ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB203

UPDATE OF THE TRACKING CODE RF-Track

A. Latina, CERN, Geneva, Switzerland

Abstract

During the last couple of years, the RF-Track particle tracking code has seen a tremendous increase in the number of its applications: medical linacs, compact injector electron guns, and positron sources are among the main ones. Following a work of consolidation of its internal structure, new simulation capabilities have been introduced, together with several new effects: arbitrary orientation of elements in space, full element overlap, short- and long-range wakefields, and laser-beam interaction through Compton scattering are the most significant ones. In this paper, some of these new features are presented and discussed.

INTRODUCTION

RF-Track is a tracking code developed at CERN during the last few years for the optimization of particle accelerators. It is written in optimized C++ and uses the scripting languages Octave [1] and Python [2] as user interfaces.

RF-Track [3] was created as a tool dedicated to perform tracking simulations of a medical linac for hadron therapy, featuring backward traveling wave structures [4], then it evolved to a multi-purpose accelerator toolbox capable of handling a large number of challenging simulation scenarios. Now, RF-Track can simulate beams of particles with arbitrary energy, mass, and charge, even mixed. It includes particles creation, and it can transport beams through common elements such as dipoles and quadrupoles, as well as through "special" ones: 1D, 2D, and 3D static or oscillating radio-frequency electromagnetic field maps (real and complex), flux concentrators for positron sources, and electron coolers.

RF-Track implements two different particle tracking methods: tracking *in time* and tracking *in space*. The tracking *in time* is preferable in space-charge-dominated regimes, where the relative positions of the particles in space matters; the tracking *in space* suits better space-charge-free regions where particles are independent of each other and they can be transported simultaneously from the entrance of an element to its end, element by element.

TRACKING ENVIRONMENTS

The two tracking methods mentioned above were initially implemented to be interchangeable. This was a useful excercise, but brought no added value from having two models. One of the major recent developments was the creation of two distinct tracking environments to fully profit from the specificity of each model: the `Lattice` and the `Volume` environments.

The environment `Lattice` represents the accelerator as a plain list of elements and performs tracking *in space*, transporting the beam, element by element, from the entrance to the exit plane of each element, sequentially.

The environment `Volume` can simulate elements with arbitrary position and orientation in the three-dimensional space. It provides more flexibility than `Lattice`, as it allows elements to overlap. `Volume` performs tracking in *in time* which is more suitable for space-charge calculations, and enables particle creation at any time and at any location, which enables the simulation of cathodes, field emission, and dark currents. Furthermore, in a `Volume`, particles can propagate in any direction (even backwards). So that particles of the same bunch can happen to occupy different elements simultaneously.

Several "special" elements, available only in `Volume`, allow taking full advantage of the flexibility of `Volume`: e.g., analytic coils and solenoids, where the magnetic field is computed from 3D analytic formulæ and permeates the whole 3D space including fully realistic fringe fields. The possibility to overlap elements makes `Volume` perfect for the simulation of, e.g., injectors, where a solenoidal magnetic field usually surrounds the accelerating field of the gun, and the effects of space-charge are critical.

Example of Volume Creation

The following lines show an example of Volume creation in Octave and how to add elements. The method `add()` of the object `Volume()` allows one to add an element to a volume. In its fullest extent, this method takes up to 8 input arguments:

```
V.add (element, Xpos, Ypos, Zpos, ...
       roll, pitch, yaw, refer='entrance') ,
```

where `Xpos`, `Ypos`, and `Zpos` are the coordinates of the element in meters; `roll`, `pitch`, and `yaw` are the orientation angles in radians (respectively around the Z, the X, and the Y axis); and `refer` specifies the intended reference point, respectively "entrance", "center", or "exit". Notice that no "small-angles" approximation is made.

The example below shows how three elements of a compact source are placed in a volume. The three elements are: an injector gun, a solenoid magnet, and an alpha magnet, which needs to be tilted by 40.71 degrees with respect to the beam axis in order to be effective:

```
% Create a Volume
V = Volume();

% add the injector gun
V.add (Gun, 0, 0, 0); % at the origin of axes

% add a solenoid 0.2 m from the cathode
V.add (Sol, 0, 0, 0.2, 'center');

% add an alpha magnet 0.5 m from the cathode
```

```
V.add (Alpha_Mag, ...
     0, 0, 0.5, ... %
     0, 0, deg2rad(40.71)); % yaw angle
```

Elements Overlap: Example of RF Beam Loading

Another good example of elements overlap, is the simulation of beam-loading in RF structures. Beam-loading fields are RF fields traveling in phase with the beam, which have a decelerating effect while the beam travels through a structure.

