CERN-ACC-NOTE-2021-0004

29 January 2021

Longitudinal Phase Space Tomography Version 3

C. Grindheim and S. Albright

Keywords: Longitudinal Beam Dynamics, Tomography, Beam Performance Tracking

Summary

Longitudinal phase space tomography is essential in most of the CERN synchrotrons to measure longitudinal beam parameters. An implementation of phase space tomography written in Fortran95 has been used for many years, and is a vital part of the beam diagnostics in the PS Complex. To enable future developments and applications of tomography, a new version of the code was required. A fully refactored and rewritten version has been developed in mixed Python/C++, which allows more flexible and easily extended longitudinal tomography. This note gives an overview of how tomography is used to reconstruct longitudinal phase space distributions, and discusses the main differences between the original and new implementations.

Contents

1 Introduction

Tomographic reconstruction is used to measure the longitudinal phase space distribution of bunches in the synchrotrons of the PS Complex, and there is an increasing desire to implement it in the higher energy machines as well. The existing algorithm was written in Fortran95 and optimised with High Performance Fortran (HPF) to give fast run times and high quality reconstruction. The objective for the new version is to maintain high quality and reasonable run times, whilst also greatly increasing the flexibility of the code. The extra flexibility comes from the complete refactoring and translation of the code into a Python library, which will allow much more variety in how the code is used and facilitate new developments. In order to maintain suitable run times, the computationally expensive parts of the algorithm are written in C++, which is accessed directly from the Python code. The source code is available from a CERN GitLab repository (https://gitlab.cern.ch/longitudinaltomography/tomographyv3) along with a comprehensive set of example files.

1.1 Tomographic Principle

Tomography is the method of reconstructing an n-dimensional object from its $(n-1)$-dimensional projections. For example, a medical Computed Tomography (CT) scan uses many one-dimensional profiles of x-ray transparency taken from different angles to give doctors an image of a two-dimensional slice through a patient. The same principle may be used for longitudinal beam tomography, using a series of one-dimensional measurements of the longitudinal bunch profile to reconstruct the two-dimensional phase space [1]. The technique takes advantage of the fact that, from turn-to-turn, the particles are moving in phase space, resulting in the time projection being from a different angle at each measurement.

Figure 1 shows the profiles recorded every 25 machine turns over 1 synchrotron period for a simulated bunch making large quadrupole oscillations, Fig. 2 shows the corresponding distribution in phase space every 30 profiles at the projections indicated by the red lines in Fig. 1, illustrating how the phase space evolves with time.

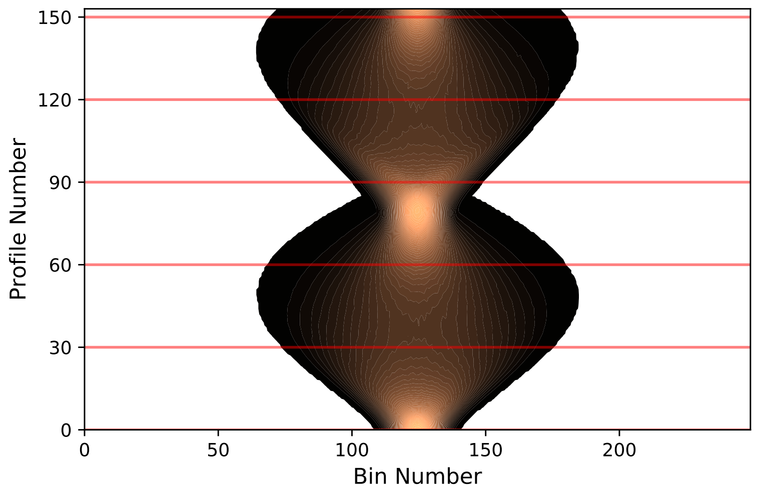
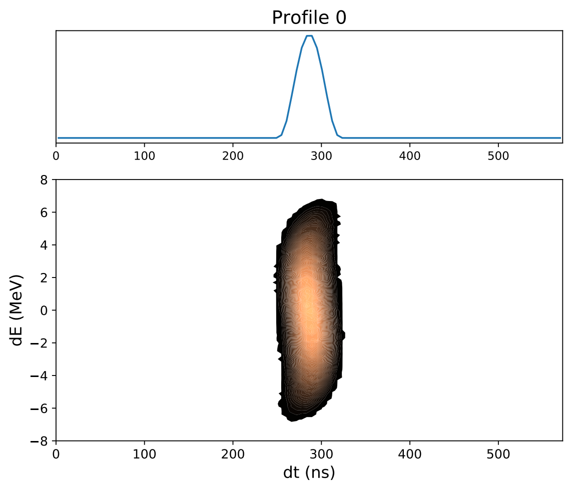
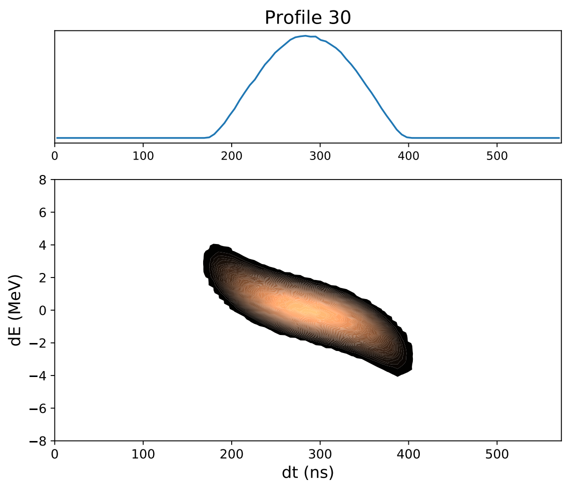


Figure 1: Waterfall plot of a bunch undergoing large quadrupole oscillations.

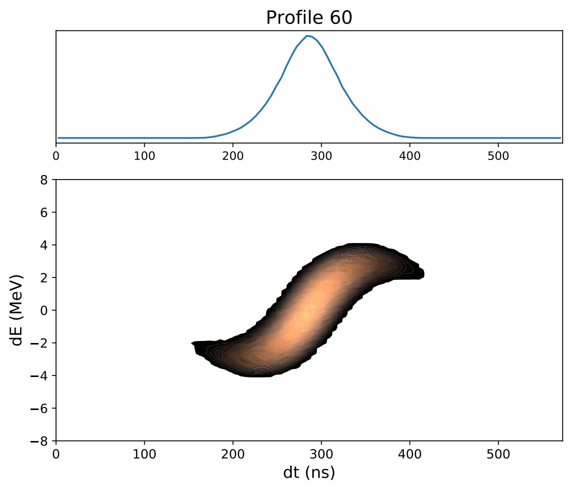
The tomography algorithm uses these measured profiles to create a reconstruction of the phase space distribution based on the Algebraic Reconstruction Technique (ART) [2]. As shown by the emergence of an S-shaped distribution in Fig. 2, synchrotron motion is non-linear. Since ART is a linear technique it would lead to a significant loss of quality if used directly. This is solved by introducing synchrotron motion to the algorithm via particle tracking. Test particles are distributed uniformly in phase space and tracked through the number of turns corresponding to the duration of measurement. On every turn that corresponds to a measured profile, the phase and energy of the particles are recorded, either directly or binned. The phase space distribution is then weighted based on the line density of each profile in an iterative process, which allows an accurate reconstruction of the phase space distribution to be made.



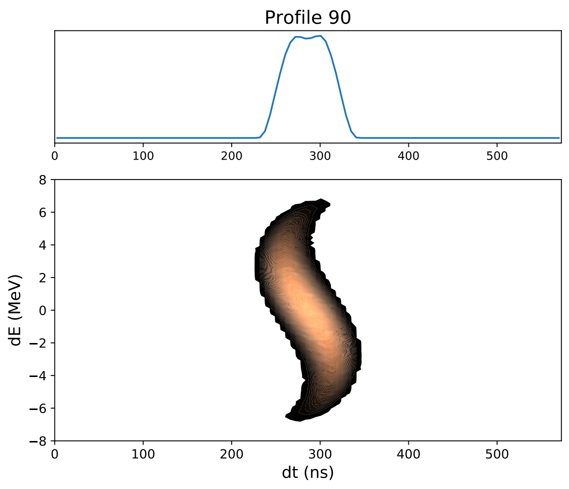
(a) Phase space at profile 0.



