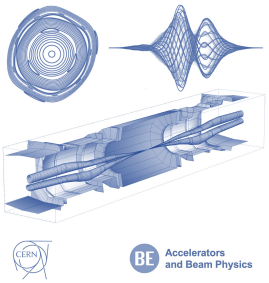


RF-Track Test Suite Development and Applications

Lale Balat

CERN Summer Student Programme 2025

Supervisors: Andrea Latina, Paula Desire Valdor

September 2025

Abstract

This project presents the RF-Track test suite, designed to ensure the reliability and consistency of the RF-Track software for accelerator and beam dynamics simulations. The suite saves accurate reference files and compares them with new results to detect bugs and unintended changes. A total of 59 components were covered with tests implemented in Python and Octave, together with reference files in `.h5` , representing roughly 70% of RF-Track's modules.. At present, all 59 tests pass successfully, confirming the stability of the framework. This work therefore provides a solid foundation for future improvements of RF-Track while safeguarding against unnoticed faults.
The full test suite is available online at RF-Track Test Suite [1].
Keywords: RF-Track, test suite, beam dynamics, accelerator physics, simulations, numerical methods, computational tools, Python, Octave

Contents

1. Introduction

RF-Track, a particle tracking code developed at CERN, provides a flexible and accurate framework for beam dynamics studies. In accelerator science, simulation software is essential because it complements theory and experiments, providing a practical way to study accelerator behaviour [2]. With RF-Track, users can observe the effects of changing parameters, design complete lattices, and analyse quantities such as phase space, beam distributions, and collective effects using both realistic field maps and conventional beamline elements.

While simulation tools are valuable for understanding the operation of current accelerators such as the LHC, they are also essential for the design of future colliders. The main projects beyond the LHC include the FCC (Future Circular Collider), CLIC (Compact Linear Collider), and the Muon Collider. Briefly, the FCC and Muon Collider are circular colliders, whereas CLIC is designed as a linear collider.

The FCC is planned in two stages: FCC-ee, an electron–positron collider for high-precision studies, followed by FCC-hh, a proton–proton collider with heavy-ion capability. With a circumference of 91 km, synchrotron radiation losses will be reduced compared to smaller circular accelerators. RF-Track has already been applied to FCC-ee injectors [3], providing reliable simulations for optimisation.

The CLIC project is based on high-gradient RF cavities and a two-beam acceleration scheme, reaching fields of about 100MV/m. It aims to collide electrons and positrons at centre-of-mass energies up to 3TeV, while keeping the facility compact and cost-effective. Being linear, CLIC avoids synchrotron radiation losses. RF-Track has been used in CLIC optimisation studies [4], showing its value for quick and simplified simulations.

These features make RF-Track well suited for high-intensity accelerator studies. After CLIC, it is worth noting that RF-Track has also been used in the design of medical machines, especially electron linacs for medical applications.

Muon colliders promise collisions of point-like particles at very high energies, without the severe synchrotron radiation losses suffered by electrons in circular machines. Since muons are about 200 times heavier than electrons, they can be accelerated efficiently in rings. One of the main challenges is ionisation cooling, and RF-Track provides an effective tool for optimising such channels in future muon collider designs [5].

In summary, simulations are essential both to understand existing accelerators and to design future ones. In the following of this report, I will first describe the main accelerator

components and the basic beam dynamics principles (Chapter 2). Then, I will introduce the main focus of my project, the RF-Track test suite (Chapter 3), which is the framework on which this work is based.

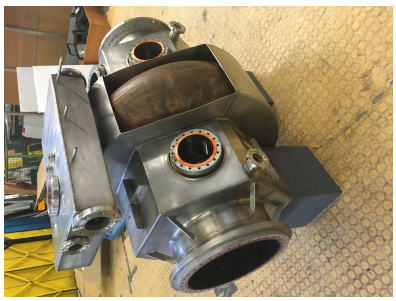
2. Accelerator Components

Accelerators are devices designed to increase the energy of microscopic particles under the influence of electromagnetic fields. This influence manifests itself in two fundamental ways: longitudinal acceleration provided by electric fields, and transverse bending or focusing of trajectories caused by electric and magnetic fields.

