

PLACET2, an upgrade of PLACET to simulate recirculating beamlines

Dario Pellegrini (CERN, EPFL)
Presented by: Andrea Latina (CERN)

LCWS - Nov 3, 2015

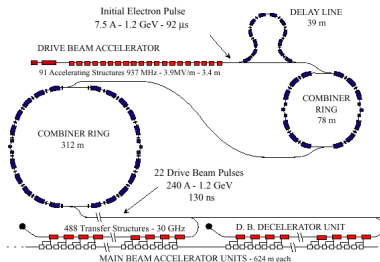


Recirculating Machines

- Recirculating machines are a special class of particle accelerators in which the beam circulates a few times in the same elements.
- Neither linac nor ring codes are fully suited.
- The bunch train is often modified during operation, split over multiple beamlines, and recombined (non-fixed bunch spacing and/or ordering).
- Multibunch effects in these machines are typically handled with small, *ad hoc* codes.

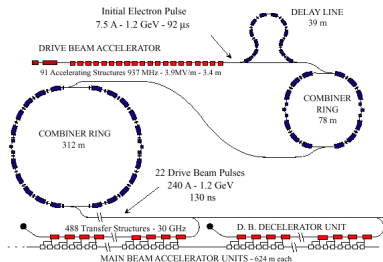
Recirculating Machines

- Recirculating machines are a special class of particle accelerators in which the beam circulates a few times in the same elements.
- Neither linac nor ring codes are fully suited.
- The bunch train is often modified during operation, split over multiple beamlines, and recombined (non-fixed bunch spacing and/or ordering).
- Multibunch effects in these machines are typically handled with small, *ad hoc* codes.



Recirculating Machines

- Recirculating machines are a special class of particle accelerators in which the beam circulates a few times in the same elements.
- Neither linac nor ring codes are fully suited.
- The bunch train is often modified during operation, split over multiple beamlines, and recombined (non-fixed bunch spacing and/or ordering).
- Multibunch effects in these machines are typically handled with small, *ad hoc* codes.



Many different and complex topologies!

PLACET2: Recirculating Placet

Why a code designed to handle such machines?

Fundamental reason: compute the impact of multibunch effects with beam recirculation:

- *Long Range Wakefields - Higher Order Modes,*
- ion/electron clouds,

PLACET2: Recirculating Placet

Why a code designed to handle such machines?

Fundamental reason: compute the impact of multibunch effects with beam recirculation:

- *Long Range Wakefields - Higher Order Modes*,
- ion/electron clouds,

...on top of a *full 6D tracking* with a number of single particle/single bunch effects:

- *Incoherent* and Coherent *Synchrotron Radiation*,
- *Short Range Wakefields and Impedances*,
- *Beam-Beam*.

PLACET2: Recirculating Placet

Why a code designed to handle such machines?

Fundamental reason: compute the impact of multibunch effects with beam recirculation:

- *Long Range Wakefields - Higher Order Modes*,
- ion/electron clouds,

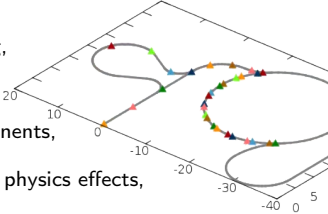
...on top of a *full 6D tracking* with a number of single particle/single bunch effects:

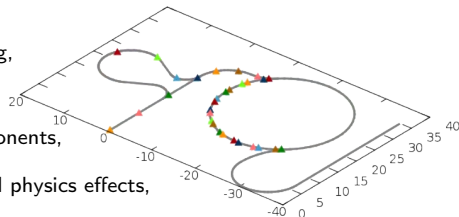
- *Incoherent* and Coherent *Synchrotron Radiation*,
- *Short Range Wakefields and Impedances*,
- *Beam-Beam*.

