlib import Pa

import bpm
import steerer

import pickle

# work in EBS simula
import os
# os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'


def measure_orm(lattice, settle_time=6):
   """
   run a response matrix measurement
   """

   # initialize RM data structure
   orm = dict(HorSteererHorBPMs=[],
           VerSteererHorBPMs=[],
           HorSteererVerBPMs=[],
           VerSteererVerBPMs=[])

   # use bluesky run engine to run measurement

   # define some additional metadata
   md = dict(
      machine=machine,
      nickname="orm",
      measurement_target='Orbit Response Matrix',
      target="orbit response measurement",
      comment="",
   )

   # initialize run engine
   RE = RunEngine(md)

#    # add to each measurement additional data: current, tunes, emittances at
every measurment
#    sd = SupplementalData()
#    RE.preprocessors.append(sd)
#    sd.baseline = [det1, det2, det3, motor1, motor2]

   # Send all metadata/data captured to the BestEffortCallback
   bec = BestEffortCallback()
   RE.subscribe(bec)

   # Insert all metadata/data captured into a temporary db.
   db = Broker.named('temp')
   RE.subscribe(db.insert)

   # # set up a progress bar manager
   RE.waiting_hook = ProgressBarManager()

   det = bpm.BPMs(name='bpms', lattice=lattice)

   # horStrs = lattice.get_elements("HorSteerers")
   # verStrs = lattice.get_elements("VerSteerers")
   horStrs = lattice.get_elements("SF2A")
   verStrs = lattice.get_elements("SF2A")

   # test
   ste = steerer.Steerer(name=horStrs[0].get_pv_name('x_kick', 'setpoint'),
                   pytacsteerer=horStrs[0],
                   field='x_kick', settle_time=settle_time)

   print(det.read())
   print(ste.read())

   DK=1e-5

   Nsteerers = len(horStrs)

   def scan_steerers():
      # steerer_devs = lattice.get_element_pv_names('SH1A', 'x_kick',
'readback')
      # steerer_devs = lattice.get_element_pv_names('HorSteerers', 'x_kick',
'readback')
      for s in horStrs:
         yield from rel_scan([det],
                        steerer.Steerer(name=s.get_pv_name('x_kick', 'setpoint'),
                                  pytacsteerer=s,
                                  field='x_kick',
                                  settle_time=6),
                        -DK, DK, 2)
      # steerer_devs = lattice.get_element_pv_names('VerSteerers', 'y_kick',
'readback')
      for s in verStrs:
         yield from rel_scan([det],
                        steerer.Steerer(name=s.get_pv_name('y_kick', 'setpoint'),
                                  pytacsteerer=s,
                                  field='y_kick',
                                  settle_time=6),
                        -DK, DK, 2)

   # print what the plan will do
   summarize_plan(scan_steerers())

   print('wait 3s')
   time.sleep(3)

   # run measurement
   RE(scan_steerers())
```

```python
   # collect data from databroker
   hor_st_hor_cod_resp = np.zeros((Nsteerers, 320))
   hor_st_ver_cod_resp = np.zeros((Nsteerers, 320))
   ver_st_ver_cod_resp = np.zeros((Nsteerers, 320))
   ver_st_hor_cod_resp = np.zeros((Nsteerers, 320))

   for k in [n+1 for n in range(Nsteerers*2)]:
      header = db[-k]  # last, so order is swapped... very cumbersome way
to "save" data in a way it will be useless forever after
      t = header.table()
            #   +DK         -         -DK        / DK
      hor_resp = t.positions[2][0] - t.positions[1][0] / DK
      ver_resp = t.positions[2][1] - t.positions[1][1] / DK
esp
sp
_resp
_resp

      name_file = header.start['motors'][0]

      pickle.dump({'hor': hor_resp, 'ver': ver_resp, 'table': t, 'steer':
header.start['motors']},
               open(f'ORMdata_{name_file}.pkl', 'wb'))

   orm['HorSteererHorBPMs'] = hor_st_hor_cod_resp
   orm['VerSteererHorBPMs'] = ver_st_hor_cod_resp
   orm['HorSteererVerBPMs'] = hor_st_ver_cod_resp
   orm['VerSteererVerBPMs'] = ver_st_ver_cod_resp

   pickle.dump({'orm': orm}, open('ORM_bluesky_pyTAC.pkl', 'wb'))

   return orm


if __name__ == '__main__':

