14th International Particle Accelerator Conference,Venice, Italy JACoW Publishing
ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL100

A PYTHON API FOR THE PARTICLE TRACKING CODE PLACET

A. Pastushenko[*], D. Schulte, A. Latina, CERN, Geneva, Switzerland

Abstract

The tracking code PLACET is widely used in the linear collider community to simulate the beam dynamics. It is a powerful tool for analyzing the static and dynamic imperfections in the lattice and has many built-in correction techniques. The original PLACET code was written in C with a Tcl interface. Detailed data analysis, including plotting is often performed with other programming languages, primarily Python. This paper describes the project of the Python application programming interface (API) for PLACET.

INTRODUCTION

PLACET is a tracking code originally developed by Daniel Schulte [1] and currently maintained by a dedicated team at CERN. The code simulates beam dynamics under static and dynamic imperfections, including both single- and multi-bunch effects, and serves as the main tool for simulating the Drive Beam and Main Beam of CLIC. PLACET is designed as a standalone application, written in C and C++ and accessed via an interface in Tcl/Tk. To better facilitate data analysis and representation, the original Tcl interface was extended [2] with Octave and Python using SWIG (Simplified Wrapper and Interface Generator) [3].

In this format, one sets up the beamline, initiates the beams, performs the tracking, and applies corrections using native Tcl commands. For additional functionality, Octave and Python capabilities can be utilized. Some Tcl commands have also been extended to work from within the interface, primarily in Octave. However, the embedded interfaces present a couple of disadvantages:

- **Limited flexibility:** Since the primary interface is still TCL, users may be limited in their ability to take advantage of Python's full range of features and libraries.
- **Less streamlined process:** Users need to switch between the Tcl and Python environments, potentially affecting the development process's efficiency.

This paper introduces a new software solution that offers an alternative approach integrating PLACET and Python. Instead of embedding a Python interface inside the Tcl environment, the proposed solution implements PLACET with Tcl interface as a Python package.

MOTIVATION AND DESIGN GOALS

Python is a general-purpose language with a large ecosystem of libraries and tools for data processing, visualization, and machine learning. It is one of the most popular programming languages used worldwide. The key features of Python in comparison to Tcl are:

- **Access to Python libraries and tools:** Python has a vast ecosystem of libraries and tools that can be used to enhance the functionality of the software, such as data visualization or machine learning.
- **Greater flexibility:** Python is a more flexible language than Tcl, which can make it easier to modify and extend the software as needed.

In order to benefit from these advantages, we develop a software that:
1. is a Python package.
2. is lightweight and does not require any PLACET modifications.
3. provides an intuitive user experience.

This approach involves implementing the software in a format where PLACET runs as a background process and is accessed through pipes. Alternatively, one could use the native PLACET code and modify it to use Python instead of Tcl. While that may be a more comprehensive solution, it would require rewriting PLACET on a wide scale, which would demand significantly more time and human resources.

OVERVIEW

The Python package `placetmachine` has been created and is available at [4]. It is actively being improved, with more PLACET commands being added. The package includes the key class `Machine` and two smaller sub-packages, as shown in Fig. 1.

`placetmachine.placet`

This sub-package forms the actual interface between the PLACET process and Python, constructing the corresponding API, namely `placetmachine.placet`. The PLACET commands, including some key Tcl ones, have their counterparts in the `Placet` class. When a command is executed in Python, it creates a request in the correct format for PLACET. The process executes the request, and the result is read and sent back to Python. So far, 49 commands have been transformed into Python and included in `Placet`. These commands mostly include the ones used to perform the Beam Based Alignment (BBA) and emittance tuning studies for the CLIC 380 GeV machine [5].

To spawn PLACET as the child process in the interactive mode, the `pexpect` module is used. The `Communicator` class handles the data transfer to and from the PLACET process spawned with `pexpect`. With additional classes they form the `Placet` class in `placetmachine.placet`.

