Recent Improvements in RF-Track

Andrea Latina
(CERN)

Table of Contents

RF-Track

- RF-Track is a fast, parallel C++ library:
  - User interface through
    - Octave
    - Python

- *Minimalistic* and *physics-oriented*: it relies on two robust and renowned open-source libraries for *"all the rest"*
  - GSL, "Gnu Scientific Library", provides a wide range of mathematical routines such as high-quality random number generators, ODE integrators, linear algebra, and much more
  - FFTW, "Fastest Fourier Transform in the West", arguably the fastest open-source library to compute discrete Fourier transforms

RF-Track user interface

RF-Track is a library, loadable from

- Octave, *a high-level language for numerical computations, mostly compatible with Matlab, open-source*
- Python, *general-purpose, high-level programming language, open-source*

Both languages offer powerful toolboxes for numerical experimentation: multidimensional optimisations, fitting routines, data analysis, control tools, ...

Example (Octave interface)

```
% load RF-Track
RF_Track;

% setup simulation
TL = setup_transferline();
B0 = create_bunch();

% track
B1 = TL.track(B0);

% inquire the phase space
T1 = B1.get_phase_space("%x %xp %y %yp");

% plot
plot(T1(:,1), T1(:,2), "*");
xlabel("x [mm]");
ylabel("x' [mrad]");
```

Tracking: Two beam models

1. Beam moving in space:
   - All particles have the same S position
   - Each particle's phase space is

     $$\left(x \text{ [mm]},\ x' \text{ [mrad]},\ y \text{ [mm]},\ y' \text{ [mrad]},\ t \text{ [mm/c]},\ P \text{ [MeV/c]}\right)$$

   - Integrates the equations of motion in dS:

     $$S \rightarrow S + dS$$

2. Beam moving in time:
   - All particles are taken at same time t
   - Each particle's phase space is

     $$\left(X \text{ [mm]},\ Y \text{ [mm]},\ S \text{ [mm]},\ P_x \text{ [MeV/c]},\ P_y \text{ [MeV/c]},\ P_z \text{ [MeV/c]}\right)$$

   - Handles particles with $P_z < 0$ or even $P_z = 0$, particles can move backward
   - Integrates the equations of motion in dt:

     $$t \rightarrow t + dt$$

- Each particle also stores

  $$m: \text{ mass [MeV/c}^2\text{]},\quad Q: \text{ charge } [e^+]$$

  $$N: \text{ nb of particles / macroparticle},\quad t_0: \text{ creation time}^{(\star)}$$

- RF-Track can simulate mixed-specie beams, particle's creation (including field emission)

$^{(\star)}$ only for beams of type 2

Tracking: Integration algorithms

- The default: "**leapfrog**": very fast, second-order, symplectic
- Higher-order, explicit, algorithms:
  - ⋆"**rk2**" Runge-Kutta (2, 3)
  - ⋆"**rk4**" 4th order (classical) Runge-Kutta
  - ⋆"**rkf45**" Runge-Kutta-Fehlberg (4, 5)
  - ⋆"**rkck**" Runge-Kutta Cash-Karp (4, 5)
  - ⋆"**rk8pd**" Runge-Kutta Prince-Dormand (8, 9)
  - ⋆"**msadams**" multistep Adams in Nordsieck form; order varies dynamically between 1 and 12
- Higher-order, implicit, algorithms:
  - ⋆"**rk1imp**", "**rk2imp**", "**rk4imp**" implicit Runge-Kutta
  - ⋆"**bsimp**" Bulirsch-Stoer method of Bader and Deuflhard
  - ⋆"**msbdf**" multistep backward differentiation formula (BDF) method in Nordsieck form
- Analytic algorithm:
  - ⋆"**analytic**" integration of the equations of motion in a locally constant EM field

Example:

```
L = Lattice();
L.append(RFQ);
L.set_odeint_algorithm("rkf45");
B1 = L.track(B0);
```

3D Space-charge: P2P and PIC

RF-Track solves the basic differential laws of magneto and electro-statics for any distribution of particles. Full 3-D solver, with two optional methods:

1. particle-2-particle:
   - computes the electromagnetic interaction between each pair of particles
   - numerically-stable summation of the forces (Kahan summation)
   - fully parallel
2. 3D particle-in-cell code: $\rightarrow$ fast
   - uses 3-D Integrated Green functions
   - computes E and B fields directly from ϕ and $\vec{A}$ (this ensures $\nabla \cdot \vec{B} = 0$)
   - can save E and B field maps on file, and use them for fast tracking
   - implements continuous beams
   - fully parallel

- No approximations such as "small velocities", or $\vec{B} \ll \vec{E}$, or rigid gaussian bunch, are made.
  - Can simulate beam-beam forces

Simulation of injectors

- An injector:
  1. It's composed by a relatively small number of elements, and it's not periodic
  2. If features very rapid dynamics, from non-relativistic to highly relativistic energies
  3. Space-charge is relevant across elements
  4. Magnetic elements are over-imposed on RF elements: e.g. solenoid coils around RF fields

