RF-TRACK: BEAM TRACKING IN FIELD MAPS INCLUDING SPACE-CHARGE EFFECTS. FEATURES AND BENCHMARKS

A. Latina, CERN, Geneva, Switzerland

Abstract

RF-Track is a novel tracking code developed at CERN for the optimization of low-energy ion linacs in presence of space-charge effects. RF-Track features great flexibility and rapid simulation speed. It can transport beams of particles with arbitrary mass and charge even mixed together, solving fully relativistic equations of motion. It implements direct space-charge effects in a physically consistent manner, using parallel algorithms. It can simulate bunched beams as well as continuous ones, and transport through conventional elements as well as through maps of oscillating radio-frequency fields. RF-Track is written in optimized and parallel C++, and it uses the scripting languages Octave and Python as user interfaces. RF-Track has been tested successfully in several cases. The main features of the code and the results of its benchmark studies are presented in this paper.

INTRODUCTION

RF-Track was developed to optimize the design and beam transport of the TULIP backward traveling-wave linac [1,2]. The main requirements were three: (1) being able to track particles in backward-traveling radio-frequency (rf) field maps; (2) being able to transport protons as well as light ions in a fully relativistic regime (β-relativistic, in TULIP, is ≈ 0.38); and (3) being able to dynamically tune the rf parameters, like e.g. the rf input power, in order to perform non-trivial optimizations of the linac's transport efficiency.

Given the limited number of codes capable of tracking in oscillating electric and magnetic field maps, the uncertainty on how these codes would handle field maps of backward-traveling structures, and the requirement of a dynamically tunable rf input power, it was decided to develop a new *ad hoc* tool, optimized and tailored for the TULIP project.

RF-Track fulfilled the requirements, and eventually grew to become a general-purpose tracking code that excels for its flexibility, accuracy, and simulation capabilities. Its main features are:

- it is fully relativistic: doesn't make any approximation such as $\beta \ll 1$ or $\gamma \gg 1$;
- it can track particles of arbitrary mass and charge, even in mixed-species beams;
- it implements direct space-charge interaction, computing both the electric and the magnetic fields acting within the particles;
- it implements several integration algorithms: fast algorithms for complex nonlinear optimizations, accurate-but-slow ones for precise tracking;
- it is fast, fully benefiting from modern multi-core CPUs;
- it is programmable, relying on powerful and expressive scripting languages like Octave and Python for its user interface.

The following sections will elucidate each of these points.

RF-TRACK INTERNALS

RF-Track has been developed in C++11, fully exploiting the multi-thread capabilities offered by this language. Every single algorithm in RF-Track has been designed to take full advantage of modern multi-core CPUs.

In an effort aimed at making RF-Track a minimalistic code, yet uncompromised in its scientific throughput, the development has been focused on all physics-related algorithms, relying on powerful and well-established numerical libraries for *all the rest*. Two libraries were chosen to provide numerical algorithms: GSL, the "Gnu Scientific Library", which offers a wide range of mathematical routines such as random number generators, ODE integrators, linear algebra, and more [3]; and FFTW, the "Fastest Fourier Transform in the West", probably the fastest opensource library to compute discrete Fourier transforms ever written [4].

The hundreds of functions and routines that constitute RF-Track are compiled into a single binary file dynamically loadable from the two scripting languages: Octave [5] and Python [6]. These powerful high-level languages are ideal for numerical and scientific experimentations. They offer a large number of *off-the-shelf* toolboxes to perform complex numerical tasks: e.g., multidimensional optimizations, nonlinear fits, complex data processing, etc. The accelerator physics capabilities embedded in RF-Track, together with these expressive and rich scientific languages, make the simulation possibilities offered by RF-Track virtually uncountable.

The interface between the internal C++ code and the aforementioned scripting languages has been obtained using SWIG [7]. A typical RF-Track script, in its Octave version, looks like this:

```
% load the RF-Track library
RF_Track;

% setup the simulation, e.g. a transfer line TL and a beam B0
TL = setup_a_transferline();
B0 = setup_a_beam();

% track B0 through TL, and store the result as B1
B1 = TL.track(B0);

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

% use Octave's plotting routines to display the results
plot(T1(:,1), T1(:,2), "*");
xlabel("x [mm]");
ylabel("x' [mrad]");
```

As shown, RF-Track's commands can be interleaved with Octave keywords.

BEAM MODELS

Internally, RF-Track represents the beam as an ensemble of macroparticles. It evolves the beam along the accelerator solving the equations of motion according to two optional beam models:

Beam Moving in Space

All particles lie on a thin sheet at the same longitudinal position S; tracking is performed integrating the equations of motion in dS: that is, S goes to $S + dS$ (typically, element-by-element); each particle's phase space is internally stored as the six-dimensional vector:

$$(x,\ x',\ y,\ y',\ t,\ P_z)$$

where t is the proper time of each particle. This is available to the user as type `Bunch6d`.