   # os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'

   machine = 'ebs'

   logger = logging.getLogger(f"{machine}-bluesky-orm")

   lattice = load('EBS-vec',

directory=Path('/machfs/liuzzo/EBS/beamdyn/MDT/2024/2024_06_01/pyta
c/src/pytac/data'),
               control_system=TangoControlSystem(wait=True,
timeout=2.0))

   from os.path import exists

   ormfile = '/machfs/MDT/2024/2024_06_01/ORM_bluesky_pyTAC.pkl'

   if not(exists(ormfile)):
      print(f'{ormfile} file does NOT exist')
      orm = measure_orm(lattice, settle_time=6)
   else:
      print(f'load existing {ormfile} file')
      d = pickle.load(open(ormfile, 'rb'))
      orm = d['orm']


   import matplotlib.pyplot as plt
   from mpl_toolkits.mplot3d import Axes3D

   # Set up grid and test data
   S=orm['HorSteererHorBPMs'].shape

   nx, ny = S[0], S[1]
   x = range(nx)
   y = range(ny)
   X, Y = np.meshgrid(x, y)  # `plot_surface` expects `x` and `y` data to be
2D

   plt.style.use('_mpl-gallery')

   fig, ax = plt.subplots(nrows=2, ncols=2, subplot_kw={"projection": "3d"})
   ax[0, 0].plot_surface(X, Y, orm['HorSteererHorBPMs'].T)
   ax[0, 0].set_ylabel('Hor BPM')

   ax[0, 1].plot_surface(X, Y, orm['VerSteererHorBPMs'].T)

   ax[1, 0].plot_surface(X, Y, orm['HorSteererVerBPMs'].T)
   ax[1, 0].set_xlabel('Hor steerers')
   ax[1, 0].set_ylabel('Ver BPM')

   ax[1, 1].plot_surface(X, Y, orm['VerSteererVerBPMs'].T)
   ax[1, 1].set_xlabel('Ver steerers')

   plt.savefig('ORM.png')
   plt.show()

   """
   fig, ax = plt.subplots(nrows=2, ncols=2)
   ax[0, 0].pcolormesh(orm['HorSteererHorBPMs'].T)
   # ax[0, 0].set_xlabel('Hor steerers')
   ax[0, 0].set_ylabel('Hor BPM')

   ax[0, 1].pcolormesh(orm['VerSteererHorBPMs'].T)

   ax[1, 0].pcolormesh(orm['HorSteererVerBPMs'].T)
   ax[1, 0].set_xlabel('Hor steerers')
   ax[1, 0].set_ylabel('Ver BPM')

   ax[1, 1].pcolormesh(orm['VerSteererVerBPMs'].T)
   ax[1, 1].set_xlabel('Ver steerers')

   plt.savefig('ORM.png')

   plt.show()
   """
```

Steerer ophyd

```python
import time
from typing import Dict, Any, List
from ophyd import BlueskyInterface
from ophyd.status import Status
from collections import OrderedDict
from ophyd.device import OrderedDictType

# temporary for safe-test purposes: force commands to simulator only.
import os
# os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'