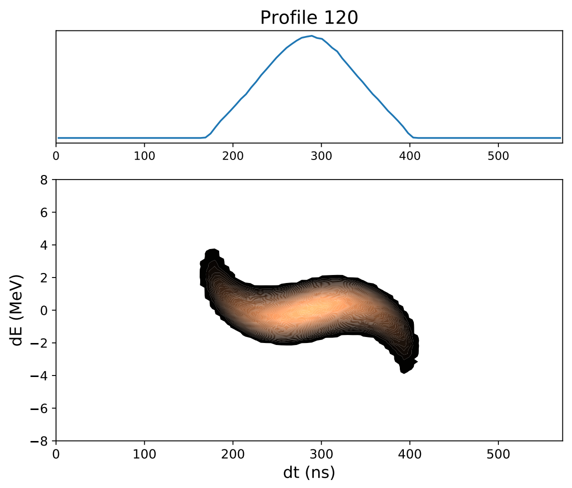
(b) Phase space at profile 30.



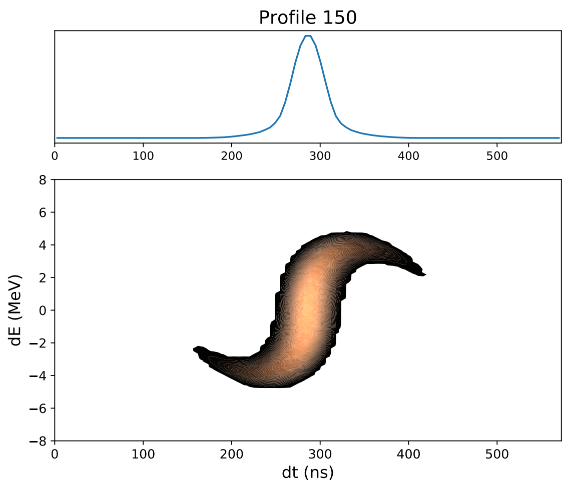
(c) Phase space at profile 60.



(d) Phase space at profile 90.



(e) Phase space at profile 120.



(f) Phase space at profile 150.

Figure 2: Simulated phase space distributions during one synchrotron period of quadrupole oscillations.

1.2 Original Fortran95 Implementation

The Fortran95 version takes input in a standardised format provided from an ASCII encoded text file or pipelined through STDIN. The output uses a mixture of ASCII files written to disc and STDOUT to return the results. For operational use, the data input and output, and data visualisation are controlled by the tomoscope application [3], an operational graphical user interface (GUI) for tomography.

Two different coordinate systems are used for the definition of the phase space. During tracking the particles have their coordinates defined in phase and energy. For binning and reconstruction these are converted to bin numbers, with each bin having width $\Delta\phi$ and height ΔE, determined by the maximum phase and energy deviations over the reconstruction. To save computational effort, the part of phase space considered is typically bounded by the iso-Hamiltonian contour corresponding to the maximum phase deviation in the measured profiles.

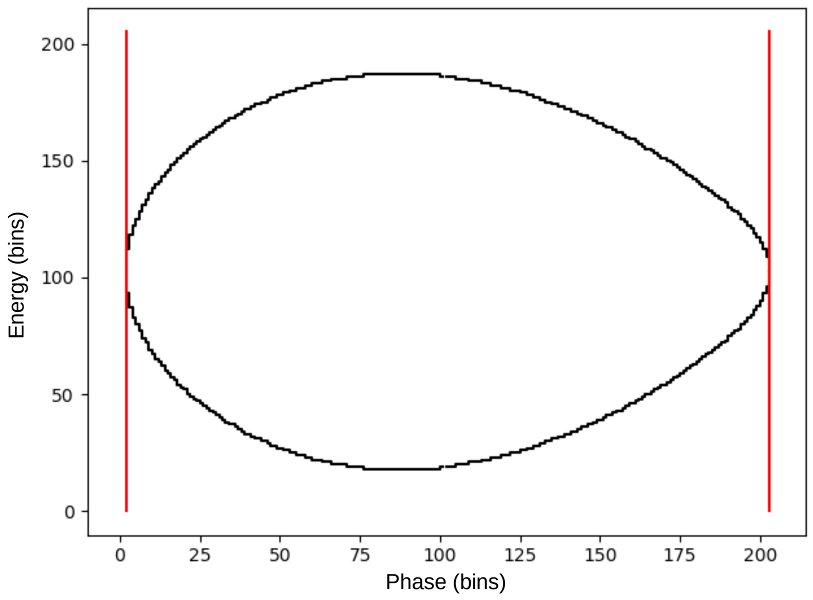


Figure 3: Limits of the reconstruction area in phase (red) and energy (black) given as number of bins. The area contained by the black contour will be populated with test particles, unless a flag is set to populate the entire phase space.

Figure 3 shows the reconstruction limits, computed for an arbitrary set of input parameters. The vertical red lines are the phase limits (in bin numbers), from these the energy limit is defined by the black line (in bin numbers). This region bounded by the black contour defines the area that will be populated with test particles, which are tracked and used for the reconstruction. Note that a flag can be set, populating the entire area with macroparticles.

The Fortran95 version uses a very memory efficient method for storing the results of tracking. Rather than storing particle coordinates at every turn (or a subset), three arrays are populated during the tracking process, *maps*, *mapsi* and *mapsweight*. These store respectively, the cell number for each cell in phase space, the phase bin number where the macroparticles from a given cell landed, and the number of macroparticles contained in each of these bins.

Figure 4 shows an example of the *maps* arrays. Here, two measured profiles correspond to a 5×5

time frame grid, with a 3×3 grid populated by test particles. This is analogous to the example shown in Fig. 3. The *maps* array is a 3-dimensional array with dimensions $m \times n \times l$, where m is the number of phase bins, n is the number of energy bins and l is the number of profiles, giving dimensions $5 \times 5 \times 2$ in this example.