Corollary benefits:

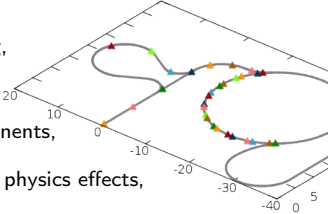
- Clearer and easy-to-maintain scripts (no multiple definition of the same element, lattice unrolling, ...),
- Alignment and timing errors are correctly applied and propagated to the whole machine,
- Can do global optimisations taking into account different bunch paths.

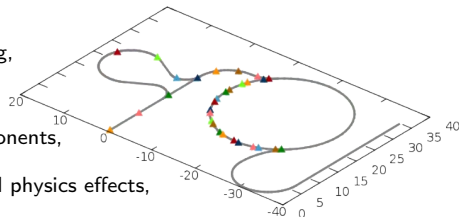
What does PLACET2 ask you?

- Definition of one or more *injectors*:
 - create and store bunches for tracking,
 - define the initial beam structure.
 - Definition of one or more *beamlines*:
 - standard, simple sequences of components,
 - include "beam instrumentation",
 - equip the elements with the required physics effects,
 - Definition of one or more *dumps*:
 - delete the bunches, freeing the memory,
 - provide additional instrumentation.
 - Connect together injectors, beamlines and dumps to create a *machine*:
 - complete description of the accelerator,
 - possible to adjust/edit properties after the previous definitions.
- 
- A 3D plot showing a complex, winding particle accelerator beamline. The beamline is represented by a series of connected line segments, each marked with a small colored triangle (red, green, blue, orange, yellow, pink). The plot is set in a 3D coordinate system with axes labeled from -40 to 20. The beamline starts at the top left, curves around, and ends at the bottom right.



What does PLACET2 ask you?

- Definition of one or more *injectors*:
 - create and store bunches for tracking,
 - define the initial beam structure.
 - Definition of one or more *beamlines*:
 - standard, simple sequences of components,
 - include "beam instrumentation",
 - equip the elements with the required physics effects,
 - Definition of one or more *dumps*:
 - delete the bunches, freeing the memory,
 - provide additional instrumentation.
 - Connect together injectors, beamlines and dumps to create a *machine*:
 - complete description of the accelerator,
 - possible to adjust/edit properties after the previous definitions.
 - Run the machine to obtain beam data!
- 



What does PLACET2 do for you?

- Tracks all the bunches determining their path in the machine at runtime,
- In case of many bunches, guarantees that each of them enters each beamline in the correct order,
- Handles all the time dependencies in the machine (element phases, HOMs damping, ...),
- Collects beam properties at the desired locations, bunch after bunch,
- Allows to follow a bunch along its path collecting many observables (Twiss, orbit, transport matrix/higher order tensors, losses, ...).

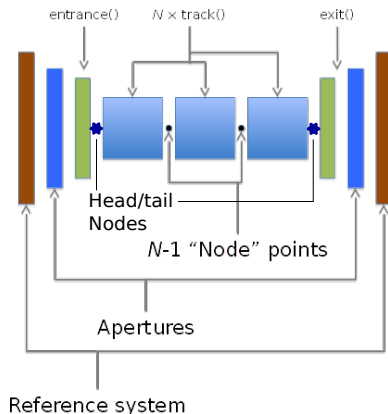
What is there behind the curtain?

PLACET2 is a full 6D tracking code:

- Totally new core written in modern C++ from scratch;
- Comes with a SWIG-generated Tcl interface plus an embedded Octave interpreter;
- Designed to be easily expanded (add more beam models, more elements, more physics effects...);
- The interface is mostly compatible with PLACET, featuring a new set commands (e.g. beam creation, element slicing, machine topology definition, ...);
- in a good state of development, under constant expansion, already being tested in a number of cases (LHeC and PERLE beam dynamics studies, simulations of the CTF3 combiner ring);
- Currently available as a PLACET branch in the clicsw SVN repository.