Though it is possible to write scripts in pure Python using this package, doing so would be inefficient. Also, since Tcl itself is also a scripting language, there is no need to wait for the process to terminate to get results. Thus, `placetmachine` can be used with tools like Jupyter.

[*] andrii.pastushenko@cern.ch

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL100

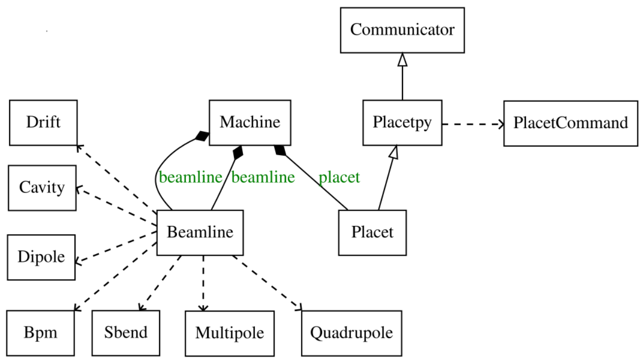

Figure 1: Class diagram of `placetmachine` package.

`placetmachine.lattice`

The second sub-package is `placetmachine.lattice`. It is used to store the beamline and element properties. The need for this package arises from the desire to have data easily accessible. For example, running the command `ElementGetAttribute` thousands of times could significantly slow down the execution. It is also driven by the design choices that will be discussed later.

`placetmachine`

The key part of the package is the `Machine` class. It features functions to create the beamline, create beams, assign static errors, perform tracking, and apply corrections. It modifies some aspects of PLACET usage that are, by default, in Tcl. In PLACET, users must define so-called surveys that tell PLACET how to displace elements before performing tracking. If a survey is not specified, no beamline errors are applied, even if the elements are displaced initially. It is common practice to save beamline alignments every time tracking or correction is done, so they can be used as a survey for the next tracking or correction. It is also common to simulate many beamlines at once by specifying the parameter `machines`. To keep track of the changes to the beamline, users must define a custom survey function and explicitly save the positions file. In `Machine`, we aimed to include these and other similar routines to allow them to be performed automatically, streamlining the code as much as possible. The changes include:

- The number of simulated machines is always 1. If more machines need to be simulated in parallel, a new instance of `Machine` can be created.
- The surveys for tracking and corrections are read from files generated prior to execution by a `Beamline` attached to the `Machine`.
- After performing a correction, the misalignments inside PLACET and `Beamline` are synchronized.

USAGE

Installation

The package can be accessed at the Gitlab repository at [4]. Instructions on how to install it are provided in the repository.

Functions Available in the `Machine` Class

The full list of functions in the `Machine` class is available in the dedicated repository. In this section, we will highlight the key functions implemented so far:

- `create_beamline()`: Creates the beamline and attaches it to the `Machine` instance.
- `import_beamline()`: Imports an existing beamline and attaches it to the `Machine` instance.
- `make_beam_slice_energy_gradient()`: Creates a sliced beam.
- `assign_errors()`: Applies static imperfections to the lattice.
- `track()`: Tracks the beam using the `TestNoCorrection` command from PLACET.
- `one_2_one()`: Applies the one-to-one steering correction using the `TestSimpleCorrection` command from PLACET.
- `RF_align()`: Realigns the accelerating structures using the wake monitors with the `TestMeasuredCorrection` command from PLACET.
- `eval_twiss()`: Evaluates the Twiss functions along the beamline using the `TwissPlotStep` command from PLACET.
- `eval_orbit()`: Evaluates the beam orbit along the beamline using the `BpmReadings` command from PLACET.

