ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE029

A LINEAR ACCELERATOR SIMULATION FRAMEWORK

J. Snuverink, John Adams Institute at Royal Holloway, University of London, Egham, UK
N. Fuster-Martinez, IFIC (CSIC-UV), Valencia, Spain
J. Pfingstner, CERN, Geneva, Switzerland and University of Oslo, Oslo, Norway

Abstract

Many good tracking tools are available for simulations for linear accelerators. However, several simple tasks need to be performed repeatedly, like lattice definitions, beam setup, output storage, etc. In addition, complex simulations can become unmanageable quite easily. A high level layer would therefore be beneficial. We propose LinSim, a linear accelerator framework with the codes PLACET and GUINEA-PIG. It provides a documented well-debugged high level layer of functionality. Users only need to provide the input settings and essential code and / or use some of the many implemented imperfections and algorithms. It can be especially useful for first-time users. Currently the following accelerators are implemented: ATF2, ILC, CLIC and FACET. This paper discusses the framework design and shows its strength in some condensed examples.

INTRODUCTION

When simulating linear accelerators or transport lines, one encounters repeating tasks that are basically the same for each simulation such as:

- Setting up the model of the beamline and the beam
- Specifying and saving simulation parameters
- Implementing a scripting structure to simulate typical scenarios
- Implementing of correction techniques
- Implementing imperfections such as ground motion and component imperfections
- Maintaining scripts to ease computing on a server farm
- Evaluating the results
- Performing backups and keeping track of changes

All these tasks are usually repeated for each simulation project that is started. As a result, there are a large number of implementations of very similar code in each beam physics team, which reduces the productivity significantly.

Instead of this approach, a unified simulation framework for linear accelerators based on PLACET [1, 2] and GUINEA-PIG [3], named LinSim, is described in this paper. It automates the mentioned tasks and takes a large (re)implementation burden away from the user. The user needs only to specify simulation settings and to add the minimal specific code for the given problem.

Additionally, many correction techniques (e.g. orbit feedbacks, IP feedbacks, dispersion free steering, wakefield alignment, and many other) are already implemented or need to be implemented only once, which increases productivity and leads to well debugged code.

Furthermore, LinSim includes more features than would usually be written, e.g. consistency input parameter checks, automated settings storing to be able to reproduce the results at a later stage, revision control, scripts for analysis of results and many others. This helps to increase the productivity of the user. Especially (but not only) for newcomers, LinSim is an enormous starting help, since they don't have to know all the details of the already provided simulation scripts.

FRAMEWORK STRUCTURE

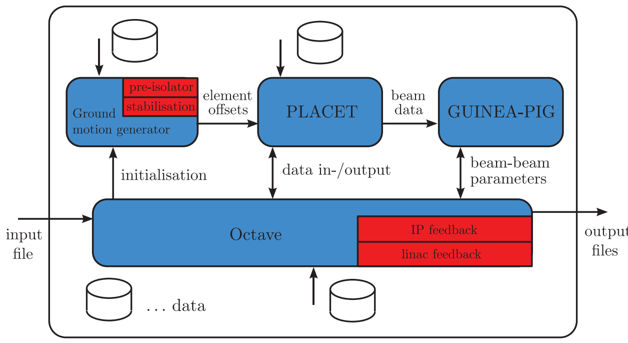

Figure 1: Internal structure of LinSim, where scripts (written in Tcl and Octave) interface the simulation codes PLACET, a ground motion generator, and GUINEA-PIG. An input file is used to control the simulations that use additional data such as lattice files and created standardised output files.

The structure of the framework is illustrated in Fig. 1. It consists of a set of scripts written in Tcl [4] and Octave [5] that interface different simulation codes: PLACET is used for the beam tracking, a ground motion simulator [6] (which has been ported to C++ and is now included in PLACET) generates realistic element misalignments, and GUINEA-PIG facilitates beam-beam simulations. Several types of imperfections are implemented and can be turned on and off. Also, many algorithms for the correction of these imperfections, e.g. beam-based alignment and orbit feedbacks, have been put in place. LinSim also provides the lattice, beam and additional information for the accelerators ATF2 [7], CLIC [8], FACET [9], and ILC [10].

SIMULATION STRUCTURE

A typical simulation consists of two components as depicted in Fig. 2. The first part is a set of provided scripts that make up LinSim itself, together with provided lattice files and additional data as, e.g. orbit response matrices. The entry point to LinSim is the script `run.tcl`, which is executed with PLACET. The second element of a simulation is the user specific test, which consists of two files: a settings file and a code file (e.g. `test_settings.tcl` and test_code.m). These files control the execution of LinSim and therefore determine the specific simulation. In the settings file, parameters and settings can be specified that overwrite the initial ones. In the code file, code can be added that is linked into the execution of the framework at several different points to make LinSim as flexible as possible.