__author__ = 'Simone Liuzzo (ESRF), Pierre Schnizer (HZB)'
__data__ = 'March 2024'


class Steerer(BlueskyInterface):
   """
   Read and set a steerer

   Parameters
   ----------
   name : string

   Example
   -------

   """
   settle_time = 6

   def stage(self, **kwargs):
      super().__init__(**kwargs)
      pass

   def unstage(self) -> List[object]:
      pass

   def trigger(self):
      return Status(success=True, done=True)

   def describe(self):
      """Return an OrderedDict with exactly the same keys as the ``read``
      method, here mapped to per-scan metadata about each field.
      """
      return dict(strength={'source': 'steerer', 'dtype': 'number', 'shape': []})

   def describe_configuration(self) -> OrderedDictType[str, Dict[str, Any]]:
      res = OrderedDict()
      res.update({'strength': {'source': 'steerer', 'dtype': 'number', 'shape': []}})
      return res

   def read(self):
      """Return an OrderedDict mapping string field name(s) to dictionaries
      of values and timestamps and optional per-point metadata.
      """
      print(self.name)
      val = self.pytacsteerer.get_value(self.field, "setpoint")  # we want to make sure that we use the
set points
      # bpms = self.name.get_elements('BPM') # using pyTAC

      return dict(strength=dict(value=val, timestamp=time.time()))


   def read_configuration(self) -> OrderedDictType[str, Dict[str, Any]]:
      res = OrderedDict()
      res.update({'strength': {'value': [], 'timestamp': []}})
      return res

   def set(self, val):
      print(self.pytacsteerer)
      print(self.pytacsteerer.get_pv_name('x_kick', 'readback'))
      # os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'
      th = os.environ['TANGO_HOST']
      print(f'tnago hosty{th}')
      self.pytacsteerer.set_value(self.field, val)
      print(f'set {self.field} to {val}.  now wait for {self.settle_time}')
      # AttributeProxy(self.name.replace(' ', '/')).write(val)
      time.sleep(self.settle_time) # hard coded for test

      return Status(success=True, done=True)

   def locate(self):

      return dict(readback=self.pytacsteerer.get_value(self.field, "readback"),
               setpoint=self.pytacsteerer.get_value(self.field, "setpoint"))

   def __init__(self, name, pytacsteerer, field, settle_time, **kwargs):
      """
      Args:
         name: either a string attribute for tango or a pytac lattice
         random_state:
         **kwargs:
      """
      super().__init__( **kwargs)

      self.name = name.replace('/', ' ')
      self.pytacsteerer=pytacsteerer  # pytac element, used to set/get strengths
      self.field = field
      self.parent = None
      self.settle_time = settle_time


if __name__ == "__main__":

   from pytac.load_csv import load
   from pytac.tango_cs import TangoControlSystem
   from pathlib import Path
   import logging
   import os

   # os.environ['TANGO-HOST'] = 'ebs-simu-1:10000'

   machine = 'ebs'

   logger = logging.getLogger(f"{machine}-bluesky-orm")

   lattice = load('EBS-vec',

directory=Path('/machfs/liuzzo/EBS/beamdyn/python_LOCO_for_CTRM/pytac/src/pytac/data'),
               control_system=TangoControlSystem(wait=True, timeout=2.0))
```

```python
   print(os.environ['TANGO-HOST'])
   time.sleep(10)
   # test steerer directly with pyTAC
   horStrs = lattice.get_elements("HorSteerers")
   print(f'readback from pyTAC: {horStrs[0].get_value("x_kick", "readback")}')
   print(f'setpoint from pyTAC: {horStrs[0].get_value("x_kick", "setpoint")}')
   time.sleep(10)
   initval = horStrs[0].get_value("x_kick", "setpoint")
   horStrs[0].set_value("x_kick", initval+1e-6)
   time.sleep(2)
   print(f'setpoint modified by pyTAC: {horStrs[0].get_value("x_kick", "setpoint")}')
   horStrs[0].set_value("x_kick", initval)
   time.sleep(2)
   print(f'setpoint restored by pyTAC: {horStrs[0].get_value("x_kick", "setpoint")}')