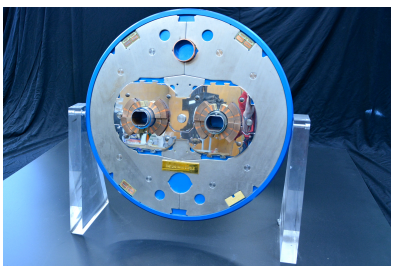
The motion of charged particles in an accelerator is governed by the relativistic extension of Newton's laws together with the Lorentz force, expressed as

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \,.$$

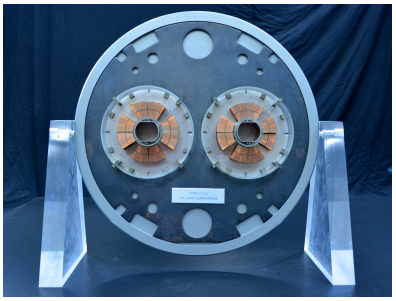
The transport of particles takes place within an environment known as the **lattice**, which is composed of all accelerator components arranged in sequence from the entrance to the exit plane of each element. In order to achieve efficient acceleration, the lattice is carefully designed by placing these elements in the proper order. The main components of the lattice are RF cavities, which provide acceleration through oscillating electromagnetic fields, and magnets, which guide, steer, and focus the particles using magnetic fields.



CERN, *LHC accelerating cavity prototype*, CERN-OBJ-AC-064.



CERN, *Slice through an LHC bending magnet*, CERN-OBJ-AC-041.



CERN, *Slice through an LHC focusing magnet*, CERN-OBJ-AC-042.

Figure 2.1: Examples of key LHC components: RF cavity, dipole, and quadrupole magnets.

2.1 RF Cavities

An essential aspect of particle acceleration is the use of RF cavities. In fact, RF cavities operate on the principle of an oscillating polarization, meaning that the direction of the electromagnetic field continuously changes. By means of these oscillating fields,

which are sustained by the polarized surfaces of the cavity walls, it becomes possible to ensure that energy is effectively transferred to charged particles, and as a result, the particles are accelerated.

Moreover, a travelling-wave RF cavity requires a supply of input power, since the RF energy is partly dissipated in the cavity itself and partly delivered to the beam. In addition, RF cavities are not only used for acceleration but also for bunching, debunching, and more general forms.

2.1.1 Plasma Acceleration

Another method to accelerate particles is plasma acceleration. This technique relies on plasma waves, also referred to as wakefields, and such accelerators have demonstrated remarkable promise because they can sustain accelerating electric fields up to a thousand times stronger than those in conventional RF accelerators. This capability opens the door to developing much more compact and cost-effective accelerator structures. Although significant progress has already been made toward designing new accelerators for future colliders, research in this field remains very much ongoing.

2.2 Magnets

In addition to RF cavities, another essential component of particle accelerators is magnets. Magnets are crucial for controlling and guiding charged particles through their magnetic fields. They play a key role in keeping the beams stable and precisely aligned. The specific control they provide depends on the magnet type: some magnets steer or bend the beam, while others focus or shape it, allowing the trajectories to be directed with high precision.

2.2.1 Dipole Magnets

Among the most common types of magnets, dipole magnets are used to curve and guide particles along the desired path. They can be implemented either as sector-bending magnets, which follow a curved reference trajectory, or as rectangular bending magnets, whose entrance and exit faces are parallel.

2.2.2 Quadrupole Magnets

In addition to dipoles, quadrupole magnets—which consist of four poles—are employed to focus and defocus the beam in the horizontal and vertical planes. They function in a way similar to optical lenses, effectively "squeezing" the beam to maintain its shape. They control the beam size by providing transverse focusing. A lattice cell consisting of alter-

nating focusing and defocusing quadrupoles is known as a FODO cell, which represents one of the most widely used methods for beam shaping.

2.2.3 Sextupole Magnets

In contrast, sextupole magnets have six poles, and their magnetic field varies quadratically with the distance from the magnet center. Their primary function is to correct beam distortions at the edges of phase space, much like an optical lens corrects chromatic aberrations.

2.2.4 Superconducting Magnets in the LHC

Therefore, all of the magnets in the LHC are electromagnets, which means that their magnetic fields are produced by the flow of electric current. The Large Hadron Collider is not only the largest cryogenic system in the world but also one of the coldest places on Earth. Its main magnets operate at a temperature of 1.9 K ($-271.3°$C), which is colder than the 2.7 K ($-270.5°$C) background temperature of outer space. Cryogenic techniques are thus essential to cool and maintain the superconducting magnets.

However, superconductors carry the risk of losing their superconducting state, a process known as a quench, during which they stop operating correctly. To mitigate such risks, recent research at CERN has focused on developing magnets of the future from high-temperature superconductors (HTS). HTS technology offers an extended operating temperature range and a larger safety margin, making it a promising path for the next generation of accelerator magnets. These developments are expected to play a central role in the design of future high-energy accelerators.