Beam Moving in Time

The particles coordinates are kept as a six-dimensional snapshot taken at the same time t; tracking is performed integrating the equations of motion in dt: that is, t goes to $t + dt$; each particle's phase space is internally stored as the six-dimensional vector:

$$\left(X,\ Y,\ S,\ P_x,\ P_y,\ P_z\right)$$

where X, Y, and S are the 3d spacial coordinates inside the accelerator. One might notice that this beam model allows to handle particles with $P_z < 0$ (moving backward) as well as $P_z = 0$ (pure transverse motion). This is available to the user as type `Bunch6dT`.

In both cases, for each macroparticle, RF-Track also stores: m, the particle mass in MeV/c^2; Q, the electric charge in units of e; and N, the number of particles per macroparticle. This allows RF-Track to simulate mixed-species beams as well as zero-current particles (i.e. ideal witness particles: that bear a charge, $Q \neq 0$, but bear no current, $N = 0$).

Great care has been given to granting the user the maximum flexibility in accessing the beam information. Both the objects `Bunch6d` and `Bunch6dT` implement a method called `get_phase_space()`, which allows to inquire the phase space from many different viewpoints. This is shown in the previous example already, where the string `"%x %xp %y %yp"` was meant to have RF-Track return the beam's phase space as a matrix with 4 columns: x positions, x' angles, y positions, and y' angles.

RF-Track doesn't impose a specific convention, and other %-identifiers include, for example: `"%Px"`, `"%Py"`, `"%Pz"`, the total momenta expressed in MeV/c ; `"%Vx"`, `"%Vy"`, `"%Vz"`, the velocities in units of the speed of light, c; `"%E"` the total energy and `"%K"` the kinetic energy, both in MeV, such that one can retrieve the beam information in great detail and with a direct physical grip. The following example shows how to access the beam information miming three well-established accelerator codes:

```
% Accessing the phase space MAD-X's style
T = B1.get_phase_space("%x %px %y %py %Z %pt");

% TRANSPORT's style
T = B1.get_phase_space("%x %xp %y %yp %dt %d");

% PLACET's style
T = B1.get_phase_space("%E %x %y %dt %xp %yp");
```

This follows the object-oriented paradigm of data encapsulation: the user can access the full information in a transparent and intuitive way, without needing to care about the internal representation of the data.

INTEGRATION ALGORITHMS

Great care has been given to the routines for solving the equations of motion. RF-Track offers more than a dozen algorithms, that can be categorized in three groups:

1. "leapfrog", a second-order integration method of symplectic nature. It is extremely fast, although a large number of integration steps is required to achieve great accuracy;
2. a battery of 12 algorithms imported from GSL, which offers a variety of low-level methods such as Runge-Kutta and Bulirsch-Stoer routines, as well as higher-level components for adaptive step-size control like the Nordsieck method (accurate to the 12th order) [3]. These algorithms are identified by labels like "rk2", "rk4", "rkf45", "rk4imp", "rk5imp", "msadams"; "msbdf", etc. They are generally slow, but offer great accuracy;
3. "analytic" integration, where the equations of motion are solved analytically assuming a constant field during one integration step (e.g. for a particle in a constant magnetic field, the 3d helical trajectory is calculated). Very useful insights on how to integrate analytically the equations of motion in a combined electric and magnetic field were found in [8]. This method is the most accurate among all methods, it is symplectic, and it is reasonably fast.

The algorithms "leapfrog" and "analytic" are original implementations in RF-Track; the others are imported from GSL. The choice of integration algorithm can be made at run-time, very easily, as shown in this example:

```
% load the fieldmap of an RFQ
RFQ = load_rfq_field_map();

% select the integration algorithm (see text)
RFQ.set_odeint_algorithm("analytic");

% tracks B0 in time, using time step dt = 1 mm/c
B1 = RFQ.track(B0, 1.0);
```

where a beam is transported through the field map of an RFQ using the "analytic" integration algorithm.

SPACE-CHARGE

RF-Track solves the differential laws of magneto and electro-statics to compute the electromagnetic forces acting within the beam. It computes the full 3d electric and the magnetic interaction using two independent methods:

Particle-to-Particle

It computes the electromagnetic interaction between each pair of particles, computing both the electric and the magnetic components of the force; it uses the Kahan summation algorithm to provide a numerically-stable summation of the forces; it is fully parallel, with complexity scaling as $O(n^2_{\mathrm{particles}}\ /n_{\mathrm{CPUs}})$.

Cloud-in-Cell

Computes the electric and the magnetic fields solving the Maxwell equations for the scalar and the vector potentials, using a FFT method; it uses 3d integrated Green functions for computing the scalar and the vector potentials, and 5th-order derivatives to compute the fields (error $O\left(h^4\right)$). It can save the $\vec{E}$ and $\vec{B}$ field maps on disk and use them later for fast tracking; it implements *continuous* beams using a modified Green function; it is fully parallel, with complexity scaling as $O(n_{\mathrm{particles}} \cdot n_{\mathrm{grid}} \cdot \log n_{\mathrm{grid}}\ /n_{\mathrm{CPUs}})$ computations. Obviously this is much faster then the particle-to-particle method.