   # bluesky interface for pyTAC steerer
   steerer = Steerer(name="srmag/hst-sh1/c01-a/Strength",
               pytacsteerer=horStrs[0],
10)

   print(f'steerer initial from bluesky {steerer.name} set point')
   print(data['strength']['value'])
   initval=data['strength']['value']

   steerer.set(data['strength']['value'] + 1e-6)

   time.sleep(2)

   datam = steerer.read()
   print(datam)

   print(f'steerer modified by bluesky {steerer.name} set point')
   print(datam['strength']['value'])

   # restore initial value
   steerer.set(data['strength']['value'])

   dataf = steerer.read()
   print(dataf)

   print(f'steerer restored by bluesky {steerer.name} set point')
   print(dataf['strength']['value'])
```

pyTAC

numpy.arrays

Bluesky

Database

```
# Insert all metadata/data captured into a temporary db.
db = Broker.named('temp')
RE.subscribe(db.insert)
```

db is the database

first index in database is the most recent, must read it backward

```
for k in [n+1 for n in range(Nsteerers*2)]:
    header = db[-k]
    t = header.table()

               #   +DK            -          -DK        / DK
    hor_resp = t.positions[2][0] - t.positions[1][0] / DK
    ver_resp = t.positions[2][1] - t.positions[1][1] / DK
```

H

V

determined by Steerers Ophyd device

MUST RECALL/UNDERSTAND the order of the scan. First –DK then +DK

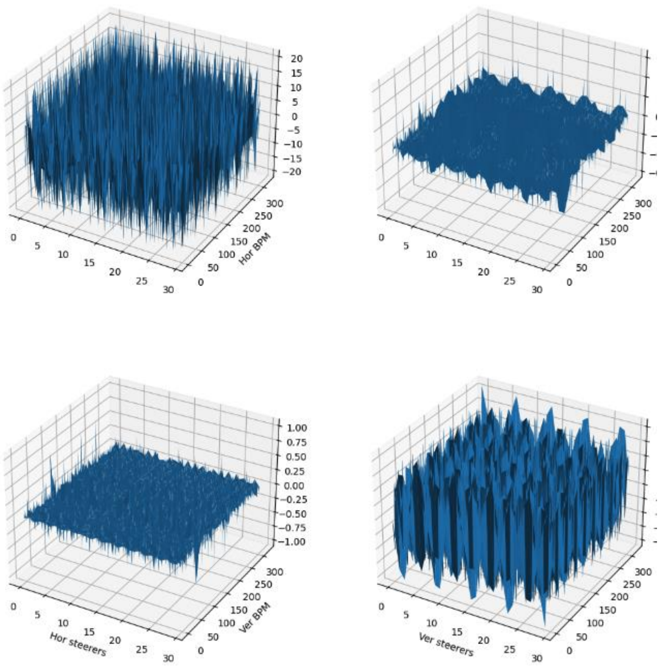
```