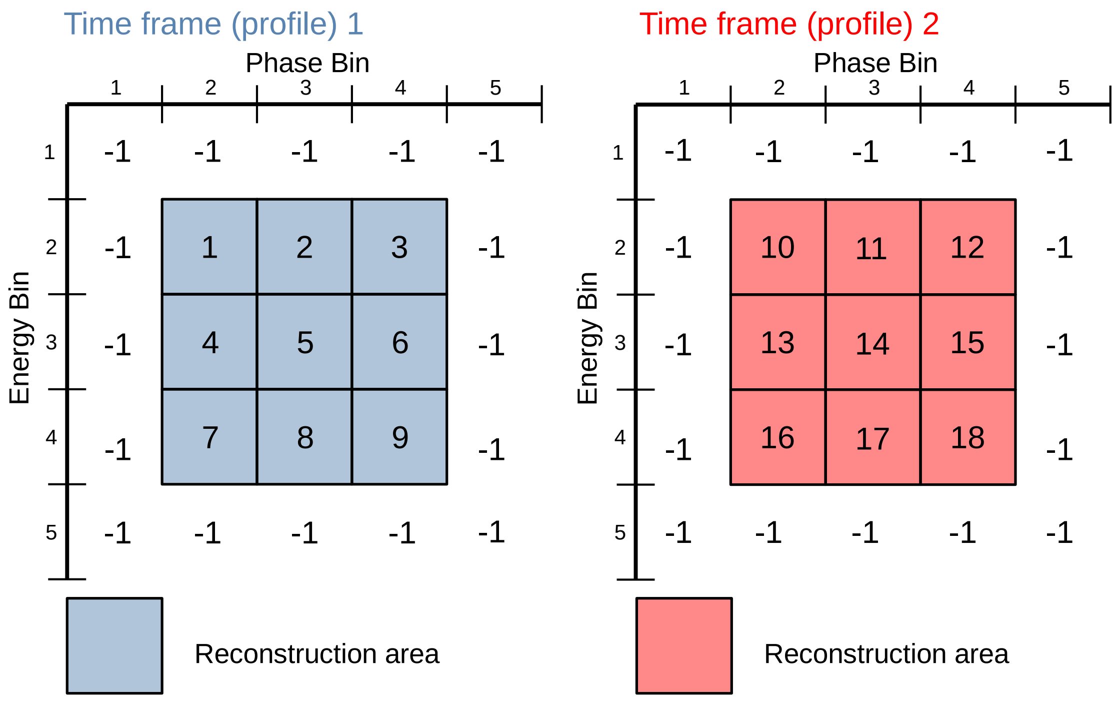


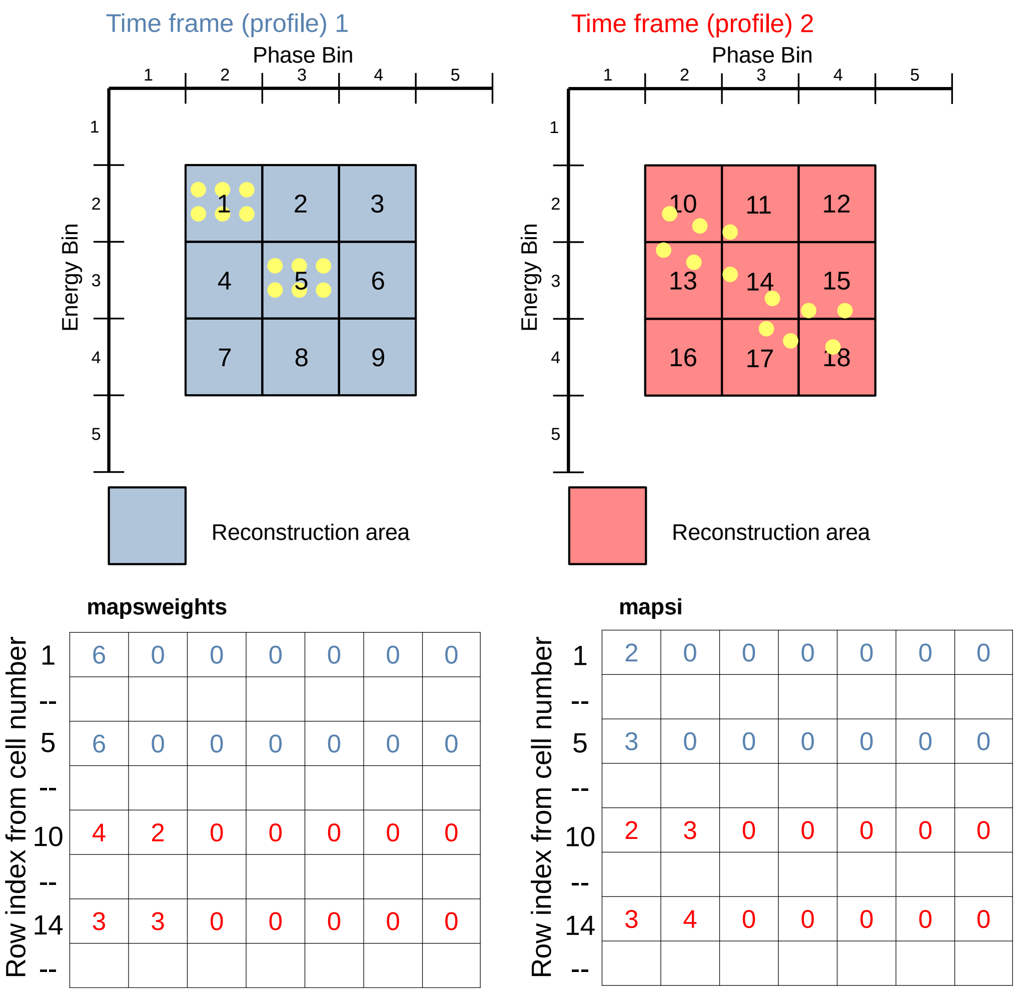
Figure 4: Layout and contents of the *maps* array. Figure shows maps array for first (blue) and second (red) time frame, out of N time frames. Empty cells are populated with -1 to flag that they are not valid for indexing.

The elements of the *maps* array that correspond to bins containing particles are numbered as shown in Fig. 4, with empty bins containing -1. The positive non-zero elements of *maps* are used to index the *mapsi* and *mapsweight* arrays. The *mapsi* and *mapsweight* arrays are initialised with dimensions $j \times k$, where j is the total number of cells in the *maps* array, and k is proportional to, and lower than, the number of macroparticles per cell. The array size can be increased if more bins are required, but this is very rare.

A uniform distribution of macroparticles is created within the reconstruction area at the turn corresponding to the profile to be reconstructed. An example sub-set is shown in Fig. 5, which uses the same time frame grids as Fig. 4, but for clarity the -1s are not shown. In this figure, six macroparticles are generated in cell number one and six are generated in cell number five (time frame 1), their positions after being tracked from the first to the second profile are shown in time frame 2. Also, the corresponding values of the *mapsweights* and *mapsi* arrays are given.

The test particles are initialised in cell numbers 1 and 5 ($maps[2, 2, 1] = 1$, $maps[3, 3, 1] = 5$), with their position after tracking to the next measured profile shown on the right. The values of the *mapsweights* and *mapsi* array, in row 1 and row 5, show that the 6 macroparticles in cell 1 are found in phase bin 2, the 6 macroparticles in cell number 5 are found in phase bin 3, and there are no others. On profile number 2, cell 1 becomes cell 10 ($maps[2, 2, 2] = 10$) and cell 5 becomes cell 14

($maps[3, 3, 2] = 14$). Rows 10 and 14 of *mapsweights* and *mapsi* are now indexed, showing 4 particles in phase bin 2 and 2 particles in phase bin 3 on row 10, and 3 particles in phase bin 3 and 3 particles in phase bin 4 on row 14.



mapsweights

1	6	0	0	0	0	0	0
--							
5	6	0	0	0	0	0	0
--							
10	4	2	0	0	0	0	0
--							
14	3	3	0	0	0	0	0
--							

mapsi

1	2	0	0	0	0	0	0
--							
5	3	0	0	0	0	0	0
--							
10	2	3	0	0	0	0	0
--							
14	3	4	0	0	0	0	0
--							

Figure 5: Figure illustrating the maps, mapsi and mapsweight arrays, together with particle positions at two time frames. Blue represents the initial time frame, and red represents the following time frame. The axes are from the binned coordinate system, counting from one (Fortran indexing).

Each particle is tracked from the first to the last time frame. Between the tracking and the sorting, the coordinates of each particle are mapped to and from the different coordinate systems and saved. When the tracking is complete, and the three arrays are filled, the program has knowledge of how many particles ended up in each phase bin on each profile. From the initial number of particles in each cell at every frame, an initial weight for the cells is given, which is updated during the reconstruction. Now, a fourth array (*phasespace*) holding the total weight factor of each cell accumulated from all time frames, is created. Everything is then set up for the tomographic reconstruction.

The tomographic reconstruction is an iterative process, consisting of three steps indicated in Fig. 6. First, the back-projection updates the weight of each element of *phasespace* proportional to the line density at each measured profile. Next, the projection takes the *phasespace* array and projects it onto the time-axis to give the line density equivalent to the current state of the reconstruction.

Finally, the difference between the reconstructed and measured line densities is taken. This process is repeated, using the difference from each step to update the *phasespace* array at the start of the next. After a specified number of iterations, the resulting *phasespace* is returned for further analysis.

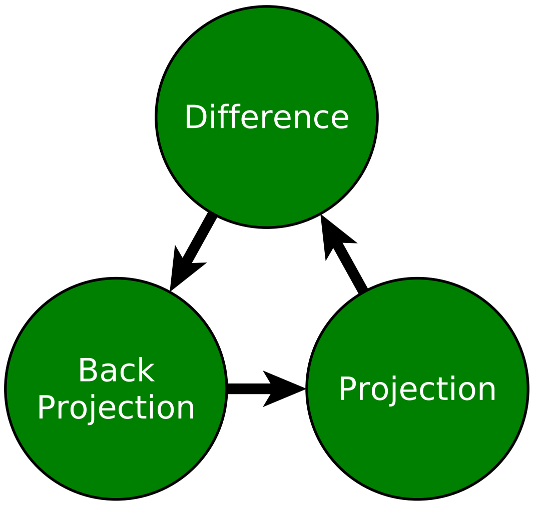


Figure 6: The three steps of the tomographic reconstruction process.

2 New Python/C++ Implementation

The main reason for the code translation is to have the tomography code in a format which is both more flexible and more modular, in a language that will facilitate this. The Fortran95 code is well tested and reliable, however due to the nature of the algorithm it would be very difficult to change the way the code is used or add more features.

The change of language and structure comes at a slight cost in run time, currently a full reconstruction in the new code takes on average about 50% longer than with the Fortran95 code. For typical operational use cases this extra time will be acceptable, as measurements are not required to be temporally close together. Additionally, it is now possible for parts of the algorithm, such as data pre-processing, to be skipped or replaced with alternative user defined versions. Further, when running repeated reconstructions with identical initial conditions the tracking and preparation can be executed once, and the coordinates used for multiple separate reconstructions corresponding to different measurements, significantly reducing the required run time.

Most of the preparation, such as populating the phase space prior to tracking, and data treatment, such as rebinning measured profiles, is unchanged in the new implementation. This section outlines the new code structure, and the differences in the reconstruction algorithm.

2.1 Refactoring from Program to Library

After initial translation to Python, the algorithm was refactored into a library. A library structure significantly increases the flexibility of the code and will simplify customisation and extension in the future. The packages and modules are identified in Table 1.