2.3 Other Components

In addition to the standard elements already discussed, modern accelerator lattices may also include a wide range of specialized components. Examples are correctors, used for fine steering adjustments; and coils or solenoids that provide longitudinal magnetic fields. RF structures can take different forms, such as the pillbox cavity, standing-wave and traveling-wave structures, and undulators for generating radiation. More advanced configurations include wakefield structures, chicanes for beam compression, photoinjectors, and implementations of inverse Compton scattering. Finally, there are also passive sections such as drifts, which simply represent free spaces without electric or magnetic fields.

2.4 Collective Effects

Apart from single-particle dynamics, accelerators must also deal with **collective effects**, which arise from interactions within the beam or with the surrounding structures. The most relevant collective effects in high-density accelerators are:

- **Synchrotron radiation** – charged particles moving on curved paths emit radiation, which causes energy loss and emittance damping. This is a problem for circular colliders, but it can be avoided in linear accelerators (LINAC). At the same time, this radiation is very useful in synchrotron light sources such as ALBA, ESRF, and PETRA, where it is collected for experiments.
- **Space charge** – dominant at low energies and high intensities, caused by Coulomb repulsion within the bunch. The accompanying magnetic attraction partly compensates for the electric repulsion, but the net effect can still result in emittance growth and an increase of the effective electric field.
- **Wakefields** – electromagnetic fields left behind by a beam as it travels through accelerating structures, which in turn act back on subsequent particles and bunches.
- **Beam–beam interactions** – strong electromagnetic interactions that occur when two intense beams collide, significantly affecting the dynamics in colliders. (Related plot 3.7)

3. RF-Track Test Suite

3.1 Motivation

The RF-Track test suite is designed to detect bugs and to capture any unintended changes in the code. When a new version of RF-Track is released, the suite verifies whether the program still works as expected by comparing the output with predefined reference files. During the development of the test suite itself, several bugs were also discovered and fixed, which shows its usefulness in improving the software. Thanks to this framework, users can clearly see the effects of parameter variations, and they are free to test new configurations without risking hidden changes in the core system. In short, the test suite both protects RF-Track from unnoticed modifications and provides a safe environment for exploring new ideas.

3.2 Implementation

3.2.1 General Structure

The test suite was developed for both Python and Octave. Each test follows the same structure and consists of three main files with identical names but different extensions: `.m`, `.py`, and `.h5`. To keep the structure simple and user-friendly, every component contains these three files, and in some cases an additional folder with the same name, such as `test_component`. Each test is implemented as an individual function that can either create a reference file (with the option `make_ref = true`) or compare the current output with the reference (default mode, `make_ref = false`). When reference files are generated, plots and detailed output are shown, whereas in testing mode only a short "All Tests PASSED Successfully!" message is printed.

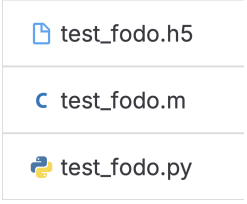


Figure 3.1: Folder structure: `.m`, `.py`, and `.h5` extensions for each.

3.2.2 Reference Files

The reference files are stored in `.h5` format in order to use a single standard for both Octave and Python. All data are saved in the same structure, so it does not matter whether the file was created with Octave or Python. These reference files serve as the baseline for checking whether new outputs match the expected results.

Initially, the reference data were stored in `.mat` format. Subsequently, an open-source, more universal and flexible solution was adopted. In order to guarantee full compatibility between Octave and Python, dedicated `save_h5` and `load_h5` functions were implemented. Thus, all reference files were converted to the Hierarchical Data Format version 5 (HDF5, `.h5`), which provides a widely used open standard in scientific computing. This ensures that the reference system remains consistent and scientifically reliable across different platforms.

3.2.3 Automation

On the Python side, the automation is handled by `pytest`, which allows running all tests in one step. In Octave, an equivalent mechanism is implemented, where `make_ref = true` generates reference files and `make_ref = false` performs the test. To simplify this process, a script called `runall_oct.sh` was added to automate the execution of all Octave tests. By default, the `false` mode is chosen to protect the reference files from being overwritten.

3.2.4 Comparison and Test Logic

To ensure consistency, the comparison is based on the `numpy.allclose()` function in Python. For Octave, an equivalent function called `allclose.m` was implemented separately, so that both environments follow the same numerical validation. The typical tolerance values used are `atol = 1e-6` and `rtol = 0.0`. This choice means that only absolute deviations are considered, while relative errors are ignored. The reason for this setting is that the tests are designed to be fully deterministic, so any difference in results should not depend on the scale of the numbers but only on small numerical rounding effects. In addition, backtracking tests compare the initial and final bunch distributions, verifying that particles can be consistently transported backward through the lattice.