Insights on the PLACET2 mechanisms

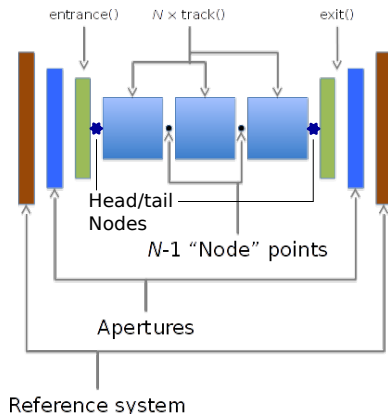
Structure of the element: an embedded integrator



Element as a collection of kicks:

- Misalignment/Aperture/Fringe fields;
- Sliced *thick* core (drift, dipole, quadrupole, RF cavity...);
- *Thin* kicks added between the slices to import physical effects (Radiation, wakefields, multipolar component, stray fields...).

Structure of the element: an embedded integrator



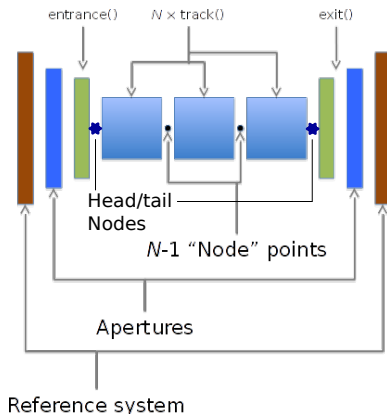
Element as a collection of kicks:

- Misalignment/Aperture/Fringe fields;
- Sliced *thick* core (drift, dipole, quadrupole, RF cavity...);
- *Thin* kicks added between the slices to import physical effects (Radiation, wakefields, multipolar component, stray fields...).

Huge flexibility:

- Select which effects to add in each element;
- Split an effect over multiple kicks.

Structure of the element: an embedded integrator



Element as a collection of kicks:

- Misalignment/Aperture/Fringe fields;
- Sliced *thick* core (drift, dipole, quadrupole, RF cavity...);
- *Thin* kicks added between the slices to import physical effects (Radiation, wakefields, multipolar component, stray fields...).

Huge flexibility:

- Select which effects to add in each element;
- Split an effect over multiple kicks.

Generalised kicks: they do not necessary "kick" the beam but can:

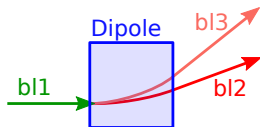
- transform the beam (match Twiss, phase advance, generate offsets, ...);
- collect the required properties of the passing bunches.

Beamlines connections and bunch routing

Example: dipole splitter

Routing according to energy:

- Beamline 1 ends at the dipole;
- Beamline 2 and 3 have two dipoles tuned at different energies.

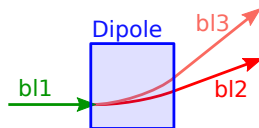


Beamlines connections and bunch routing

Example: dipole splitter

Routing according to energy:

- Beamline 1 ends at the dipole;
- Beamline 2 and 3 have two dipoles tuned at different energies.



```
1 | myMachine link -in bl1 -out bl2 -cmd {abs([bunch getd energy] - 20)}
2 | myMachine link -in bl1 -out bl3 -cmd {abs([bunch getd energy] - 10)}
```

Beamlines connections and bunch routing

Example: dipole splitter

Routing according to energy:

- Beamline 1 ends at the dipole;
- Beamline 2 and 3 have two dipoles tuned at different energies.

```
1 | myMachine link -in bl1 -out bl2 -cmd {abs([bunch getd energy] - 20)}  
2 | myMachine link -in bl1 -out bl3 -cmd {abs([bunch getd energy] - 10)}
```

Routing according to offset:

- Beamline 1 contains the dipole, bunches at different energies end up with different positions;
- Beamline 2 and 3 start after the dipole.

```
1 | myMachine link -in bl1 -out bl2 -cmd {abs([bunch getd cx] - 2e-3)}  
2 | myMachine link -in bl1 -out bl3 -cmd {abs([bunch getd cx] - 1e-3)}
```