Package Name	Modules
cpp_routines	c++ source code
	tomolib_wrappers.py
data	profiles.py
tomography	__tomography.py
	tomography.py
tracking	machine.py
	__tracking.py
	tracking.py
	particles.py
	phase_space_info.py
utils	assertions.py
	exceptions.py
	data_treatment.py
	physics.py
	tomo_input.py
	tomo_output.py

Table 1: Packages and modules of the tomography library.

The purpose of each module is:

- C++ source code: The functionality written in C++, not to be accessed directly.
- tomolib_wrappers: Python wrappers around the C++ functions.
- profiles: Storage and treatment of measured profiles and calculation of self fields.
- __tomography: Base functionality for reconstruction classes.
- tomography: Derives and extends functionality of __tomography for default usage. Holds routines for full reconstruction based on a set of measured profiles and profile coordinates.
- machine: Storage and treatment of accelerator parameters.
- __tracking: Base functionality for particle tracking.
- tracking: Derives and extends functionality of __tracking for default usage. Holds routines to track macro-particles for given machine settings.
- particles: Generation and storage of particle distributions for tracking.
- phase_space_info: Calculation of phase space parameters.
- assertions: Assertions to check data is suitable for reconstruction.
- exceptions: Custom exceptions raised when errors specific to the tomography arise.
- data_treatment: Processing measured and generated data.
- physics: Fundamental calculations.
- tomo_input: Handling input of information from different sources.
- tomo_output: Handling output of information in different formats.

Together, the modules of the library can aid the user in quickly creating customised tomography programs. For typical operational usage, the data flow is shown in Fig. 7.

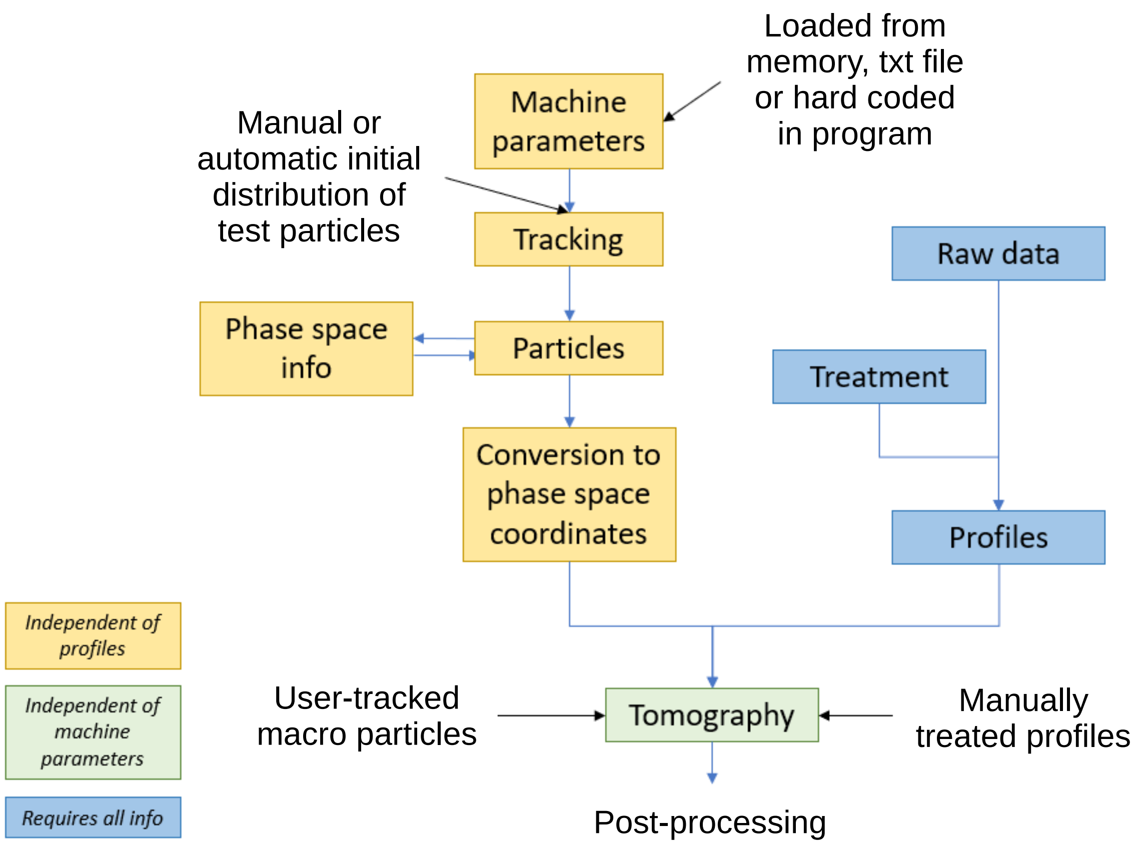

Loaded from memory, txt file or hard coded in program
Machine parameters
Manual or automatic initial distribution of test particles
Tracking
Raw data
Phase space info
Particles
Treatment
Conversion to phase space coordinates
Profiles
Independent of profiles
Independent of machine parameters
Requires all info
User-tracked macro particles
Tomography
Manually treated profiles
Post-processing

Figure 7: Restructured program. Black text represents optional user choices.

2.2 Tracking

In the Fortran95 implementation, the maps, mapsi and mapsweight arrays are populated during tracking, which requires the tracking to be periodically interrupted to perform the relevant calculations. The new implementation does not use the maps structure for the reconstruction, instead the coordinates of each particle are stored at the instant of every measured profile. Therefore, the tracking can be done in a continuous loop and the array populated with particle coordinates independently to the location of other particles. This structure also allows the tracking to be made highly parallelised, and makes it readily portable to GPU at a later date.

As in the Fortran95 implementation, both forward and backward tracking are used by default, with a maximum of two RF harmonics. In the forward direction the change in energy and phase is given by

$$\begin{aligned} \Delta E_{i+1} =& \Delta E_i + V_1 \sin\left(\Delta\varphi_i + \varphi_0\right) + V_2 \sin\left[h_r(\Delta\varphi_i + \varphi_0 - \varphi_{1,2})\right] - \Delta E_{acc}, \\ \Delta\varphi_{i+1} =& \Delta\varphi_i - \frac{2\pi h \eta_0}{\beta^2 E}\Delta E_i, \end{aligned} \tag{1}$$

where ΔE_i is the particle energy deviation from the synchronous energy on turn i, V_1 is the fundamental harmonic voltage, $\Delta\varphi_i$ is the particle phase deviation from the synchronous phase on turn i, φ_0 is the phase offset of the fundamental harmonic, V_2 is the voltage of the higher harmonic, h_r is the ratio of the higher harmonic to the fundamental harmonic, $\varphi_{1,2}$ is the phase offset between the higher and fundamental harmonics, ΔE_{acc} is the energy gain per turn, h is the fundamental harmonic number, η_0 is the phase slip factor, β is the synchronous relativistic velocity and E is the synchronous energy. For backwards tracking the equations are modified to

$$\begin{aligned} \Delta E_{i+1} =& \Delta E_i - V_1 \sin\left(\Delta\varphi_i + \varphi_0\right) - V_2 \sin\left[h_r(\Delta\varphi_i + \varphi_0 - \varphi_{1,2})\right] - \Delta E_{acc}, \\ \Delta\varphi_{i+1} =& \Delta\varphi_i + \frac{2\pi h \eta_0}{\beta^2 E}\Delta E_i. \end{aligned} \tag{2}$$

Note that only the sign of the additions to ΔE_i and $\Delta\varphi_i$ are changed (highlighted in red). Care should be taken with ΔE_{acc} for back tracking, since the change in energy per turn will change sign for the backward tracking, the acceleration should still be subtracted.

In the translation, the decision was taken to use the same coordinate system for tracking as in the original implementation $(\Delta\varphi, \Delta E)$. The coordinate system could be changed to an alternative, but for typical use cases there would be no benefit to the user. Further, for advanced use cases, it is likely that more complex tracking would be beneficial, in which case the built-in tracking routines would not be used.

To allow more complex tracking, an external tracker such as BLonD can be used [4]. BLonD allows highly complex tracking scenarios, far beyond what would be suitable for inclusion in the tomography code, the result of this tracking can then be imported, replacing the built-in tracking step. The tracking in BLonD uses $(\Delta t, \Delta E)$ coordinates, the particle coordinates must be converted to bin numbers prior to reconstruction, which would be required for any coordinate system.

2.3 Reconstruction

The principle of back-projection, projection, and difference has not been modified. However, because the particle coordinates are stored, each particle is weighted individually, rather than weights being applied to a grid.