Despite this general application, in the Compton source test the values can become very large, so even a small percentage difference results in a large absolute deviation. For this reason a relative tolerance (`rtol = 2e-2`) is applied, allowing small variations that scale with the magnitude of the numbers.

3.2.5 Stabilizing Randomness

When creating reference files, random data generation is controlled by fixing the seed:

```
rng_set("mt19937");
rng_set_seed(42);
```

This ensures that pseudo-random distributions are reproduced exactly across different runs. In addition, for certain tests the computational methods are explicitly fixed, using numerical algorithms such as `leapfrog` or `rk2` to generate consistent reference data.

Also, in some tests such as `test_particle_losses` and `test_absorber_lattice` quasi-random sequences were used to create the bunch. Quasi-random, also called low-discrepancy sequences, are not truly random but distribute points more evenly across space. Because of this property, they can provide better accuracy than purely random numbers, and in these tests they allow the creation of more balanced bunches.

3.3 Script Structure

The structures of the Octave and Python tests are shown side by side. Most parts remain almost identical, with the same reference–handling logic and deterministic random seed. This ensures that results can be directly compared while avoiding artificial differences between the two implementations. (Full version is in the Appendix).

```python
ref_file="test_fodo.h5"
ref_data = {}

def test_fodo(make_ref=False):
    #Determine the seed
    rft.rng_set('mt19937')
    rft.rng_set_seed(42)

    ## Bunch parameters, setup the
        elements and lattice, define the
         beam using the twiss parameters
        , create the bunch, perform
        tracking

    if make_ref:
        # Save the reference data
        ref_data['T'] = T
        ref_data['M'] = M
        #Plots
    else:
       # Load reference data
        ref = load_h5(ref_file)
        T_ref = ref['T']
        M_ref = ref['M']
        #Compare the outputs and
            references
        assert np.allclose(T, T_ref,
            rtol=0.0, atol=1e-6)
        assert np.allclose(M, M_ref,
            rtol=0.0, atol=1e-6)
    if make_ref:
        save_h5(ref_file,ref_data)
if __name__ == '__main__':
    test_fodo(make_ref=True)
```

Listing 3.1: Python structure

```octave
function test_fodo(make_ref)
    RF_Track;
    if nargin == 0
        make_ref = false;
    end
    % Reference data directory
    ref_file = 'test_fodo.h5';
    Results= struct();
    %Determine the seed
    rng_set('mt19937');
    rng_set_seed(42);
    % Bunch parameters, setup the
        elements and lattice, define the
         beam using the twiss parameters
        , create the bunch, perform
        tracking
    if make_ref
    %% Save the reference data
        Results.T = T;
        Results.M = M;
        %% Plots
    else
        % Load reference data
        ref = load_h5(ref_file);
        T_ref= ref.T;
        M_ref= ref.M;
        assert(allclose(T, T_ref,0, 1e
            -6));
        assert(allclose(M, M_ref,0, 1e
            -6));
        fprintf('
            All_Tests_PASSED_Successfully
            !');
    end
    if make_ref
        save_h5(ref_file,Results);
    end
    if ~make_ref
        if ~isempty(get(0, 'Children'))
            close all;
        end
    end
end
%% Octave test blocks
%!test
%! test_fodo(false);
```

Listing 3.2: Octave structure

3.4 Test Suite Overview

While saving the reference data, RF-Track displays several plots and outputs. With these, users and developers can check whether their scripts and results are correct. In contrast, during the test runs no plots or outputs are shown, since the purpose of the RF-Track test suite is to perform the checks quickly and clearly, without any additional information.

In this project, 59 different test titles were created. Among them, 44 are individual tests, while others are grouped: some correspond to backtracking, some to lattice, and some to volume. In addition, 7 tests include both volume and lattice cases, and 7 files combine forward and backtracking tests.

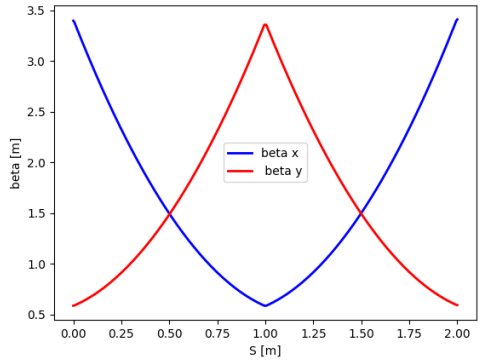
From the beginning, 59 Python test files were created, always considering that Octave versions should exist in parallel. Altogether, 111 coding files (Python and Octave) were written. By default, 59 reference files in `.h5` format were also generated as a consequence of creating the reference data.