The beam-loading field overlaps with the accelerating field, but operates at a different RF phase, since the accelerating field operates at an arbitrary phase dictated by the operational requirements, whereas the beam loading field is in phase with the bunch. The simulation of this effect requires the two fields to be acting on the beam at different phases simultaneously. The overlap of these two field maps, which is computed automatically at run-time, allows one to dynamically change the RF phase of the accelerating field, while preserving the effect of the beam loaded field.

Figure 1 shows the accelerating field experienced by a particle while it travels through a traveling wave structure sustaining the effects of beam loading (top), as well as the "unloaded" field (bottom).

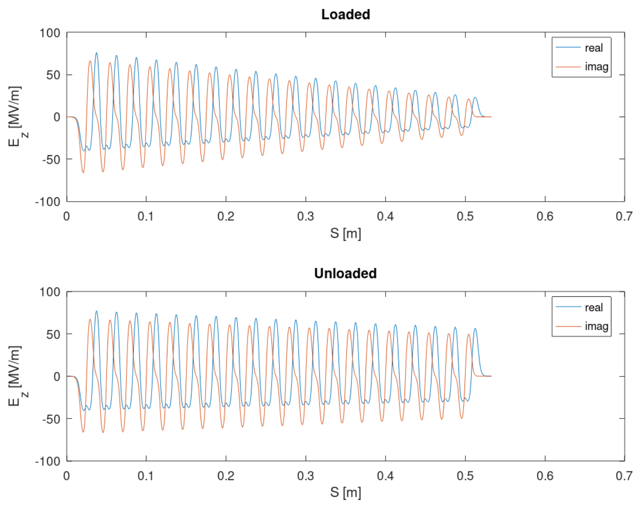

Figure 1: (top) "Loaded" field as the summation of the unloaded plus beam-loading fields; (bottom) "unloaded" field, the field traveling through the unloaded structure.

Remarks

It should be noticed that also the environment Lattice can take into account space-charge effects. However, since Bunch6d maintains the distribution of the particles on the same longitudinal plane, the calculation of space-charge forces needs an on-the-fly extrapolation of each particle's longitudinal position, using the different arrival times and the velocities of each particle, to reconstruct the three-dimensional distribution.

Because of this somehow artificial manipulation of the phase space, we judge Lattice as most suitable for space-charge-free regions of the accelerator. Volume, which through Bunch6dT maintains the full spatial distribution of the beam, should be the preferred choice for space-charge-dominated regimes.

In conclusion, Lattice is straightforward and fast, but better limited to space-charge-free regions. On the other hand, Volume is extremely flexible and rich, but this additional flexibility comes at the cost of being computationally more expensive.

INVERSE COMPTON SCATTERING

For the optimization of light sources based on Compton scattering between an electron beam and a laser, a new element implementing the beam-laser interaction was implemented, the LaserBeam. This element constructor allows the user to fully specify the parameters of the laser beam.

Here we provide an example based on the numbers reported in the Technical Design Report of the ThomX project at LAL, Orsay [5]. The following lines illustrate how to set up the laser beam interaction region. First, it is useful to define some key parameters:

```
% Beam-laser interaction point (IP)
L_FP = 1; % m, length of the IP region
IP_Lstar = L_FP/2; % m, position of the IP
X_angle = 2; % deg, crossing angle
L_dir = ... % laser direction
    [ sind(180-X_angle), 0, cosd(180-X_angle) ];
```

L_dir is an oriented 3D vector indicating the propagation axis of the laser beam. In this case, the laser beam moves towards the electron beam, traveling nearly parallel to the Z axis, but in the opposite direction, with a 2-degree crossing angle in the XZ plane (notice the cosine of 180 deg – *crossing angle* along the Z axis). It should be noted that *any* laser orientation in the 3D space is possible. The code will take care of performing the appropriate Lorentz boosts to simulate the scattering process.

The following lines show the creation of the proper particle beam-laser interaction region:

```
% Create a laser-beam IP region
FP = LaserBeam(); % Fabry-Perot resonator
FP.pulse_energy = 28; % mJ, laser pulse energy
FP.pulse_length = 5; % ps, laser pulse length
FP.wavelength = 1030; % nm, laser wavelength
FP.set_direction(L_dir); % 2 deg crossing angle
FP.length = L_FP; % m, element length
FP.set_IP_position(L_FP / 2.0); % m
FP.R = 0.035; % mm, laser beam radius at waist
FP.zR = 15; % mm, Rayleigh length
FP.min_number_of_gammas_per_slice = 10;
FP.set_nslices(20);
```

The implementation takes into account fully relativistic transformations of the reference system, and computes the interaction choosing automatically between the Thompson cross section and the Klein–Nishina cross section, depending on the particle's and photon's energies.

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-THPAB203

Notice that the two cross sections have been implemented in all their generality, and no assumption is made on the scattering particle being an electron. The simulation can compute Compton scattering between proton or ion beams against photons.