- A "lattice-based" description of the injector is not really suitable. "Lattices" work well for long, periodic, sequences of elements (i.e. for tracking in ΔS, where the whole beam is moved along, element by element)

- A new environment dedicated to tracking in Δt was created: the Volume

Three new features of RF-Track

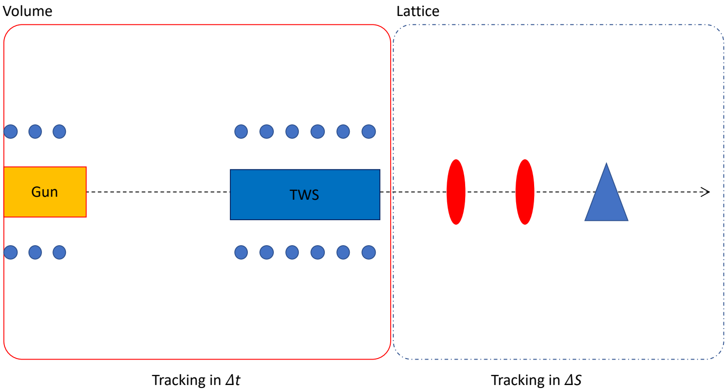
1. New environment "Volume", for complex tracking in Δt
2. Space-charge effects from mirror charges on cathode
3. New elements: 1D Fields, Coil, SW structure, TW structure

A new environment: Volume

Dedicated to space-charge dominated regions.

1. It's a 3D portion of space that can host an unlimited number of elements, among all those offered by RF-Track (e.g. RF field maps, static field maps, quadrupoles, etc.)
2. The elements can be placed at *any arbitrary location*: given position X, Y, Z and Euler angles: pitch, roll, yaw (no small-angle approximation)
3. Full *overlap between elements is possible*
4. It tracks in Δt, using *a large number of integration algorithms* (from order 1 to 12, adaptive) w/space-charge (3D PIC)
5. Tracking is performed starting from any initial particles' phase space, until all particles have left the Volume
6. Particle can be created at arbitrary times and positions, with any energy, mass, and charge
7. Full diagnostics is possible during tracking: e.g. tracking the emittances, or saving the beam on disk at constant time intervals

Volume and Lattice



(SC-dominated regimes)

Example: `Volume.add(Element, X, Y, Z, roll, pitch, yaw, "reference");`

- X,Y,Z: arbitrary position in the 3D space
- roll, pitch, yaw: Euler angles
- reference point: "center", "entrance", "exit"

New element: 1D RF field map

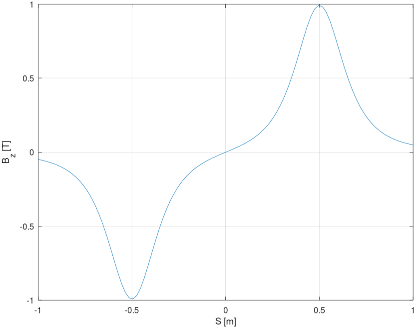
- 3D field maps
  - Tri-linear / tri-cubic interpolation
  - Rectangular / Cylindrical mesh
  - Automatic mirroring of the field map
  - Dynamically change the input Power
  - Consider Not-A-Number(s) as walls
- **1D on-axis field maps**:
  - E_z: linear / cubic interpolation
  - From E_z on axis reconstructs E_r and B_ϕ in the 3D volume (3$^{\text{rd}}$-order Taylor expansion)

New element: Coil

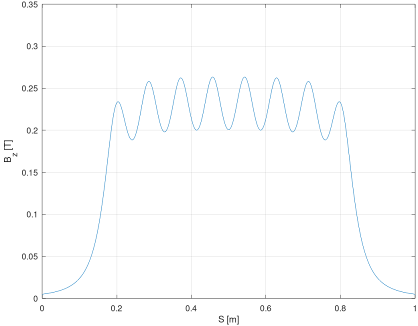
Analytic coil equations. The coil is specified given radius and current (or alternatively radius and max field). Its field extend to the whole Volume (with fringe fields)

Two examples:

2 coils:



8 coils:



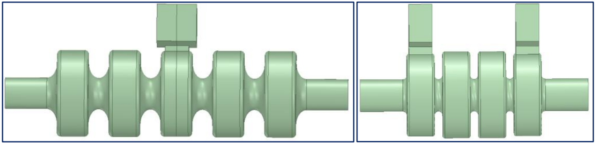
Example 1: (left) On-axis field of two coils, located respectively at $S = -0.5$ m and $S = 0.5$ m, carrying an equal electric current which flows in opposite directions. The volume extends from -1 to 1 m.

Example 2: (right) On-axis field of a sequence of 8 coils to form a solenoid-like magnetic field.