3.5 Examples

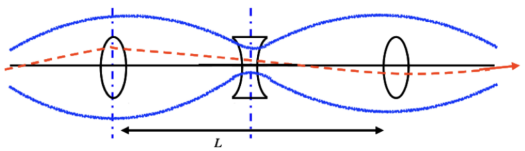
Here are some important examples from RF-Track plots:

3.5.1 test_fodo

A FODO lattice basically consists of one focusing quadrupole (F), one defocusing quadrupole (D), and two drift spaces (O) located between the two quadrupoles. In this case, the FODO cell is designed without the thin-lens approximation, since the length of the quadrupoles is not zero. To keep the beam size constant after each cell, a numerical method is used to minimize a merit function, which provides the matching conditions.



(a) Beam envelope evolution in a FODO lattice simulated with RF-Track.



(b) Schematic drawing of a FODO cell [6].

3.5.2 test_backtracking_quadrupole

The quadrupole, introduced in Chapter 2, is one of the main types of magnets. Here we show a visualisation of the phase space of a particle bunch in a quadrupole, together with the corresponding backtracking result. As seen in the backtracking plot, the initial and final bunch distributions overlap, indicating that the transport is correctly reversed.

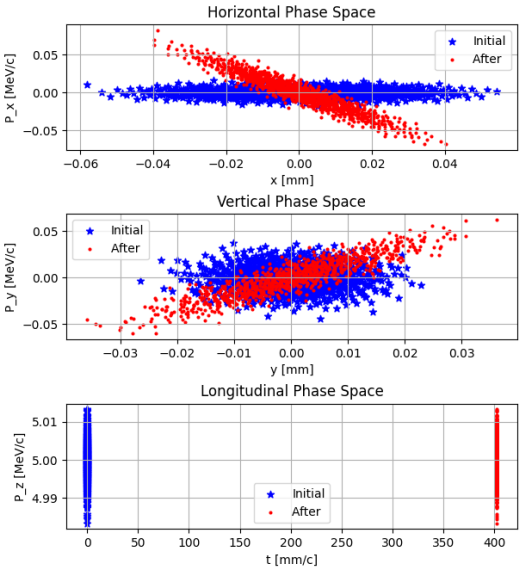


Figure 3.3: Phase space of a bunch tracked through a quadrupole.

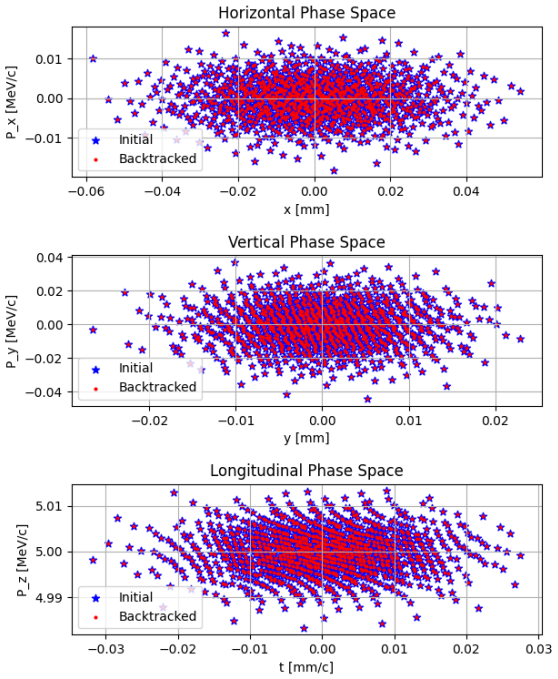


Figure 3.4: Backtracking result: initial and final bunch distributions coincide.

3.5.3 test_particle_losses_lattice

This test demonstrates particle losses in a collimator. A uniform square beam is transported through a drift and a rotated collimator. Particles outside the aperture are lost, as illustrated below.

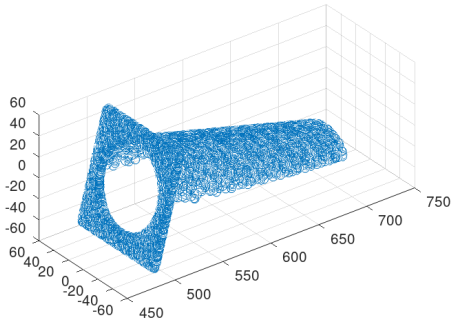


Figure 3.5: Lost particle distribution after the collimator.