>>> d = pickle.load(open('/machfs/liuzzo/EBS/beamdyn/MDT/2024/2024_06_01/ORMdata_srmag hst-sf2 c10-a Strength.pkl','rb'))
>>> d['table']
                              time                                                positions  strength
seq_num
1       2024-05-30 12:18:32.410099506  [[5.781650598015143e-05, -2.7994524738073546e-...  -0.00001
2       2024-05-30 12:18:38.433176517  [[-5.79179119039193e-05, 2.8087759775409007e-0...   0.00001
```

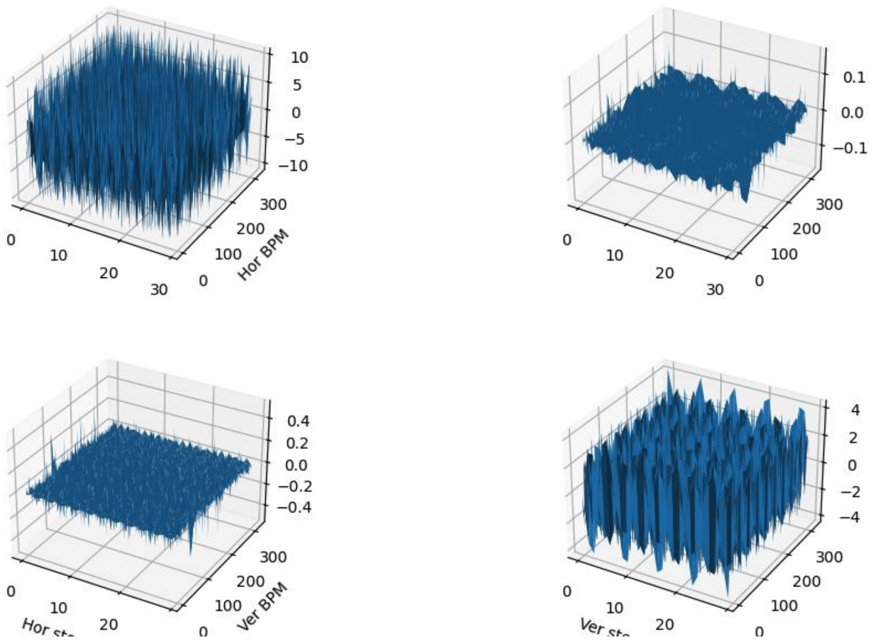
N.B. db object may not be pickled (saved to file)

SIMULATIONS

pyTAC

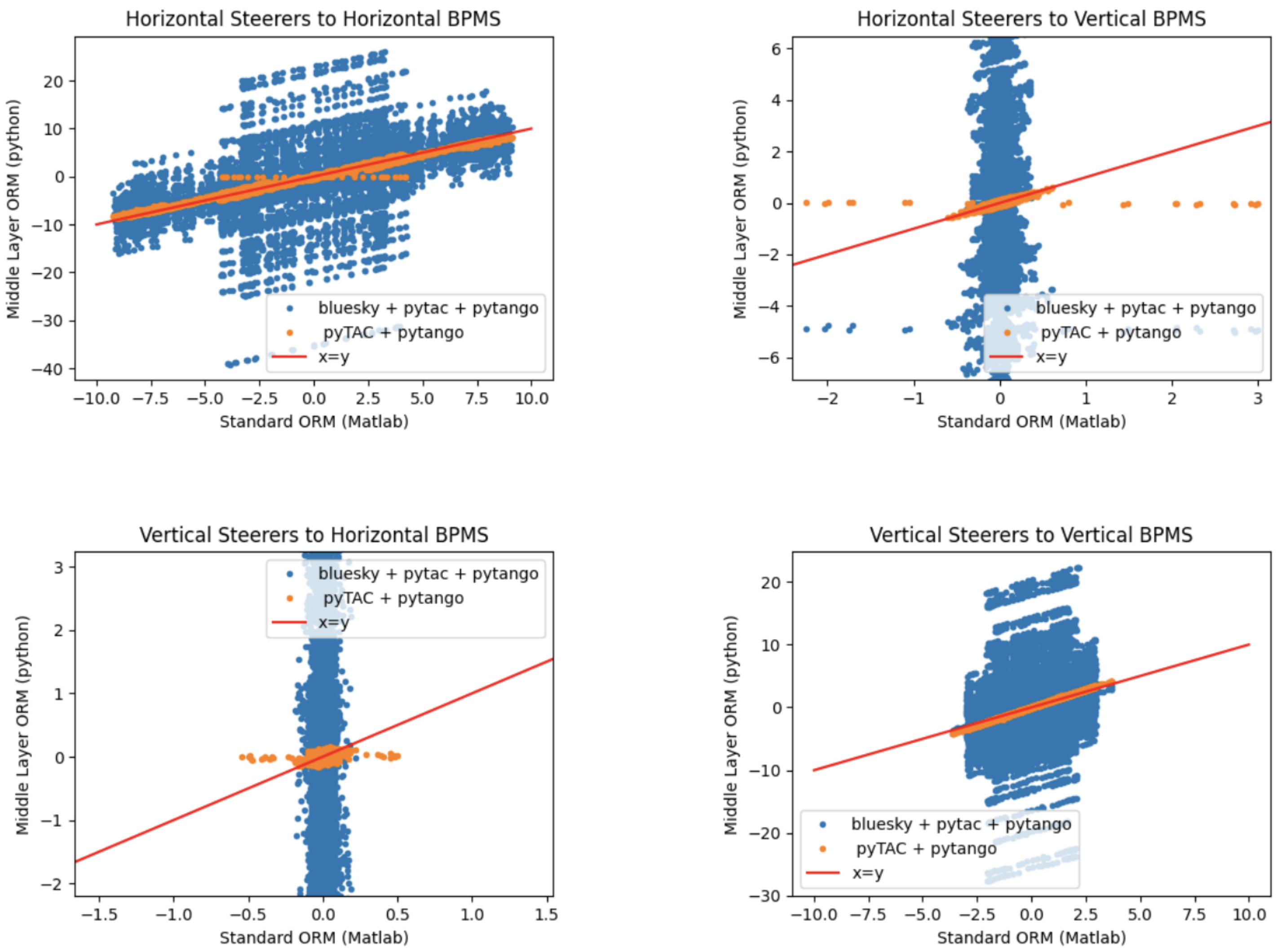
pyTAC + Bluesky





pyTAC == Bluesky + pyTAC == Matlab measurement

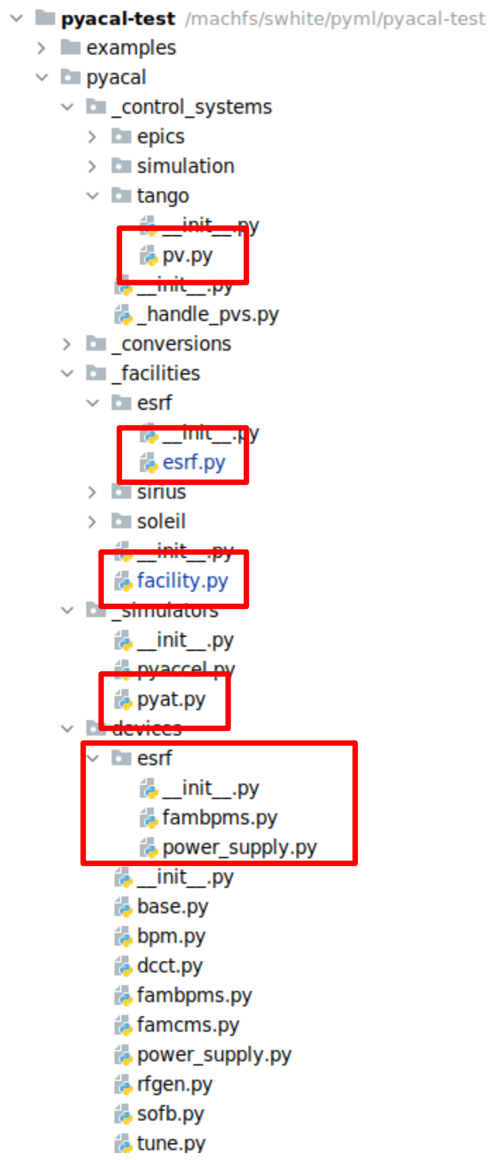
MEASUREMENT WITH BEAM, 2ND JUNE 2024



MEASUREMENT

Issue with data saving/synchro in bluesky.
No idea where to look to fix it.

PYACAL ADAPTED FOR ESRF



To get it working at ESRF few features had to be added modified:

- It took a total 3 days of work + 1 meeting for the SIRIUS team to have some thing running on the ESRF simulation
- New features:
  - TANGO interface
  - pyAT interface
  - ESRF facility definition
  - ESRF specific devices
- Few additional minor modifications to load AT model from file for example
- This implementation should be usable by any TANGO/pyAT users as long as the facility is defined in the subpackage ***facilities***

TEST OF PYACAL AT ESRF