2.3.1 Back Projection

In the back projection step, the weight of each particle is adjusted. At the end of each iteration, the particle weights are given by

$$W_i = W_i + \sum_{j=0}^{N} P_j(X_{i,j}), \tag{3}$$

where W_i is the weight of the ith particle, N is the number of measured profiles, P_j is the jth profile and $X_{i,j}$ is the phase bin of the ith particle at the jth profile. On the first iteration W_i is 0 and P_j are the measured profiles. On subsequent iterations P_j is replaced by the difference between measured and reconstructed profiles (ΔP_j).

Figure 8 shows how 3 particles would have their weight updated (weight indicated by size) on 5 steps under identical profiles. The weight of the particles at the end of each step is indicated in Table 2, on each step the weight of the particle is increased by the amplitude of the profile in the corresponding bin.

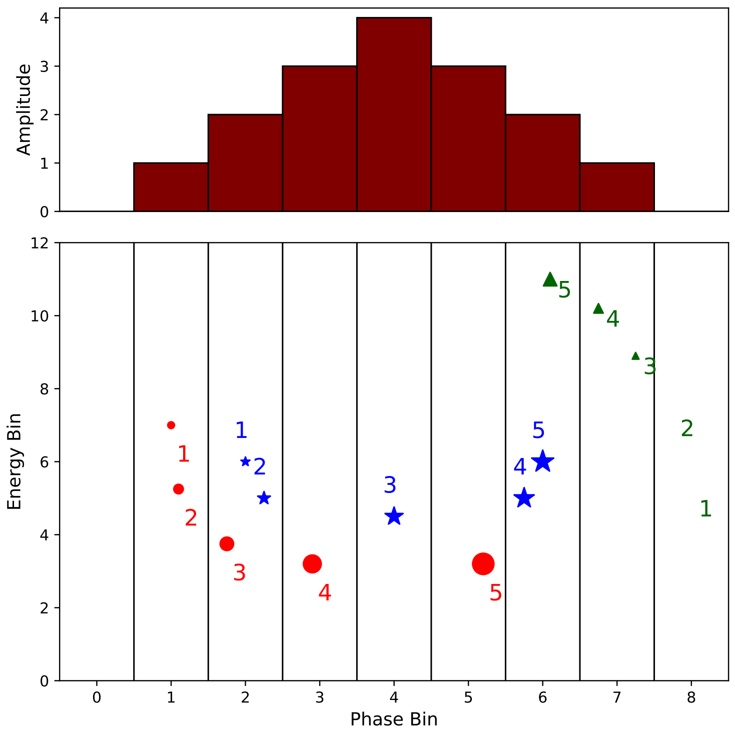


Figure 8: Weights of particles tracked over 5 steps, weights are given in Table 2. On steps 1 and 2, the weight of the green triangle is 0, therefore it cannot be seen.

Step	Red circle	Blue star	Green triangle
1	1	2	0
2	2	4	0
3	4	8	1
4	7	10	2
5	10	12	4

Table 2: Weights of the particles shown in Fig. 8.

On the first iteration, the weights of the particles are defined by the measured beam profile. For all subsequent iterations the difference between the measured and reconstructed profiles is used. Figure 9 shows the second back projection iteration, where the weights of the particles can be increased or decreased, depending on the bin. The corresponding weights are given in Table 3. Note that the green triangle has its weight reduced to 0 on step 2, but it does not go negative as this would be un-physical. At the end of each iteration, any particle with $W < 0$ is reset to $W = 0$.

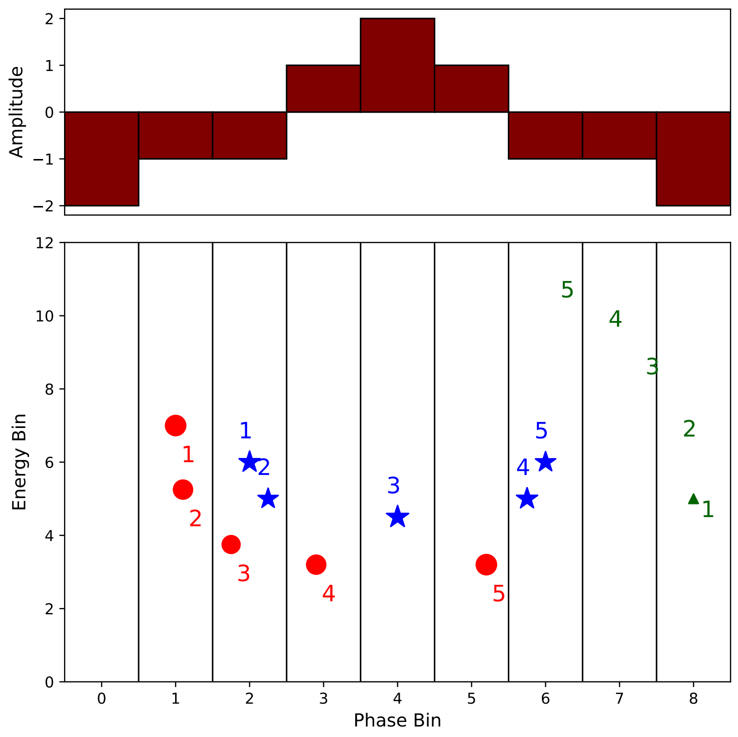


Figure 9: Weights of particles tracked over 5 steps, weights are given in Table 3. After the first step, the weight of the green triangle is 0, therefore it can no longer be seen.

Step	Red circle	Blue star	Green triangle
1	9	11	2
2	8	10	0
3	7	12	0
4	8	11	0
5	9	10	0

Table 3: Weights of the particles shown in Fig. 9.

2.3.2 Projection

In the projection step, the weights of the particles are summed in each bin, creating a time projection of the reconstructed phase space. The measured and reconstructed profiles can then be directly compared, and the difference used to update the reconstructed phase space. For each iteration the reconstructed projection is given by

$$\bar{P}_{j,n} = \sum_{i=0}^{M} w, w = \begin{cases} W_i, & \text{if } X_{i,j} = n \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where $\bar{P}_{j,n}$ is the nth bin of the jth reconstructed profile, M is the number of tracked particles, W_i is the weight of the ith particle and $X_{i,j}$ is the bin number of the ith particle at the jth profile.

Figure 10 shows the three particles discussed in the previous section (circle, star, triangle) contributing to the projections of 5 sequential profiles. For each bin of each profile (red, 1; blue, 2; green, 3; purple, 4; orange, 5) the amplitude is given by the sum of the weights of particles of corresponding colour and number in that bin. After computing the projections, the reconstructed profiles are normalised and can then be directly compared to the measured profiles, which are also normalised.

2.3.3 Difference

This step is unchanged in the new implementation, but is described in detail for completeness. After the projection step, the difference between the measured and reconstructed profiles is computed by

$$\Delta P_{j,n} = P_{j,n} - \bar{P}_{j,n}, \tag{5}$$

where $\Delta P_{j,n}$ is the difference between the measurement and reconstruction of the jth profile in the nth bin, $P_{j,n}$ is the measured value in the jth profile and nth bin, $\bar{P}_{j,n}$ is the reconstructed value at the jth profile and nth bin.

In order to obtain a faster convergence, the difference is multiplied by a weighting factor, defined by the number of macroparticles in each bin. In the tails of the bunch the number of particles is relatively small, therefore a difference in the projection at the tail indicates a larger error in weight than a difference in the centre. The weighting of each bin is given by

$$d_{j,n} = \Delta P_{j,n} \times \frac{M_{max}}{M_n}, \tag{6}$$

where $d_{j,n}$ is the weight of the jth profile of the nth bin, M_{max} is the number of macroparticles in the most densely populated bin and M_n is the number of macroparticles in the nth bin. The effect

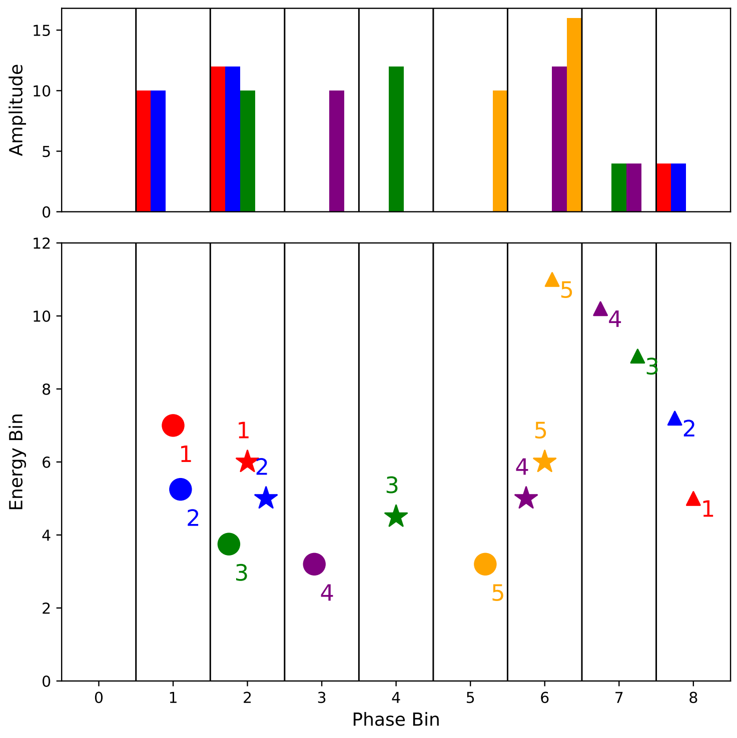


Figure 10: The contributions made by 3 particles with weights 10 (circle), 12 (star) and 4 (triangle) to the projections on 5 sequential profiles (red, blue, green, purple and orange).

of this can be seen in Fig. 11, which shows the measured and reconstructed profiles on the left, and the weighted and unweighted differences on the right after 1 iteration.