3.5.4 test_wakefield_lattice

Wakefields arise from the interaction of a bunch with its own electromagnetic fields. Radiation generated by the head of the bunch acts back on the tail, distorting its regular trajectory.

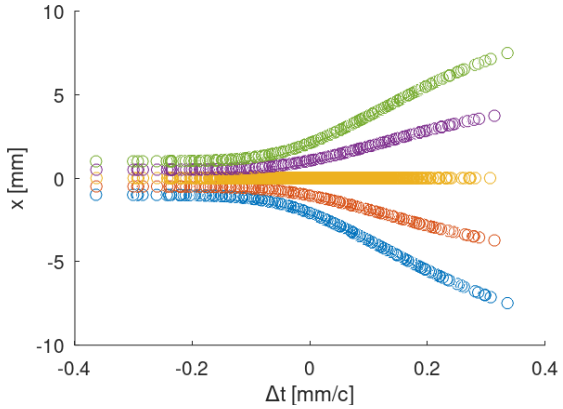
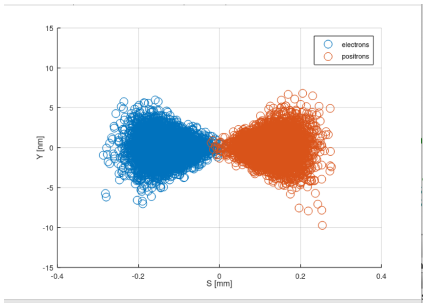


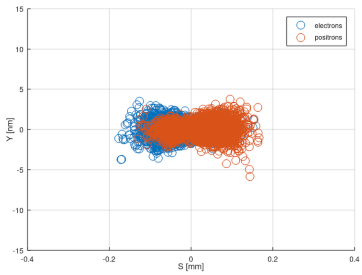
Figure 3.6: Wakefield effects on the bunch inside the lattice.

3.5.5 test_beam_beam

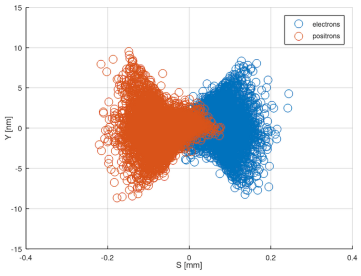
Beam–beam effects arise when two counter-propagating charged beams interact with each other. This interaction can cause instabilities in the bunches due to strong electromagnetic forces. The plots below show beam–beam interactions between an electron and a positron beam, where distortions of the bunches can be clearly observed.



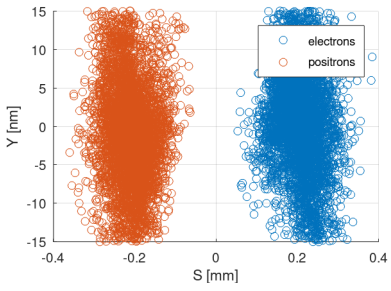
(a) Stage 1



(b) Stage 2



(c) Stage 3



(d) Stage 4

Figure 3.7: Beam–beam simulation result of electron–positron interactions.

4. Discussion and Conclusions

4.1 Discussion & Future Work

During the development of the RF-Track test suite, all files were systematically tested. In this process, several bugs were identified and corrected, even while the suite was still under construction. As a result, the newer versions of RF-Track were released in a more stable and reliable way, showing that the test suite is an effective tool to ensure consistency and correctness.

The tests generally use absolute tolerance values instead of relative tolerance, and to ensure consistency, a fixed random seed was applied together with numerical computing methods. Looking ahead, the test suite should be maintained in step with RF-Track development. When new modules or features are introduced, corresponding tests should be added as well. This ensures that the suite remains consistent with the code and helps preserve long-term reliability by preventing unnoticed changes.

4.2 Conclusion

To sum up, RF-Track is a well-qualified framework for accelerator and beam dynamics simulations. However, such a framework also requires a dedicated test suite to remain consistent with its reference data files. This project established the RF-Track test suite from the beginning and provided a foundation for its further development. Most of the modules of RF-Track now have their own tests implemented in both Python and Octave versions. So, this work represents a first step to keep RF-Track reliable and to make it easier to improve in the future.

5. Acknowledgements

I would first like to express my deepest appreciation to my supervisor, Andrea Latina, for his constant support and guidance throughout this project. His motivating and encouraging attitude always kept me focused, and his collaboration made me feel that the work I was doing was meaningful and impactful.

