# Building Digital Models with thor_scsi
# An Evolutionary Approach

Waheedullah Sulaiman Khail, Pierre Schnizer, Paul Goslawski

Helmholtz-Zentrum Berlin (HZB), Germany

31. August 2023

# Overview

# Acknowledgement



**Johan Bengtsson** for preparing his code base, the updated documentation of the physics and maths involved [1], many tests and reviews of the developed code, reimplementing linear optics optimisation code in python, teaching proper dynamics. . . , kayaking

**Markus Ries** practical machine steering knowledge . . . good nerves

**Guabao Shen** for NSLS II virtual accelerator code share

**Thomas Birke** introduction to EPICS control system

**BESSY II and MLS** all people that make it all actual work

**all** that I am not even aware that they make my work possible

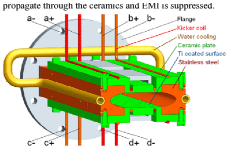
# Refactoring: Overview

- code basis: split up
- lattice parser ← FLAME [2]
- TPSA → gtpsa [3] ← gtpsa-cpp
- modernised language "std::" containers, "arma::mat" for matrices (interface)
- autotools → cmake
- split up: multipole evaluation → field kick
  - delegates:
    - field interpolation
    - radiation calculation (only if there)
  - lets observe: phase space

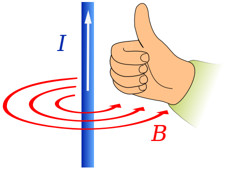
  thus fine grained control if required or not
- python interface ← pybind11 [4] → elements in pyton
- many parameters: double or truncated power series objects
- worked on user interface simplification

# Machine elements in python

## Example: non linear kicker




[5]

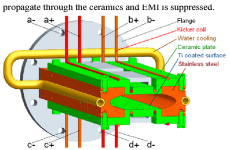



```python
class AirCoilMagneticField(tslib.Field2DInterpolation):
    """Field of an air coil"""

    def __init__(self, *, positions, currents):
        tslib.Field2DInterpolation.__init__(self)

    def field_py(self, pos, field):
        x, y = pos
        dz = x + y * 1j - self.positions  # offset from wire
        r = np.absolute(dz),  phi = np.angle(dz)
        B = (self.precomp * 1 / r * np.exp((phi + np.pi / 2) * 1j)).sum()
        field[0], field[1] = B.imag, B.real


class NonlinearKickerField(AirCoilMagneticField):
    """Field created by a classical telephone transmission cable"""

    def __init__(self, *, pos, current):
        p = np.array([pos, pos.conjugate(), -pos.conjugate(), -pos])
        currents = np.array([current] * len(pos)) * [1, -1, -1, 1]
        AirCoilMagneticField.__init__(self, positions=pos, currents=currents)
```

Source: Wikipedia
by Jfmelero
Element in Python
Called from C++ code

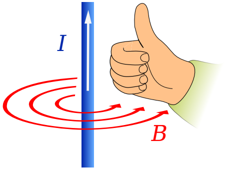
# Machine elements in python

## Example: non linear kicker




[5]

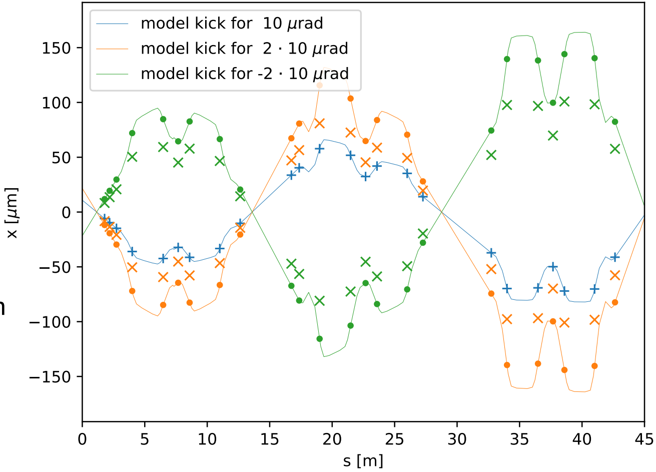
```
struct aircoil_filament {
    double x, y, current;
};

template<class C>
AirCoilMagneticFieldKnobbed(
    const std::vector<aircoil_filament_t> filaments,
    const double scale=1e0);
template<typename T>
inline void _field(const T& x, const T& y, T *Bx, T *By) const {
    const double precomp = mu0 / (2 * M_PI) * this->m_scale;
    *Bx = *By = 0e0;
    for(const auto& f: this->m_filaments){
        const T dx=x-f.x, dy=y-f.y, r2=dx*dx + dy*dy; // offset from wire
        *By += precomp * f.current / r2 * dx;
        *Bx += precomp * f.current / r2 * dy;
    }
}
```