```
[1]: import sys
     sys.path.append('/machfs/swhite/pyml/pyacal-test')
     import pyacal
     import os
     import warnings
     import at
     import matplotlib.pyplot as plt
     import time
     import numpy
     plt.rcParams['figure.figsize'] = [20, 10]
     from pyacal.experiments.disp_chrom import DispChrom
```

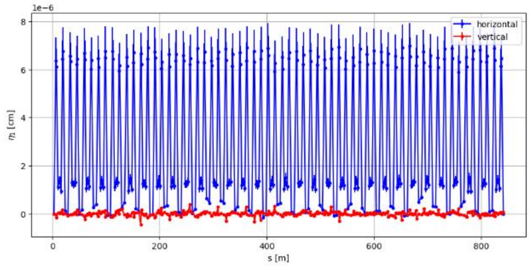
```
[2]: pyacal.set_facility('esrf')
     with warnings.catch_warnings():
         warnings.simplefilter("ignore")
         path = '/operation/beamdyn/matlab/optics/sr/theory/'
         ring = at.load_lattice('./betamodel.mat', use='betamodel')
     pyacal.set_model('EBS', ring)
```

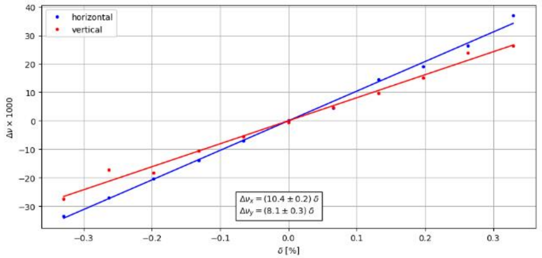
```
[3]: dispchrom = DispChrom(accelerator='EBS', isonline=True)
```