I am equally grateful to my co-supervisor, Paula Desire Valdor, for her close guidance and continuous dedication. Through our regular studies, she helped me to grasp the fundamental concepts of accelerators and beam dynamics by focusing on the most essential aspects. Beyond the technical work, she enriched my experience at CERN by guiding me through projects, inviting me to events, and making me feel truly part of the community. I feel very fortunate for the invaluable support of both of my supervisors, who ensured that I never felt lost or without purpose, and who helped me maximise this experience.

Finally, I would like to sincerely thank my professors, Dr. Bora Akgün and Prof. Dr. Serkant Ali Çetin, for their trust, support, and invaluable references. Their continuous communication and belief in me have been a great source of encouragement throughout this journey.

6. Appendix

Octave test script for "test_fodo.m"

```
function test_fodo(make_ref)
    RF_Track;

    % Set default mode
    if nargin == 0
        make_ref = false;
    end

    % Reference data directory
    ref_file = 'test_fodo.h5';
    Results= struct();

    Part.mass = RF_Track.electronmass; % MeV/c^2
    Part.P = 100; % MeV/c
    Part.Q = -1; % e+
    B_rho = Part.P / Part.Q; % MV/c, reference rigidity

    Lcell = 2; % m
    Lquad = 0.01; % m
    mu = 90; % deg
    k1L = sind(mu/2) / (Lcell/4); % 1/m
    strength = k1L * B_rho; % MeV/m
    % gradient = strength / Lquad * 1e6 / RF_Track.clight; % T/m = V/m/m
       /c

    rng_set('mt19937');
    rng_set_seed(42);

    Qf = Quadrupole(Lquad/2,  strength/2);
    Qd = Quadrupole(Lquad, -strength);
    D = Drift(Lcell/2 - Lquad);
    D.set_tt_nsteps(100);

    FODOcell = Lattice();
    FODOcell.append(Qf);
    FODOcell.append(D);
```

```
    FODOcell.append(Qd);
    FODOcell.append(D);
    FODOcell.append(Qf);

    Nparticles = 1000;
    Twiss = Bunch6d_twiss();
    Twiss.emitt_x = 1; % mm.mrad normalised emittance
    Twiss.emitt_y = 1; % mm.mrad
    Twiss.beta_x = Lcell * (1 + sind(mu/2)) / sind(mu); % m
    Twiss.beta_y = Lcell * (1 - sind(mu/2)) / sind(mu); % m
    Twiss.alpha_x = 0;
    Twiss.alpha_y = 0;

    B0 = Bunch6d_QR(Part.mass, 1e10, Part.Q, Part.P, Twiss, Nparticles);

    B1 = FODOcell.track(B0);
    T = FODOcell.get_transport_table('%S␣%beta_x␣%beta_y');
    M = B1.get_phase_space();

    %% Reference data handling
    if make_ref
        Results.T = T;
        Results.M = M;

        %% Plots
        figure(1)
        clf
        plot(T(:,1), T(:,2), 'b-', T(:,1), T(:,3), 'r-');
        legend('beta␣x', 'beta␣y');
        xlabel('S␣[m]');
        ylabel('beta␣[m]');

        figure(2)
        clf
        hold on
        scatter(B0.get_phase_space('%x'), B0.get_phase_space('%xp'));
        scatter(B1.get_phase_space('%x'), B1.get_phase_space('%xp'), 'r'
            );
        title('initial␣and␣final␣horizontal␣phase␣space');
        xlabel('x␣[mm]');
        ylabel('P_x␣[MeV/c]');

        figure(3)
        clf
        hold on
        scatter(B0.get_phase_space('%y'), B0.get_phase_space('%yp'));
        scatter(B1.get_phase_space('%y'), B1.get_phase_space('%yp'), 'r'
            );
```

```
82          title('initial␣and␣final␣vertical␣phase␣space');
83          xlabel('y␣[mm]');
84          ylabel('P_y␣[MeV/c]');
85
86      else
87          % Load reference data
88          ref = load_h5(ref_file);
89
90          T_ref= ref.T;
91          M_ref= ref.M;
92
93          assert(allclose(T, T_ref, 0, 1e-6));
94          assert(allclose(M, M_ref, 0, 1e-6));
95
96          fprintf('\n---␣All␣Tests␣PASSED␣Successfully!␣---\n');
97      end
98      if make_ref
99          save_h5(ref_file, Results);
100         disp('All␣reference␣data␣saved.');
101     end
102
103     %% Close figures in test mode
104     if ~make_ref
105         if ~isempty(get(0, 'Children'))
106             close all;
107         end
108     end
109 end
110
111 %% Octave test blocks
112 %!test
113 %! test_fodo(false);
```