Example Scripts

Listing 1 demonstrates how how to perform beam tracking with the `Machine` class. First, we create a new instance with default parameters, which starts the PLACET process running in the background. Second, we create the beamline from a file containing the lattice in PLACET format using the `create_beamline()` function. This creates a `Beamline` object that is an attribute of the `Machine` and can be accessed as a `clic.beamline`. We assign a name to the beamline (`name = "ml"`) and setup up the cavity parameters, such us dimensions, phase, etc., which are required for wake kick estimations. Next, we create the beam using the `make_beam_slice_energy_gradient()` function. Similar to PLACET, we set the name, provide the number of slices, number of macroparticles per slice, energy and energy gradient offsets, error seed, beam emittance, beta functions, etc. Finally, the `track()` function tracks the beam through the beamline and returns a `pandas.DataFrame` with the the name of the beamline and the beam used, as well as the horizontal and vertical emittances.

Listing 1: Particle tracking for the error-free lattice

```
import placetmachine as pl

clic = pl.Machine()

clic.create_beamline("ml_beamline.tcl", name = "
     ml", cavities_setup = {...})

beam = clic.make_beam_slice_energy_gradient("
     main_beam", 11, 5, 1.0, 1.0, 1111, emitt_x =
      8.0, emitty = 0.1, ..)

perfect_line = clic.track(beam)
```

The Listing 2 shows how to perform a simulation of the one-to-one steering applied to a beamline with randomly distributed static misalignments 100 times. In the for loop, we use the `assign_errors()` function that misaligns the elements in the current beamline according to the PLACET built-in survey. In this case, `"default_clic"` corresponds to the `Clic` survey in PLACET. The RMS values of the misalignments are given as a dictionary of `errors`. For example, `'quadrupole_x'` is the RMS error of the horizontal alignment of quadrupoles and `'bpm_x'` is the RMS error of the horizontal alignment of BPMs. The function also accepts the parameters of the girder alignment (`'scatter_y'` and `'flo_y'`). The one-to-one steering is performed with the `one_2_one()` function. It requires the name of the beam to be used in the tracking and accepts many extra parameters, such as `'bpm_resolution'`, which defines the BPMs resolution for the one-to-one steering.

Listing 2: Particle tracking and one-to-one steering

```
import pandas as pd

...

errors = {
    'quadrupole_x': 14.0,
    'bpm_x': 14.0,
    ...
}
for i in range(100):
    clic.assign_errors("default_clic",
        static_errors = errors, scatter_y =
        12.0, flo_y = 5.0)

    summary = clic.track(beam)
    one2one = clic.one_2_one(beam, bpm_resolution
         = 0.1)
    summary = pd.concat([summary, one2one])
```

SUMMARY AND OUTLOOK

In this paper, we introduced the `placetmachine` Python package as an alternative interface for the PLACET particle tracking software. The package streamlines the process of setting up, running, and analyzing simulations with PLACET, while taking adantage of Python's extensive ecosystem. It includes essential features for managing beamlines, creating and tracking particle beams, and applying correction methods. Future developments will focus on incorporating more PLACET commands and adapting various correction routines (DFS, RF alignment, etc.) for Python. Ultimately, `placetmachine` aims to provide a comprehensive user-friendly solution to access PLACET.

REFERENCES

[1] D. Schulte, "PLACET: A Program to Simulate Drive Beams", in *Proc. EPAC'00*, Vienna, Austria, Jun. 2000, paper TUP7B05, pp. 1402–1404.

[2] A. Latina, Y. I. Levinsen, D. Schulte, and J. Snuverink, "Evolution of the Tracking Code PLACET", in *Proc. IPAC'13*, Shanghai, China, May 2013, paper MOPWO053, pp. 1014–1016.

[3] Simplified Wrapper and Interface Generator, `https://www.swig.org/`.

[4] Placetmachine, `https://gitlab.cern.ch/apastush/Placetmachine/`.

[5] A. Pastushenko and D. Schulte, "Emittance tuning bumps for the Main Linac of CLIC 380 GeV", presented at the IPAC'23, Venice, Italy, May 2023, paper THPL087, this conference.