Source: Wikipedia
by Jfmelero
Element in Python
Called from C++ code

# Model based Beam based alignment

- 1. quad: $\Delta K, -\Delta K \rightarrow$ derive orbit distortion
- ideal orbit distortion for this quad
- expected distortions: bpm measurement .
- measured distortions: bpm measurement x
- scale expected to measurement $\rightarrow$ dipole from feed down $\rightarrow$ quadrupole offset



# Truncated power series: variables by name

## Variables, knobs by name

- dimension names

```
d = dict(x=0, px=1, y=2, py=3, delta=4, ct=5,
         K=6, dx=7, dy=8)
named_index = gtpsa.IndexMapping(d)
```

- variables

```
delta.set_variable(1e-3, "delta")
```

- knobs

```
dx.set_knob(1e-3, "dx")
```

## Quadrupole strength as knob

## Access to field advance

# Truncated power series: variables by name





## Variables, knobs by name

## Quadrupole strength as knob

```
k_org = magnet.get_main_multipole_strength()
k = gtpsa.ctpsa(desc, po,
                mapping=named_index)
k.set_knob(k_org, "K")
muls = magnet.get_multipoles()
muls.set_multipole(2,
                   gtpsa.CTpsaOrComplex(k))
```

## Access to field advance

# Truncated power series: variables by name

## Variables, knobs by name

## Quadrupole strength as knob

## Access to field advance

```
dq_dK = ps.ct.get(K=1)
```

# Example beam based alignment: phase advance

Cross check

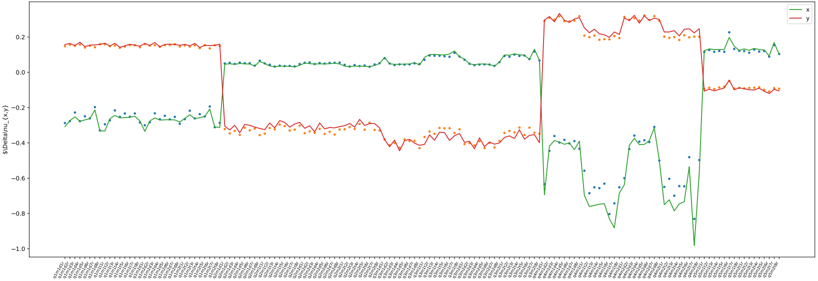
- predicted phase advance
- measured by tune measurement
- cross check of: polarity $\Delta K$ applied in machine

## Field advance computed

- instrumented quad $K$
- propagated phase space

```
dq_dK = ps.ct.get(K=1)
```

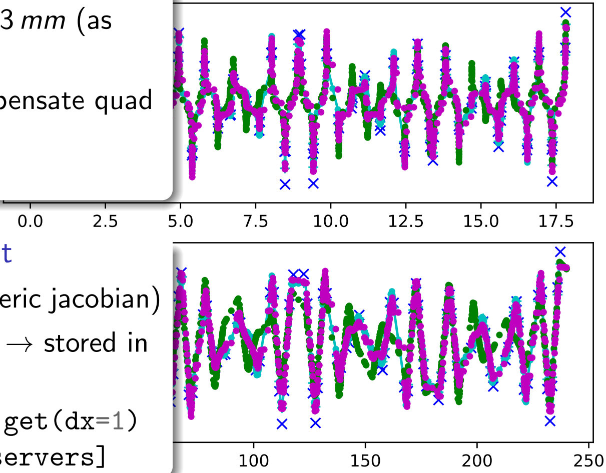
Similar approach for orbit distortion due to $K$



# Beam based alignment: distorted orbit

## Model setup

- quadrupole displaced 0.3 *mm* (as knob)
- artifical steerer ← compensate quad feed down
- $\Delta K$ 2%