Figure 2 shows the result of a simulation reproducing the ThomX parameters. The numbers are perfectly matching those in the ThomX Technical Design Report.

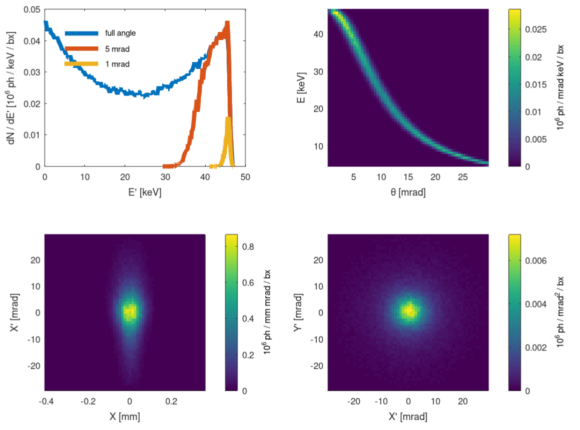

Figure 2: Scatter plots and photon spectrum of one bunch crossing of a ThomX-like electron beam-laser interaction, as computed with RF-Track using the new Inverse-Compton scattering module.

WAKEFIELDS

A recent breakthrough in the RF-Track development consisted in the possibility to distribute arbitrary kicks of "collective effects" over both the environments `Lattice` and `Volume`. The implementation, which is fully general, enables the distribution of kicks along the beam line regardless of their nature. The first effects that we implemented are wakefields. More effects, like coherent synchrotron radiation, will be added in the future.

The simulation of wakefields is crucial whether one considers short-range wakefields effects or long-range (bunch-to-bunch) effects. Three distinct models of wakefields have been implemented, to provide full flexibility:

```
SR = ShortRange_Wakefield (a, g, l);
LR = LongRange_Wakefield (f, A, Q);
WF = Wakefield_1d (W_transv, W_long, hz);
```

The first one is based on the K. Bane's approximation [6], where a, g, and l are respectively the iris aperture radius, the gap length, and the cell length in meters. The second model allows the simulation of long-range effects; it takes the frequency f in Hz, the amplitude A in V/pC/mm/m, and the Q values of an array of modes. Each input is in fact a vector quantity. The third model takes directly two one-dimensional meshes, $W_{\text{transverse}}$ and $W_{\text{longitudinal}}$, to sample the wakefunction at equally spaced point located h_z meters from each other; $W_{\text{transverse}}$ is expressed in V/pC/mm/m, while $W_{\text{longitudinal}}$ is in V/pC/m. This model can be used to input *arbitrary* wakefield functions, for instance: long-range multi-bunch wakefields, or resistive wall wakes.

These effects can be added to an element using the method: `add_collective_effect()`. For instance:

```
a = 3.2; % mm, iris aperture radius
l = 8.332012; % mm, cell length
g = l - 1.5; % mm, gap length

% Define the wake
SR = ShortRange_Wakefield (a/1e3, g/1e3, l/1e3);

freq = 12e9; % GHz, X-band structure
TW = RF_FieldMap (Ex, Ey, Ez, hx, hy, hz, freq);
TW.add_collective_effect (SR);
```

CONCLUSIONS

In this paper, we have presented new features of the tracking code RF-Track which opened new important simulation scenarios: Inverse Compton-Scattering light sources, dynamic optimizations of beam performance including beam loading and wakefields effects, arbitrary overlay and orientation of elements in space. Together with the flexibility offered by its interface towards Octave and Python, these new features open a vast range of complex simulation scenarios that can lead to robust and highly optimized start-to-end performance. The code is acquiring acknowledgment and recognition, as it is witnessed by the growing number of studies made using it and publications.

REFERENCES

[1] *GNU Octave version 5.2.0 manual: a high-level interactive language for numerical computations*, J. W. Eaton, Boston, MA, USA, 2020, pp. 1-1121, `https://octave.org/octave.pdf`

[2] *Python reference manual*, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands, 1995, `https://www.python.org`

[3] *RF-Track Reference Manual* CERN, Geneva, Switzerland, 2020, `https://zenodo.org/record/3887085#.YOySPuhKjIU`

[4] S. Benedetti, "High-gradient and high-efficiency linear accelerators for hadron therapy," Ph.D. Thesis, Faculté Des Sciences De Base, École Polytechnique Fédérale De Lausanne, Lausanne, Suisse, 2018.

[5] A. Variola, J. Haissinski, A. Loulergue, and F. Zomer, "ThomX Technical Design Report," HAL archives-ouvertes, Online, Rep. in2p3-00971281, Apr. 2014.

[6] K. Bane, "Short range dipole wakefields in accelerating structures for the NLC," SLAC, Menlo Park, CA, USA, Rep. SLAC-PUB-9663, Mar. 2003.