It must be noted that no approximations such as "small transverse velocities", or $\vec{B} \ll \vec{E}$, or gaussian bunch distribution, are made. Furthermore, as these algorithms compute also the B field, it can simulate beam-beam forces.

BENCHMARKS

RF-Track has been benchmarked against other codes in a large range of cases, always finding excellent agreement. We report here of three such cases.

ELENA Transfer Line

Antiprotons with kinetic energy E_{kinetic}=100 keV ($\beta_{\mathrm{rel}} \approx$ 0.015), transported through a transfer line with 6 FODO cells. The comparison against PTC shows perfect agreement on a particle-to-particle level, showing that RF-Track and PTC implement similar transfer maps for the most common accelerator elements (see Fig. 1).

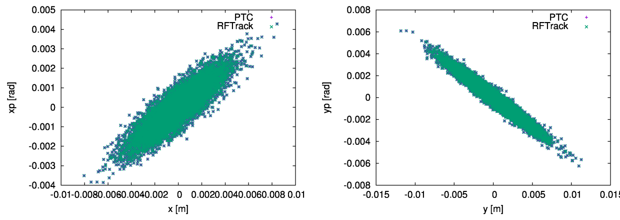

Figure 1: Antiprotons transported through a transfer line with 6 FODO cells.

CERN's 750 MHz RFQ

The final distribution matched with very good agreement the results obtained using the code PATH. The plots show the horizontal and longitudinal phase spaces (see Fig. 2).

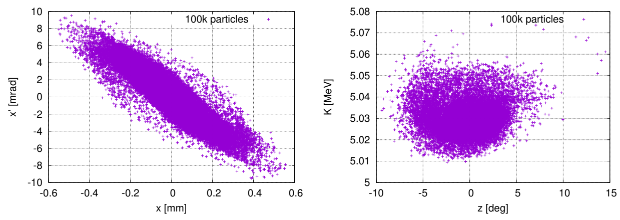

Figure 2: Tracking of 100'000 particles in the field map of the CERN's 750 MHz RFQ.

Lead Ion Source for Linac 3

The distribution contains oxygen ions from O^{1+} to O^{8+}, and lead ions from Pb^{21+} to Pb^{36+}. The plots show the exit x-y plane for lead ions with charge state Q = 29+. Left-hand and right-hand plots show the result without and with space-charge, respectively. Excellent agreement has been found against the original IBSimu simulations (see Fig. 3).

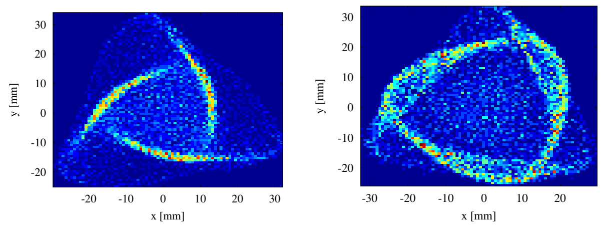

Figure 3: Tracking of an IBSimu-generated input distribution through the complex field map of the CERN's Linac 3 ion source.

TULIP Project

The results of the simulations obtained with RF-Track in the context of the TULIP project are documented in [2] (presented in this conference).

SUMMARY AND OUTLOOK

A new code with a great potential for a large range of application has been created: RF-Track. It implements accurate tracking and fast space-charge solvers. Future developments foreseen include the simulation of electron cooling and indirect space-charge. To receive further information, contact the author.

ACKNOWLEDGMENTS

The author wishes to thank all those who have helped performing the benchmarks, those who use RF-Track providing useful feedbacks, and those who have encouraged its development: Alessandra Lombardi, Veliko Atanasov Dimov, Stefano Benedetti, Marc Maintrot, Ville Toivanen, Elias Metral, and Roberto Corsini.

REFERENCES

[1] A. Degiovanni *et al.*, "Design of a Fast-Cycling High-Gradient Rotating Linac for Protontherapy", in *Proc. IPAC'13*, paper THPWA008.

[2] S. Benedetti *et al.*, "Design of a 750 MHZ IH Structure for Medial Applications", presented at LINAC'16, East Lansing, MI, USA, paper MOPLR049, this conference.

[3] M. Galassi *et al.*, GNU Scientific Library Reference Manual (3rd Ed.), ISBN 0954612078, `http://www.gnu.org/software/gsl`

[4] M. Frigo and S. G. Johnson, FFTW user's manual, MIT Press, May 1999, `http://www.fftw.org`

[5] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations, 2015, `http://www.gnu.org/software/octave`

[6] G. van Rossum, Python tutorial, Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[7] David M. Beazley, "SWIG: an easy to use tool for integrating scripting languages with C and C++", TCLTK'96 Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4.

[8] D. Hestenes, "New Foundations for Classical Mechanics", Kluwer Academic Publishers, 2nd ed. 1999.