Beamlines connections and bunch routing

Example: dipole splitter

Routing according to energy:

- Beamline 1 ends at the dipole;
- Beamline 2 and 3 have two dipoles tuned at different energies.

```
1 | myMachine link -in bl1 -out bl2 -cmd {abs([bunch getd energy] - 20)}
2 | myMachine link -in bl1 -out bl3 -cmd {abs([bunch getd energy] - 10)}
```

Routing according to offset:

- Beamline 1 contains the dipole, bunches at different energies end up with different positions;
- Beamline 2 and 3 start after the dipole.

```
1 | myMachine link -in bl1 -out bl2 -cmd {abs([bunch getd cx] - 2e-3)}
2 | myMachine link -in bl1 -out bl3 -cmd {abs([bunch getd cx] - 1e-3)}
```

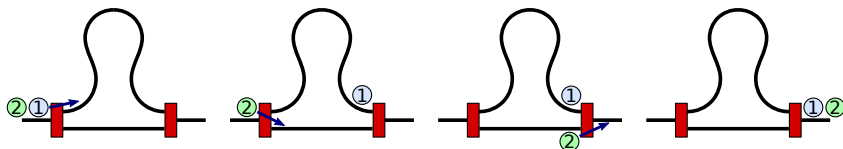
- The *bunch* keyword in the interface, becomes the actual bunch object being tracked by the C++ core: you can call its member functions (*getd* for instance)!
- The bunch proceeds in the beamline whose command evaluates to the *smaller* value.
- Beamlines merges are simpler: do not require to specify a routing criterion.

Machine operation and Synchronisation

- The machine owns *global timer* used for synchronisation → increases at small steps;
- Each bunch owns an *internal timer* → increases as bunch travels through thick elements;
- Bunches travel straight down beamlines, their timer can be greatly increased,
- but are forced to wait at the subsequent joints until the global timer exceeds their internal one.

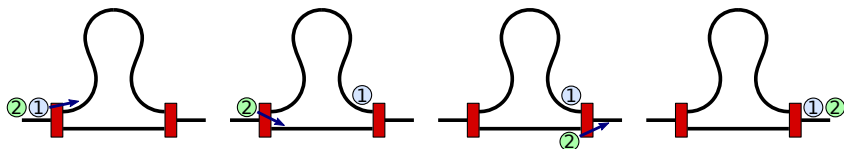
Machine operation and Synchronisation

- The machine owns *global timer* used for synchronisation → increases at small steps;
- Each bunch owns an *internal timer* → increases as bunch travels through thick elements;
- Bunches travel straight down beamlines, their timer can be greatly increased,
- but are forced to wait at the subsequent joints until the global timer exceeds their internal one.



Machine operation and Synchronisation

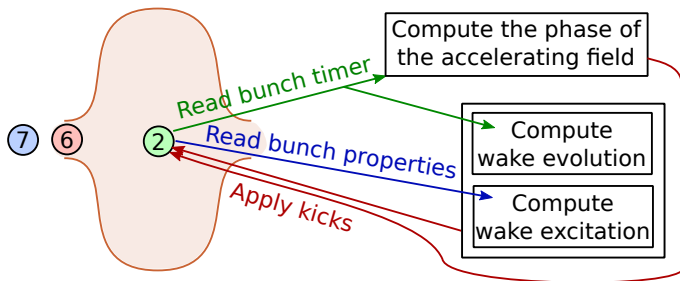
- The machine owns *global timer* used for synchronisation → increases at small steps;
- Each bunch owns an *internal timer* → increases as bunch travels through thick elements;
- Bunches travel straight down beamlines, their timer can be greatly increased,
- but are forced to wait at the subsequent joints until the global timer exceeds their internal one.