Source functions to save and load HDF5 format

```python
import h5py, numpy as np

def save_h5(filename, ref):
    with h5py.File(filename, "w") as f:
        for k, v in ref.items():
            a = np.asarray(v)
            if a.ndim == 0:             # scalar -> 1x1
                a = a.reshape(1, 1)
            elif a.ndim == 1:           # 1D -> Nx1
                a = a.reshape(-1, 1)
            elif a.ndim == 2:           # 2D -> transpose
                a = a.T
            elif a.ndim > 2:
                raise ValueError(f"{k}: only up to 2D supported")
            a = a.astype(np.float64, copy=False)
            #Act like octave save function
            g = f.create_group(str(k))
            g.create_dataset('type', data=np.array(b'double', dtype='S6'), dtype='S6')
            g.create_dataset('value', data=a, dtype='float64', compression=None, shuffle=False, chunks=None)

def load_h5(fname):
    out = {}
    with h5py.File(fname, "r") as f:
        for k in f.keys():
            arr = np.array(f[k]['value'])
            if arr.ndim == 2:
                arr = arr.T             # (rows, cols)
            elif arr.ndim == 1:
                arr = arr[None, :]      # 1xN (row)
            else:
                arr = arr.reshape(1, 1)
            out[k] = arr
    return out
```

Listing 1: Python source functions for HDF5

```octave
function save_h5(filename, ref)
  save('-hdf5', filename, '-struct', 'ref');
end

function out = load_h5(fname)
  s = load("-hdf5", fname);
  out = struct();
  f = fieldnames(s);
  for i = 1:numel(f)
    k = f{i};
    A = s.(k);
    if isstruct(A) && isfield(A, "value"), A = A.value; end         % T. value -> T
    base = regexprep(k, '_value$', '');                             % T_value -> T
    out.(base) = A;
  end
end
```

Listing 2: Octave source functions for HDF5

Bibliography

[1] CERN. "RF-Track Test Suite. "[Online]. Available: https://gitlab.cern.ch/rf-track/rf-track-test-suite.

[2] A. Latina, *RF-Track Reference Manual*, CERN, 2025. [Online]. Available: https://gitlab.cern.ch/rf-track/rf-track-reference-manual.

[3] A. Latina, "RF-Track Development for the FCC-ee Injectors," in *FCC Week*, Vienna, Austria, May 2025. [Online]. Available: https://indico.cern.ch/event/1408515/contributions/6515053/attachments/3072119/5438886/Latina_RFTrack.pdf.

[4] Y. Zhao and A. Latina, "Optimization of the compact linear collider rings-to-main-linac at 380 GeV," *Physical Review Accelerators and Beams*, vol. 28, p. 021 003, 2025. DOI: 10.1103/PhysRevAccelBeams.28.021003.

[5] B. Stechauner, E. Fol, R. Frühwirth, A. Latina, C. Rogers, D. Schulte, and J. Schieck, "Final cooling simulations for a muon collider," in *IMCC & MuCol Annual Meeting*, CERN, Geneva, Mar. 2024. [Online]. Available: https://indico.cern.ch/event/1325963/contributions/5793008/.

[6] B. Holzer, "Transverse beam dynamics," in *CERN Yellow Reports: Monographs, CERN-2024-003*, Geneva, Switzerland: CERN, 2024. DOI: 10.23731/CYRM-2024-003.115. [Online]. Available: https://inspirehep.net/literature/2744283.

[7] M. Conte and W. W. MacKay, *An Introduction to the Physics of Particle Accelerators*. World Scientific, 1991, ISBN: 9789810205729.

[8] A. Wolski, *Beam Dynamics in High Energy Particle Accelerators*. Imperial College Press, 2014, ISBN: 9781783262836.

[9] A. Latina, "The Tracking Code RF-Track v2.4.1," Conference presentation, 2025. [Online]. Available: https://indico.cern.ch/event/1533557/contributions/6452646/attachments/3069094/5429189/RFTrack_presentation.pdf.

[10] E. Jensen, "RF Cavity Design," in *Proceedings of the CAS-CERN Accelerator School: Advanced Accelerator Physics*, Trondheim, Norway: CERN, 2014, pp. 405–429. [Online]. Available: https://cds.cern.ch/record/1982429.

[11] CERN, "HTS for HEP — Executive Summary," CERN, Executive summary, Mar. 31, 2025. [Online]. Available: https://indico.cern.ch/event/1439855/contributions/6461516/attachments/3045888/5381824/HTS_4_HEP_executive_summary_final_20250331.pdf.