## Comparison: distorted orbit

- Closed orbitfinder (numeric jacobian)
- phase space at element → stored in observer → extract:

```
dx = [ob.getTPSA().get(dx=1)
          for ob in observers]
```

# Data models

## Simplify processing

### Definition

- intuitve schema of used data
- uses:
  - sub data models
  - primitive types

### Example: BBA

measurements for magnet → measurement point → bpm's → bpm planes

# Recommendations I

## Start: definitions

- target
- basis
- Cross check with original author

Very useful: documentation of physics model [1]

## Start: perparations

- code parts: standard libraries → replacement
- version control system
- automatic documentation tool (sphinx, doxygen,)

# Recommendations II

## Refactoring preparation

- work plan → "identify rip apart and reassemble"
- build and test system (run frequently)
- Build up of test system
  - total function test
  - "saftey warnings"

## Refactoring: Step I

- upgrade code base → modern standard
- as long as checkable with test base

End: Hold point: upgraded code base

# Recommendations III

### Refactoring: Step II

- Start with largest intervention
- Run full funtion test (e.g. with compatability layer)

### Refactoring: cont.

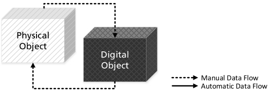
similar to above

### Don't forget

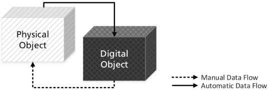
- distribute early
- distriubte often

Detailed in [6]

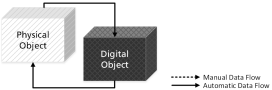
# Digital twin: nomen clature




model




shadow




twin

## Status

- different beam dynamics models available ← interaction
- **M**atlab **M**iddle **L**ayer [7] + **A**ccelerator **T**oolbox: [8] similar simulation many different ring light sources
- further online model implementations for ring light sources: DIAMOND [9], NSLS II [10], SLS [11] Solaris [12],
- FLAME: FRIB online model [2]

## Need to go further?

- 20 years → experience gained
- software industry → Futures (“beer garden” buzzers), async, $\mu$-services
- compare: iso standard , functional mock-up interface

split up [13], iso standard [14]

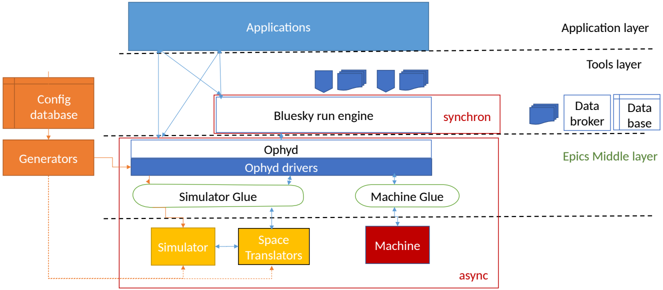
# Digital twin

## Developments performed



- started with Tracy-2 and Guabao's implementation [10]
- first motivation to refactor Tracy 2 (setting multipoles with methods)
- Tracy-2 $\rightarrow$ thor-scsi $\rightarrow$ python wrapper
- implemented as IOC using pydevice [15, 16]
- REST-API interface
- Data models being developed
- Bluesky[17] based measurement scripts
- used as basis for refactoring (model based) **B**eam **B**ased **A**lignment

# ~~Architecture~~ chosen




## Simulator

- virtual accelerator (as PyEPICS IOC[1])
  - properties: getter / setters
  - requesting (delayed) calculations
  - EPICS interface: records update data export
- facade
  - initialisation
  - calculation functors
  - method resolvers

## Space translators

- engineering $\leftrightarrow$ physics
- implemented as EPICS IOC'S $\rightarrow$ update on machine change

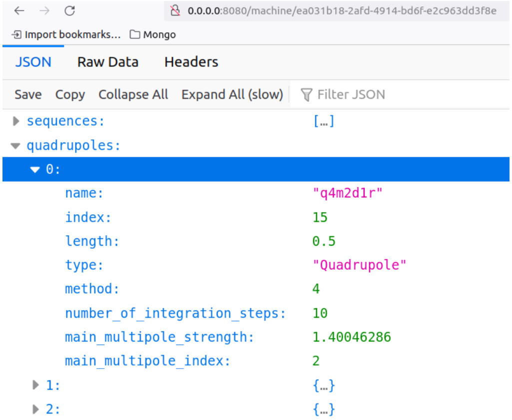
## Applications

- "Bluesky" as interface
- standard applications REST API based

[1] input output controller

# REST API: advantages

- proper data model → data stored in database
- with a few lines of python code:
- standard services: display
  - data model / schema
  - data services