- *Global timer steps smaller than shortest beamline* ⇒ bunch order is preserved;
- Elements always see bunches in the correct time sequence ⇒ *the element time is the bunch-being-tracked time*.

An example: long range wakefields

- A global computation is prohibitive in recirculating machines where bunches are shuffled.
- Model the local bunch-mode interaction and apply it in all the relevant elements.



- This only requires the correct propagation of the bunches.

Simulation of the Phase Monitor output signal

- The Phase Monitor (BPR) multiplies the beam induced signal with a reference sine function.
- Makes possible to monitor the phase stability of a train and/or turn after turn.

Simulation of the Phase Monitor output signal

- The Phase Monitor (BPR) multiplies the beam induced signal with a reference sine function.
- Makes possible to monitor the phase stability of a train and/or turn after turn.

Signal Shaping:

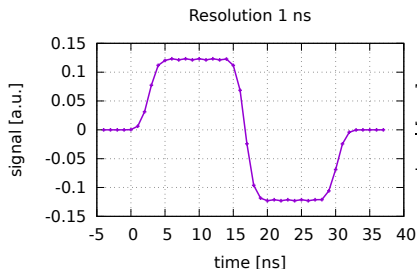
- Slice the beam longitudinally;
- Multiply the charge in each slice with the reference function;
- Convolve with a gaussian function to simulate the time resolution;
- The signal is collected in a histogram which extends as more bunches comes.

Simulation of the Phase Monitor output signal

- The Phase Monitor (BPR) multiplies the beam induced signal with a reference sine function.
- Makes possible to monitor the phase stability of a train and/or turn after turn.

Signal Shaping:

- Slice the beam longitudinally;
- Multiply the charge in each slice with the reference function;
- Convolve with a gaussian function to simulate the time resolution;
- The signal is collected in a histogram which extends as more bunches comes.

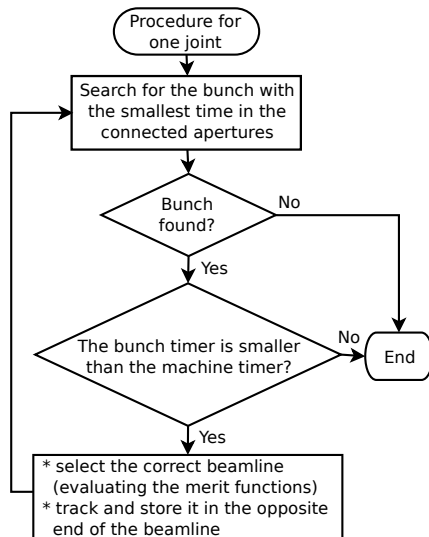
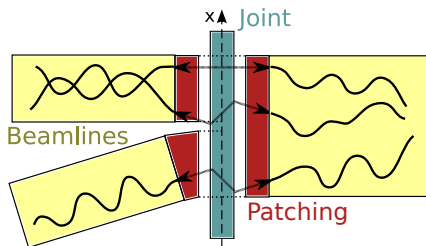


The Running Algorithms

Machine Operation

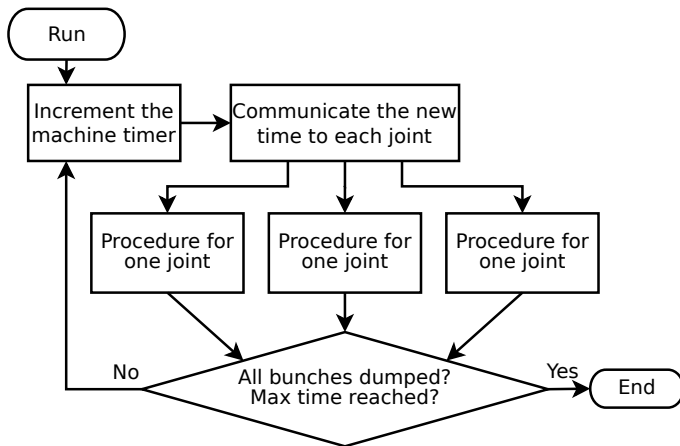
Advance Procedure

- Transmit the *machine time* to each joint.
- Each joint applies the advancing algorithm.