```
[4]: dispchrom.params.max_delta_freq = +100  # [Hz]
     dispchrom.params.min_delta_freq = -100  # [Hz]
     dispchrom.params.wait_tune = 10 # [s]
     dispchrom.params.meas_nrsteps = 11
     print(dispchrom.params)

     max_delta_freq [Hz]      =   100.000
     min_delta_freq [Hz]      =  -100.000
     meas_nrsteps             =        11
     wait_tune [s]            =    10.000
     sofb_nrpoints            =        10
```

```
[25]: dispchrom._run()
```





Chromaticity and dispersion measurement notebook and results

Once debugged on the ESRF simulator PYACAL could be launched on the ESRF-EBS machine without any further modifications

Tested 2 experiments:

- ORM measurement
- Dispersion and chromaticity measurement

Those are the simplest examples provided

CONCLUSIONS AND REMARKS

pyAML is a project to replace and improve MML. The collaboration is growing and the collaboration is actively participating to the software development.

Networking funds proposal has been submitted. Reply expected in 1 year from now.

On going exploratory tests and detailed definition of specifications and use cases.

Some remarks:

- It was relatively straight forward to adapt pyTAC and PYACAL to ESRF-EBS.
- pyTAC was also adapted to HZB, pyACAL was also adapted to MAXIV.
- pyTAC calibration and set up is based on files, rather easy to interpret and user friendly
- pyTAC + bluesky resulted in very high level of complication. The easy scripting friendliness is not evident for this case.
- For PYACAL once the TANGO and pyAT interface had been implemented all “higher” level applications worked out of the box
- PYACAL was developed with EPICS in mind with the present architecture we cannot profit form certain TANGO features such as Devices or Family Devices: for the test workaround were implemented to make it work
- PYACAL give access to a full suite of measurements and controls scripts with potentially minor or no modifications needed

Next steps:

- Run more complex experiments: BBA, Orbit feedback?
- We would be very interested in testing pyQT GUI at ESRF
- Can we run experiments on the model? Is the switch to simulation implemented in the test version?