# Model Beam based alignment



## Uncertainties

- Quadrupole $\Delta K = f(\Delta I)$
- calculated quadrupole offset

## Cross check

- $\Delta K = f(\Delta I)$ : measure tune advance, predict in model $\rightarrow$ compare
- offset:
  - "displace" quadrupole in model
  - add "compensating" steerer at exact same position
  - run measurement script versus model

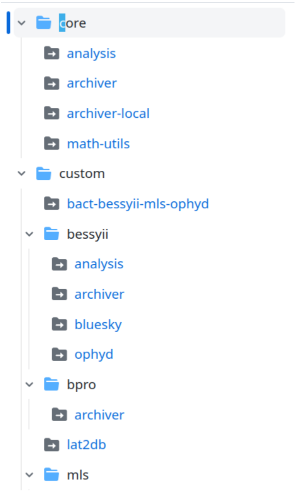
# Digital twin for different machines

## simplification: data model

- devices
- preprocessed (physics ready) data
- analysis results

## Software: split up

- **B**erlin **a**ccelerator **c**omissioning **t**oolkit
  - split up in (sub)repositories)
  - functionalities:
    analysis, Ophyd drivers, Bluesy plans
  - split up:
    - core: machine independent
    - custom
- epics IOC: `dt4acc`
  - engineering $\leftrightarrow$ physics
  - pushing data to simulator



# What's missing: data models

## proper data models

representation of

- measured (raw) data of devices
- preparation of calculations
- analysis results

## Modelling individual blocks

micro-service like structure see functional modelling standard

## Physics Engineering Conversion

Current implementation

- based as EPICS IOC
- loaded from text file
- needs: proper data models

# Digital twin: the other world

### Accelerator community

- Online models
- Switchable models

### Aerospace community

For space probes

- analogue twins
- analogue twin test beds
- digital models of subsystems
- digital test beds

### Think different: PLC[2]

- Design machine
- Design control application
- Implement control application vs digital twin
- switch over to real machine

### Naval industry

- Functional mock-up interface (time concept!)
- open simulation platform

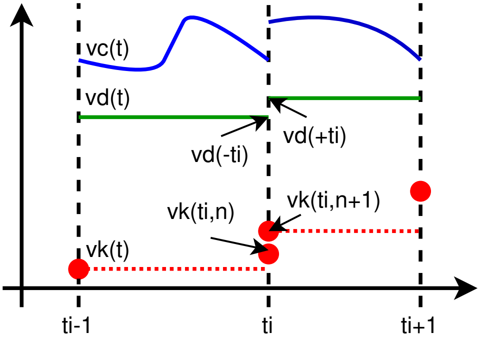
[2] PLC: Product life cycle

# Functional mock-up interface / Open simulation platform



- Industry approach: split up in Functional mock-up units (C-libraries)
  - ODE[3]equation, (sub)simulation, time concept
  - derivatives **Jv**
  - loading settings / state
  - XML: description
- integration: open simulation platform open integration platform [18, 19]: separation of conversion / communication
- alternate approach: Lume project (LCLS-II)

## Concept of time




advancing time

[3] Ordinary Differential Equation

# Conclusion

- Tracy II → thor_scsi → refactored same code base
  - tracking (doubles)
  - g(tpsa)
- digital twin
  - based on: available tools
    - bluesky
    - REST API . . .
  - focus: existing machines: BESSY II & MLS
  - implementation: separated in subpackages
- started data models
  - devices
  - preprocessed (physics ready) data
  - analysis result

# Conclusion

- Tracy II → thor_scsi → refactored same code base
  - tracking (doubles)
  - g(tpsa)
- digital twin
  - based on: available tools
    - bluesky
    - REST API ...
  - focus: existing machines: BESSY II & MLS
  - implementation: separated in subpackages
- started data models
  - devices
  - preprocessed (physics ready) data
  - analysis result

What's happening:

elsewhere ?

in industry ?

Looking into

functional mock-up interface

open simulation platform

lume project

Towards a full facility digital twin?

Adapt to our needs