Machine operation

Run Procedure

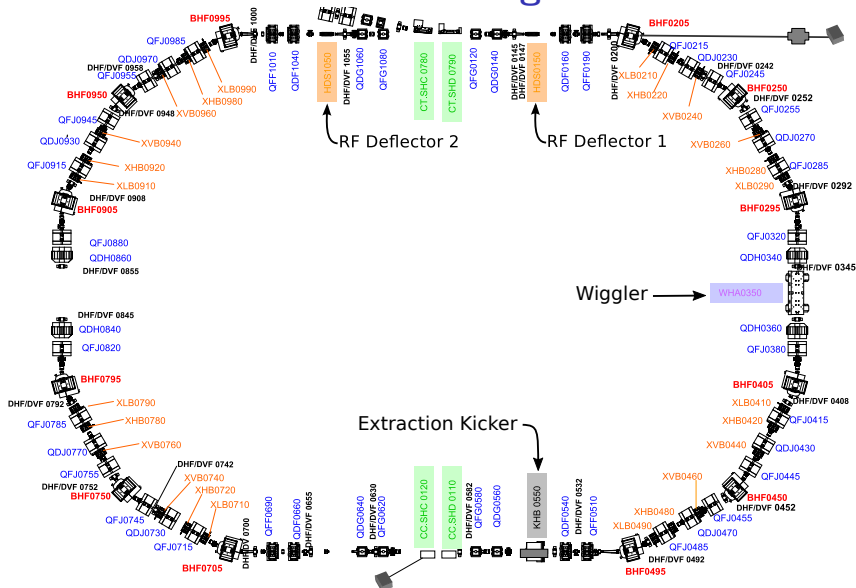


- Advancing joints can easily be made in parallel!
- Possibility to slice a beamline adding joints in the middle.

Applications to CTF3

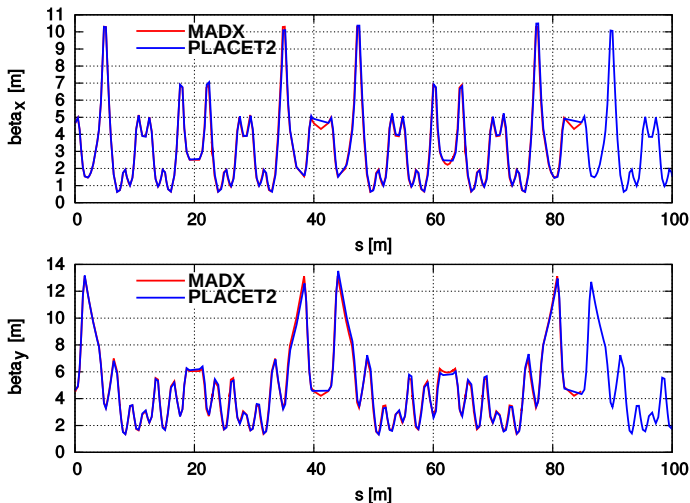
CLIC Test Facility 3

The Combiner Ring Lattice



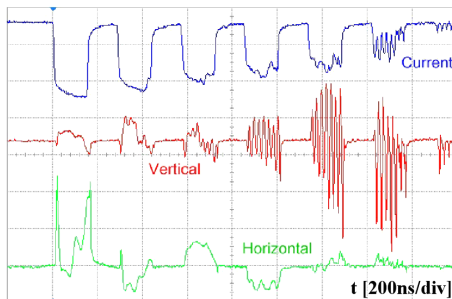
One turn Beta Functions

Optics parameters are collected following a test bunch along its path in the machine. Good agreement with MAD-X computation.