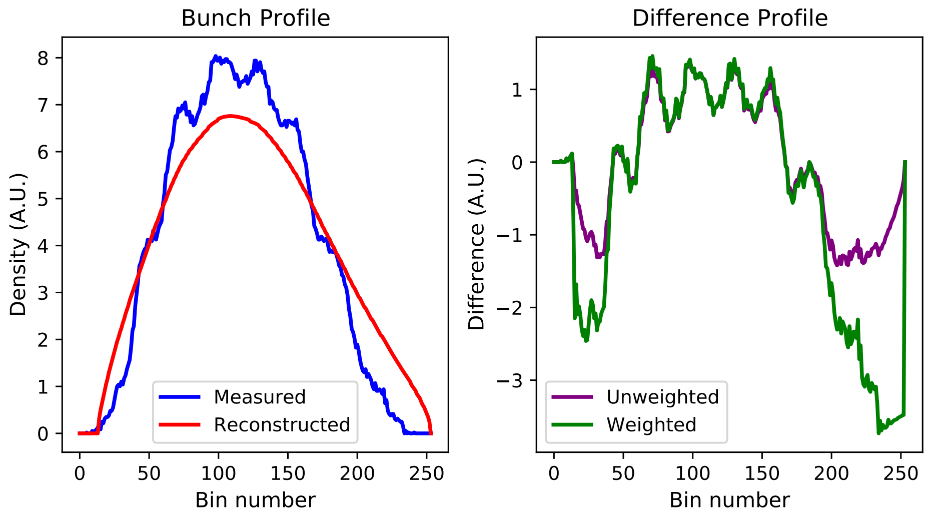


Figure 11: Measured and reconstructed profiles (left) and weighted and unweighted differences (right) after 1 iteration.

After each iteration, ΔP can be used to define a figure of merit for the quality of reconstruction called the discrepancy. The discrepancy between the reconstructed and the measured profiles is defined as

$$D = \sqrt{\frac{\sum_{j=1}^{N} \sum_{n=1}^{L} \Delta P_{j,n}}{NL}} \tag{7}$$

where D is the discrepancy for the current iteration, N is the number of measured profiles, L is the number of bins per profile.

2.3.4 Iteration

As in the original Fortran95 implementation, a reconstruction is computed in an iterative process. The back projection, projection and difference are run sequentially for a user-defined number of iterations. For suitable input data, it will rapidly converge to an accurate reconstruction of the longitudinal phase space.

Figure 12 shows the first profile for a bunch undergoing large quadrupole oscillations. The black line shows the measured profile, the coloured lines show the projections after 0, 1, 5, 10 and 30 iterations. Figure 13 shows the measured, reconstructed and difference waterfalls after the same number of iterations, clearly showing how the complete reconstruction converges with the measured profiles.

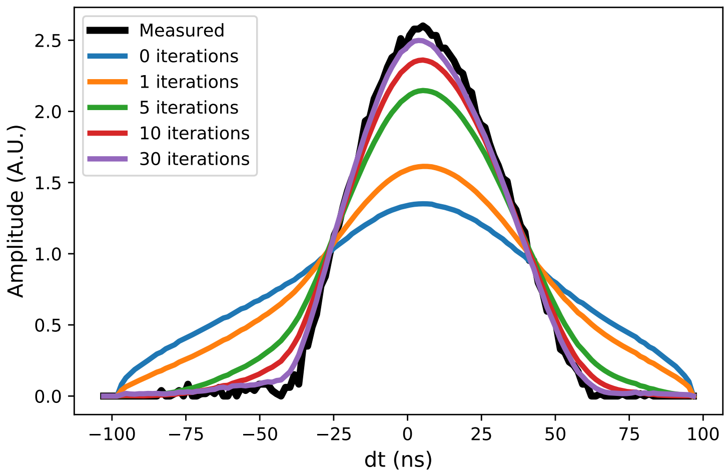


Figure 12: The projection of the first profile for a bunch undergoing large quadrupole oscillations after different numbers of iterations.

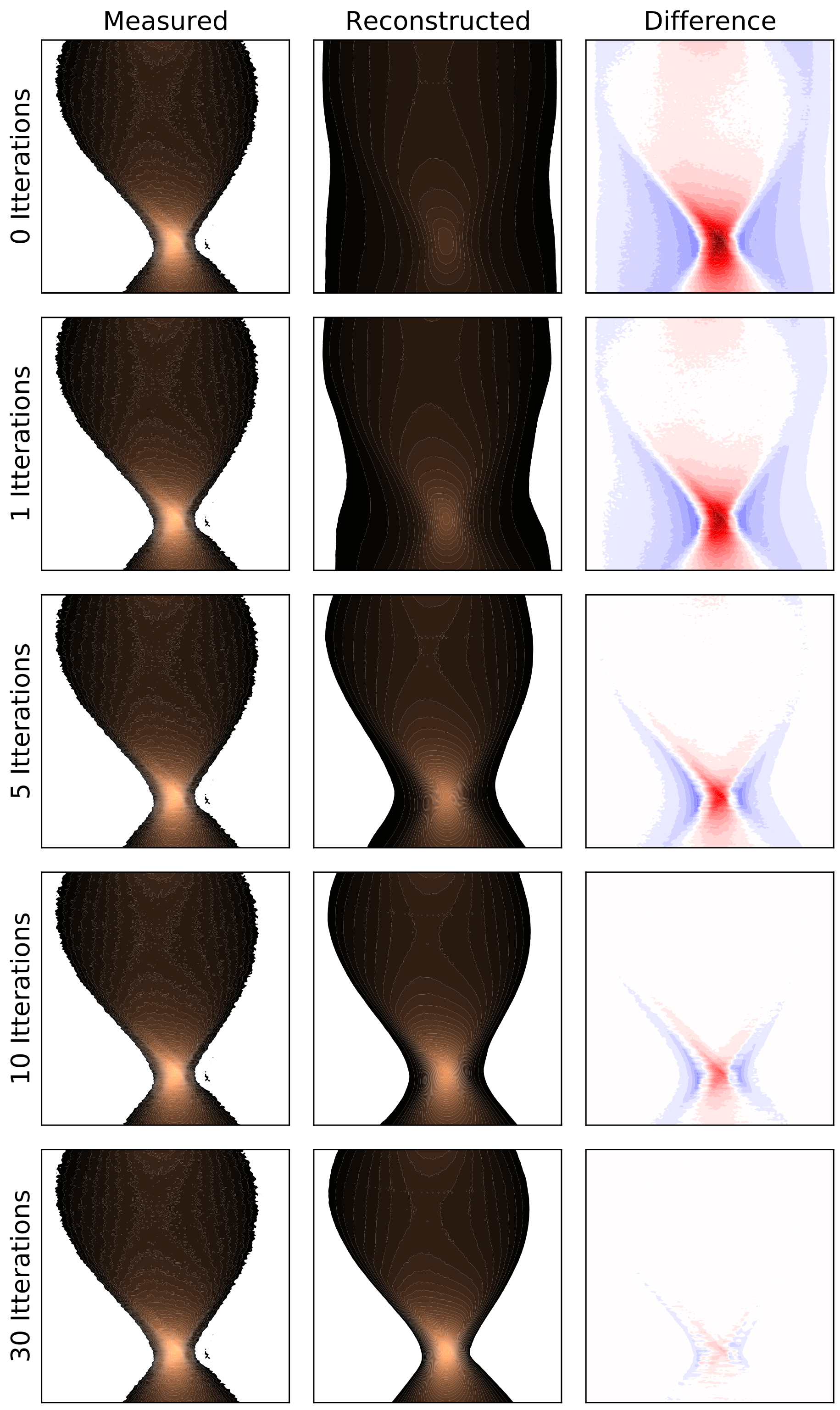


Figure 13: Measured, recreated and difference profiles for a bunch undergoing large quadrupole oscillations after different numbers of iterations. The colour axis are fixed, showing how an increasing number of iterations allows the reconstruction to converge to the measured profiles.