Example of Octave script

```
%% Load RF-Track
RF_Track;

%% Declare two coils
Cm = Coil(0.01, -1.0, 0.2);   % L length [m],
                              % B field at the center of the coil [T],
                              % R radius [m]
Cp = Coil(0.01, +1.0, 0.2);

%% Create a Volume
V = Volume();

% Add the two coils
V.add(Cm, 0, 0, -0.5);
V.add(Cp, 0, 0, 0.5);

% Set the boundaries
V.set_s0(-1.0); % -1 m
V.set_s1(+1.0); % +1 m
```

New elements: SW and TW structures



HFSS model of a metallic RF cavity. Left panel: Standing-Wave cavity. Right panel: Traveling-Wave cavity.

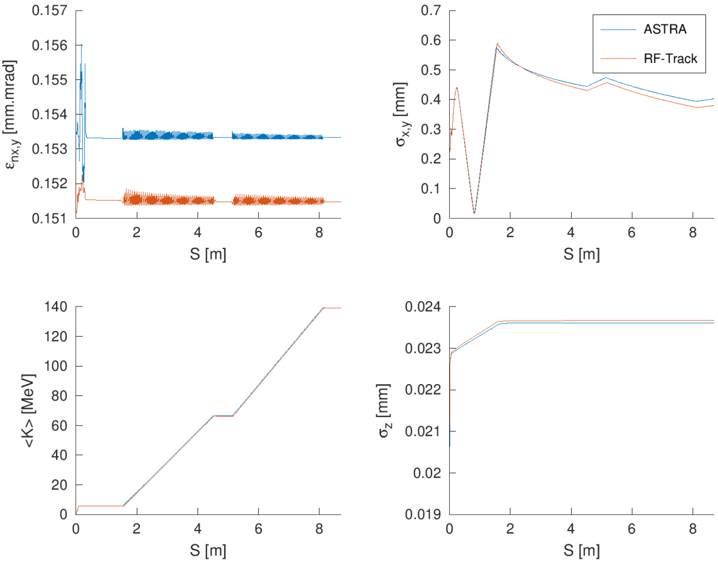
RF-Track takes the Fourier coefficients (any order) and creates a SW or TW structure. It reconstructs the $\vec{E}$ and $\vec{B}$ fields in the full 3D space.

New elements: SW and TW structures

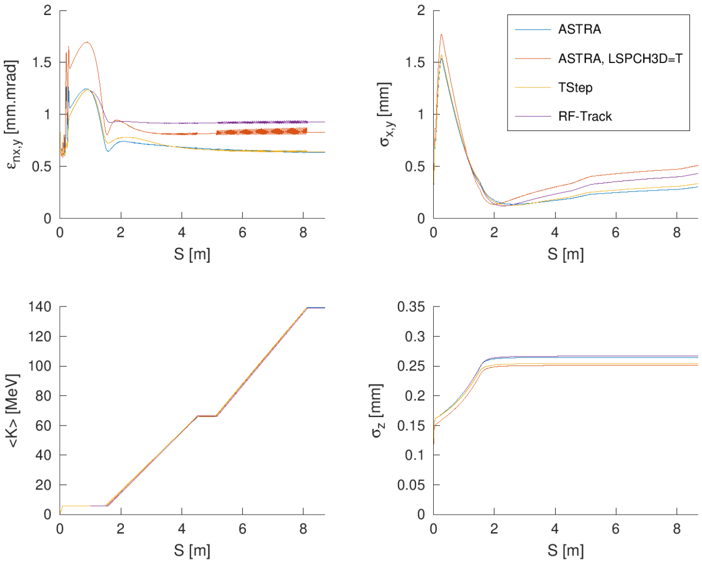
Example:

```
%% define Fourier coefficients
a0 = 1.0; % TW structure, principal Fourier coefficient, V/m
a1 = 1.0; % SW structure, principal Fourier coefficient, V/m

% structure params
freq = 2.856e9; % Hz, frequency
lambda = clight / freq; % m, wavelength
ph_adv = 2*pi/3; % rad, phase advance per cell
l_sw_cell = 0.020; % m, length of the SW full cell

%% number of cells
% a negative sign indicates a start from the beginning of the cell
% a positive sign indicates a start from the middle of the cell

% define the accelerating structure
AS = Lattice();
AS.append(SW_Structure(a1, freq, l_sw_cell, -0.5)); % half SW, entrance coupler
AS.append(TW_Structure(a0, 0, freq, ph_adv, +3.0)); % TW part, 3 cells
AS.append(SW_Structure(a1, freq, l_sw_cell, +0.5)); % half SW, exit coupler

% place the AS in the volume
V = Volume();
V.add(AS, 0, 0, 0);
```

D3.2: S-band injector (noSC)



D3.2: S-band injector (SC)



Conclusions

- RF-Track has recently undergone some improvements to make it suitable for lijector guns simulations
- Advantages
  1. it's open and free
  2. it's parallel and fast
  3. it's developed in house
  4. it's flexible and quite powerful
- `https://gitlab.cern.ch/alatina/rf-track-2.0`