Wakefields in the CTF3 RF deflectors

During the commissioning of the combiner ring, an unforeseen vertical instability appeared, limiting the operability to short trains and low charge. It was later eliminated with a new design of the deflectors.

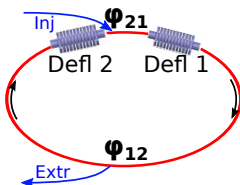


A circulating (not recombined) beam of 400 bunches is subjected to heavy losses due to the excitation of the trapped vertical mode:

| | |
|-----------|----------------|
| Frequency | 3.0443 GHz |
| Impedance | 1.6 M Ω |
| Q-value | 11500 |

A dedicated tracking code was written by D. Alesini to simulate the excitation¹, it uses few parameters (beta, phase advance, ...) while PLACET2 has the full model of the combiner ring.

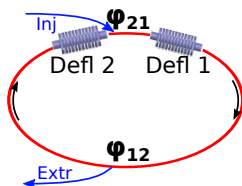
Vertical Instability: Phase Scan



The circulating beam establishes a feedback whose sign depends on:

- *Betatron phase advance,*
- Mode Frequency,
- Time of flight.

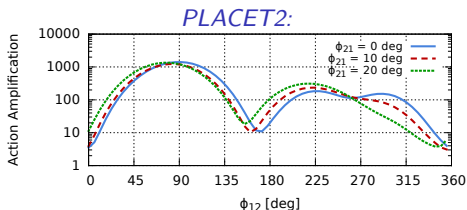
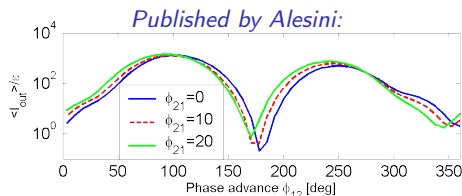
Vertical Instability: Phase Scan



The circulating beam establishes a feedback whose sign depends on:

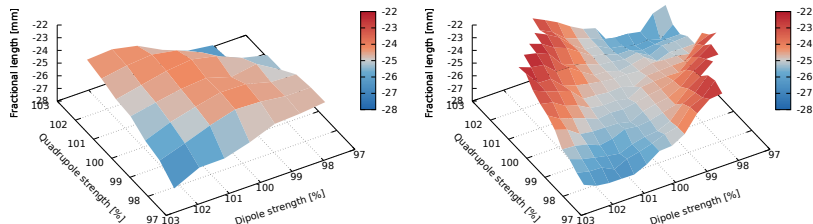
- *Betatron phase advance,*
- Mode Frequency,
- Time of flight.

Track 400 bunches with 2 mm initial offset for 4 turns. Look at the average action of the train.



Beam based length measure @ CTF3 CR

- A mismatched ring length has repercussions on the efficiency of the deceleration;
- Wanted to investigate the impact of optics errors/detuning on the ring length;
- Measures done with a phase monitor (no RF cavity in the ring).



Measured data (left) compared to simulated phase monitor output from PLACET2 (right) processed with the same signal analysis algorithms.

- Demonstrated the possibility to make the ring shorter.
- Not yet understood the vanishing T_{566} in the ring.

Summary

- Overview of the capability and mechanisms of PLACET2:
 - element slicing,
 - connection of beamlines,
 - synchronization of multibunch tracking,
 - shaping of instrumentation signals (BPR).
- Applications to the CTF3 Combiner Ring:
 - multiturn tracking,
 - long range wakefields instability,
 - measure and simulation of the length response to optics detuning.

Summary

- Overview of the capability and mechanisms of PLACET2:
 - element slicing,
 - connection of beamlines,
 - synchronization of multibunch tracking,
 - shaping of instrumentation signals (BPR).
- Applications to the CTF3 Combiner Ring:
 - multiturn tracking,
 - long range wakefields instability,
 - measure and simulation of the length response to optics detuning.

Thank you!