2.4 Online Tomography

The input/output of the Fortran95 code required intermediate files to be written to disk, and the use of STDIN and STDOUT. For backwards compatibility, this functionality can be replicated using functionality found in the tomo_input and tomo_output modules. However, it is no longer mandatory. The ability to perform a full reconstruction and retrieve results with all information remaining in memory raises the possibility of faster run times in suitable situations. One such use case would be online tomography, where every cycle from an accelerator would be measured, the reconstruction performed and phase space parameters published [5].

Online tomography would impose strict requirements on the run time of the reconstructions. Since the library allows for the tomography routines to run independently from the rest of the algorithm, a minimal version can be used, which only executes the iterative tomography routines. The functions for this approach to reconstruction have been written in optimised C++, parallelized using OpenMP.

In this case, instead of tracking particles for each reconstruction, tracking is performed once for given measurement conditions and saved to a database. Then, for each reconstruction, particle coordinates are loaded from the database and only the iterative tomography routine is run. This has been demonstrated using reference measurements from run 2, and shown to allow a reconstruction to be computed in significantly under 1 second, the study is discussed in the following section.

3 Reconstruction Quality and Performance

The two primary objectives for the translated version were:

1. The reconstruction quality be at least as good as the original version. This was determined by comparing the discrepancy between the measured and reconstructed profiles for a large set of reference measurements.

2. Run times below half a basic period must be possible. This limit was chosen so that two reconstructions would be possible within the length of the PSB cycle, to allow an injection and extraction measurement to be done.

This section describes the tests used to confirm that both these objectives were met.

3.1 Output Quality

To maintain accurate measurements of the longitudinal distribution, it is imperative that the quality of the results is at least as good with the new implementation as it is with the Fortran95 version. To that end a benchmarking was done of the Python/C++ version with the Fortran95 version. At the end of reconstruction, the discrepancy between the measured and reconstructed profiles gives a measure of the reconstruction quality.

For the PS, PSB, AD and LEIR a large number of reference measurements exist, which have been used to compare the discrepancy of both implementations. In total 1219 input files are available, however many are not intended for reconstruction, such as references for PS transition crossing. Of the available files, 919 gave sufficiently accurate reconstructions to be of interest. For each of these input files the reconstruction was performed with both versions of the code, and the discrepancy of the final reconstruction compared. Of the 919 reference files considered, 586 had a lower discrepancy when reconstructed with the Python/C++ implementation.

Figure 14 gives a comparison between the two implementations for all reference input with a discrepancy below 4×10^{-2}, which is the majority of cases. The main plot shows a histogram of the discrepancy of each input file, the Python histogram is in red and the Fortran in blue, where the two overlap is shown in purple. The inset plot shows a histogram of the ratio of the Python/C++ discrepancy to the Fortran discrepancy. From Fig. 14 it can be seen that there is very little difference between the discrepancies for the two codes, with a slightly lower average discrepancy from the Python/C++ implementation.

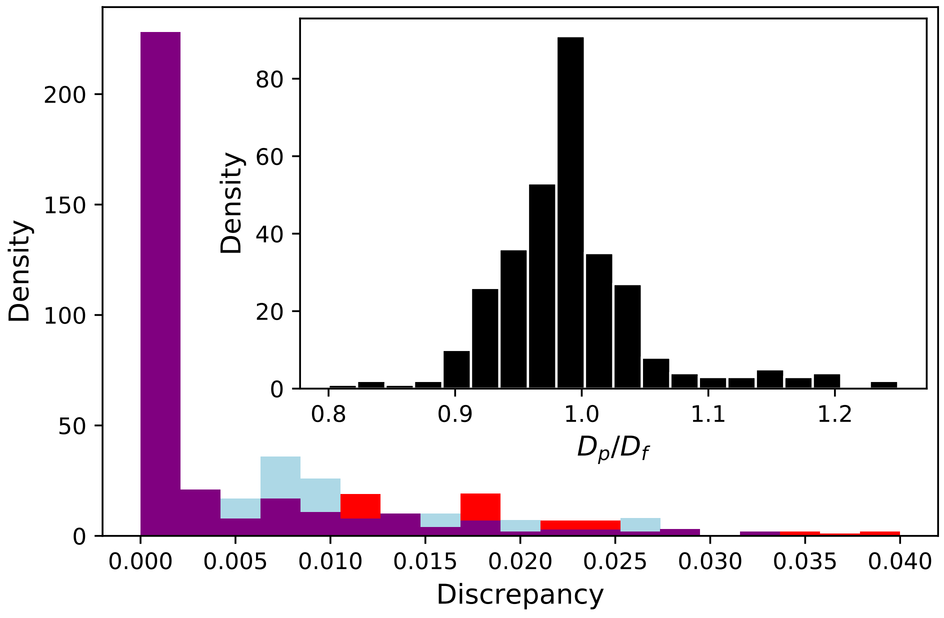


Figure 14: A comparison of the reconstruction discrepancy for tomography reference files. Main plot: Histogram of discrepancies for the Python/C++ code (red) and Fortran (blue), overlapping regions are shown in purple. Inset: Histogram of the Python/C++ discrepancy to the Fortran discrepancy for each reconstruction.

3.2 Fast Reconstruction

As described in Section 2.1, the tomography library is designed in such a way that parts of the reconstruction process can be executed separately. This allows a minimalist program to be created, consisting only of the tomographic reconstruction without particle tracking or data treatment. A database can be filled with tracked particles corresponding to frequently used machine settings, these can then be loaded on demand and combined with the measured profiles for faster reconstruction.

The objective of this benchmark was to find the fastest reconstruction time in the Python/C++ implementation that can produce a reconstruction at least as accurate as the Fortran95 implementation for a given input data file. For each test case the rebinning factor and the number of tracked macroparticles were varied. For one of the input files, the resulting run times and discrepancies are illustrated in Fig. 15, where the horizontal dashed line indicates the discrepancy of the original implementation with the rebinning factor and number of tracked particles (snpt) as defined in the reference file. The snpt defines the square root of the number of macroparticles to be tracked per phase space bin. In this case it can be seen that a rebin factor of 3, and snpt of 1 gives a lower discrepancy than the Fortran95 and slightly more than 0.1 s reconstruction time.

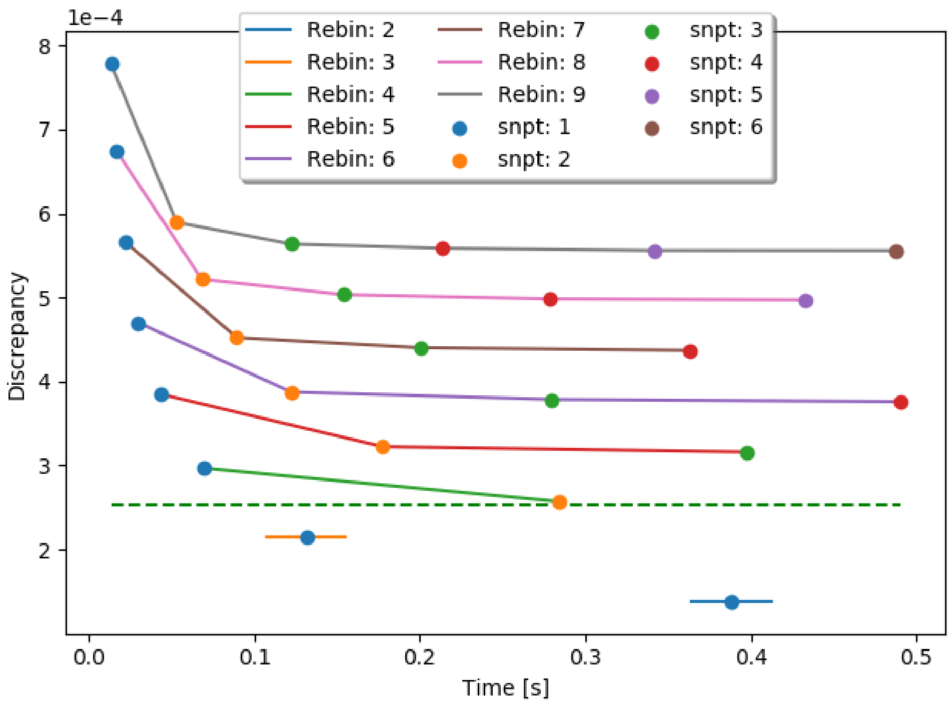


Figure 15: Discrepancy and run time of the tomographic reconstruction while varying the input parameters. The colours of the lines correspond to the value of the rebinning parameter, the colours of the dots indicate the snpt parameter, which is the square root of the number of tracked particles per area of phase space. The dashed green line shows the discrepancy of this input file when reconstructed with the Fortran95 code.

For each of the input files tested the fastest reconstruction that gives a discrepancy at most equal to that from the Fortran95 implementation was used. The reconstruction time and relative discrepancy for each test case are shown in Fig. 16. The vertical dashed line indicates a run time of 0.5 seconds, this was chosen as an upper limit to ensure that it would be possible to perform two reconstructions per 1.2 s basic period, whilst allowing an additional 100 ms per reconstruction for data acquisition, pre- and post-processing and data transfer. The cycles and settings that correspond to these results are shown in Table 4.

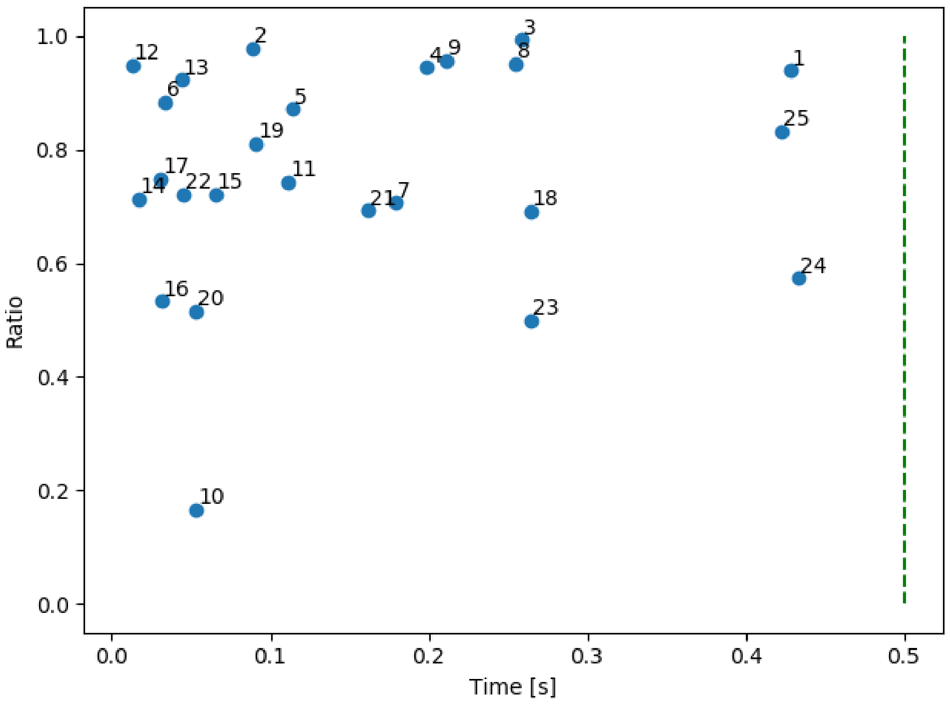


Figure 16: Figure showing the run time and ratio between the discrepancy of the Python/C++ version and the Fortran version. Each dot represents a file with some optimal reconstruction parameters (see Table 4). The dashed green line marks the 0.5 second limit.

#input	Machine	Cycle	Cycle-time	Run Time	D_p/D_f	Rebin	Snpt
1	LEIR	NOMINAL	C1840	0.428	0.939	2	1
2	LEIR	EARLY	C1675	0.089	0.979	2	2
3	PSB	8b4e	C796	0.259	0.993	2	2
4	PSB	BCMS	C798	0.199	0.945	2	2
5	PSB	LHC25	C798	0.114	0.872	2	2
6	PSB	LHCINDIV	C798	0.033	0.883	2	1
7	PSB	NORMGPS	C778	0.179	0.706	2	2
8	PSB	SFTPRO (left)	C798	0.255	0.952	1	4
9	PSB	SFTPRO (right)	C798	0.211	0.957	1	4
10	PS	AD	C1060	0.053	0.165	1	1
11	PS	AD	C200	0.111	0.742	1	1
12	PS	ILHC100#4b	C1130	0.013	0.948	2	1
13	PS	ILHC75#3b	C1050	0.044	0.923	2	2
14	PS	ILHC75#3b	C237	0.017	0.714	8	2
15	PS	ILHC75#3b	C344	0.065	0.722	2	1
16	PS	LHC25#48_BCMS	C1940	0.031	0.533	2	1
17	PS	LHCINDIV	C1395	0.031	0.748	2	1
18	PS	LHCINDIV	C200	0.264	0.690	1	1
19	PS	LHCINDIV	C740	0.091	0.811	2	1
20	PS	SFT129Xe39+	C1130	0.053	0.515	1	1
21	PS	SFT129Xe39+	C237	0.162	0.693	4	2
22	PS	SFT129Xe39+	C500	0.045	0.721	2	1
23	PS	TOF	C690	0.264	0.498	1	1
24	PS	TOF	C200	0.433	0.573	2	1
25	LEIR	NOMh3h6	C1840	0.423	0.831	2	2

Table 4: 2018 Reference measurements for scatter plot in Figure 16.

4 Conclusion

Longitudinal phase space tomography is a well established operational tool in the PS, PSB, AD and LEIR. The operational implementation was a Fortran95 program optimised with High Performance Fortran (HPF), giving fast high quality reconstructions in a short time. To allow future developments, a new implementation written in mixed Python/C++ has been developed.

Thorough testing of the new implementation has shown that the reconstruction quality is equivalent to that of the original, and whilst the run time is longer for operational use cases the difference is small enough to not be a concern. Further, due to the flexibility that has been introduced it is now possible to run the reconstructions in a variety of ways. For applications requiring many reconstructions with the same accelerator parameters, it is possible to pre-track the test particles and then run only the iterative tomography routine, saving significant execution time and enabling online tomography. In instances where more complicated tracking is desired, such as with intensity effects or complex voltage functions, external tracking in a code like BLonD can be used in conjunction with the tomographic reconstruction.

Process	New/Modified Functionality	Module
Data I/O	All data can be loaded from/stored in memory	Multiple modules
Accelerator parameters	Additional input methods	machine.py
Tracking	Initial distribution can be user defined	tracking.py
Tomography	Tracked coordinates can be input by user	tomography.py
Measured profiles	Pre-processed profiles can be input directly	profiles.py
Phase Space Weighting	Test particles weighted directly	tomography.py
Post-processing	Direct conversion to macro-particle distribution	tomo_output.py

Table 5: Most significant changes and new features introduced to the Tomography code.

Table 5 summarises the most significant changes that have been made to how different processes within the reconstruction operate and new features that have been added. Further developments are planned in the future, these will be aimed at facilitating tomography in the SPS and LHC, and making the pre and post-processing suitable for online tomography.

5 Future Work

Two main areas of development are intended for the future. Firstly, the proposal for automated tomography, detailed in [5], is planned to be implemented. The tomography routines described here allow the reconstruction to be run very quickly, the future developments will focus on suitable pre- and post-processing algorithms to rapidly treat the measured data and analyse the reconstructed phase space. Secondly, the new algorithm is well suited for parallelisation, to allow even faster reconstruction a GPU accelerated implementation is planned. The GPU version should allow reasonable reconstruction times for very large numbers of bunches, giving the opportunity to reconstruct complete bunch trains in the PS, SPS and LHC.

References

[1] S. Hancock, "A simple algorithm for longitudinal phase space tomography," Tech. Rep. CERN-PS-RF-NOTE-97-06.

[2] R. Gordon, "A tutorial on art (algebraic reconstruction techniques)," IEEE Transactions on Nuclear Science, vol. 21, no. 3, pp. 78–93, 1974.

[3] S. Hancock and J.-L. Sanchez Alvarez, "A pedestrian guide to online phase space tomography in the CERN complex," Tech. Rep. CERN-PS-RF-NOTE-2001-010, CERN, Geneva, Switzerland.

[4] CERN, "Cern beam longitudinal dynamics code blond." http://blond.web.cern.ch, 2019. [Online; accessed 21-October-2020].

[5] S. Albright, "A Proposal for an Automatic Longitudinal Tomography Demonstration Project in the Proton Synchrotron Booster," Tech. Rep. CERN-ACC-NOTE-2020-0014, CERN, Geneva, Switzerland, 2020.