The structure of each simulation can be divided in four parts depicted with different colours:

- Initial setup (yellow): setup_mode
- Seed loop (orange): static_mode
- Long-term loop (blue): long_term_mode
- Short-term loop (green): short_term_mode

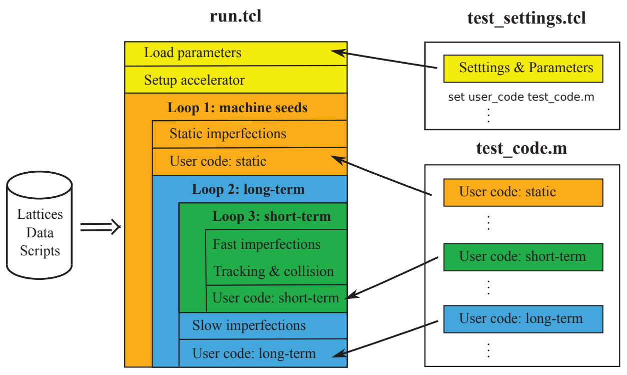

Figure 2: Structure of a simulation with LinSim (entry point is run.tcl), which used external data and is controlled via two test file, e.g. test_settings.tcl and test_code.m. Code from test files is linked into four different parts of the execution, which increases the flexibility significantly.

Note that most simulations will only consist of a subset of these four parts as, e.g. initial setup and short-term loop. The specific structure depends on the parameter choice. These four parts give the user a flexible simulation environment. The first part of LinSim, the initial setup, will always be executed (yellow in Fig. 2). It consists of loading the initial settings and overwriting them with the user specific ones. Then, the accelerator model and the according beam(s) are set up. There are many parameters related to the setup of the accelerator model, which are explained in the manual [11]. Here only a few important parameters are discussed necessary for the following explanations and examples:

- machine_name: The accelerator to be simulated: ATF2, CLIC, FACET, or ILC.
- use_only_one_arm: For a collider (CLIC and ILC), usually the e^- and the e^+ parts are simulated and the created beam can be used for beam-beam simulations. If the parameter is 1 then only the e^- arm is simulated. The beam can still be collided with itself.
- use_main_linac: The main linac of the accelerator is used (values 0 or 1).
- use_bds: The beam delivery system of the accelerator is used (values 0 or 1).
- use_beam_beam: The beam-beam simulations are performed (values 0 or 1).

After the initial setup, the code enters the first of the three interleaved loops. This first loop (orange in Fig. 2) iterates over a number of seeds. Each seed corresponds to a different setup of the imperfections. Then, the user specified code in the section ”User code: static” of the test file is executed. This code allows to create user-specific initial imperfections.

As a next step, the execution enters a double loop consisting of a short-term loop (green in Fig. 2) and a long-term loop (blue in Fig. 2). Each step of the short-time loop corresponds to one beam tracking. The number of iterations can be controlled via the parameters nr_time_steps_short. Within each step of the short-term loop the following activities are performed:

- Apply the specified dynamic imperfections, e.g. ground motion
- Track the beam(s) through the accelerator
- Collide the beams (if specified)
- Apply already implemented correction methods (if specified)
- Execute the user code in the test file under ”User code: short-term”

The short-term loop is the right place to test train-to-train effects and systems as orbit feedback systems and system identification schemes.

Since simulating all bunch trains over a long time is computationally expensive, it is often not possible to simulate effects on larger time scales in full detail. Therefore, the long-term loop can be used. This loop starts with the creation of imperfection that corresponds to the time specified in the variable delta_T_long. Then the short-term loop is executed, and finally the user specific code in the test file under ”User code: long-term” is performed. The long-term loop is repeated nr_time_steps_long times. It is the right place to test long-term effects such as ground motion and mitigation methods such as dispersion free steering and IP feedback.

USAGE

LinSim is started in a terminal with the command

```
placet run.tcl test_settings.tcl
```

As can be seen, the code PLACET interprets the script run.tcl. The script run.tcl reads the test settings file test_settings.tcl in which the user has specified the simulation settings in Tcl language. Note that only the test settings file is passed, since the test code file is specified in the test settings file in the variable user_code. The code file, in Octave language, consists of the three sections specified in Fig. 2. E.g. the first part of code corresponds to initial misalignments, static imperfections, and initial correction methods, and has to be filled between the lines

```
if(sim_mode == static_mode)
    ...
end
```

SIMPLE EXAMPLE

This example simulates a luminosity scan over different roll settings of the last quadrupole QD0 for CLIC. The settings file `qd0_rollscan_settings.tcl` is (with comments preceded by a %):

```
% code file
set user_code "tests/qd0_rollscan_code.m"
% machine
set machine_name "CLIC"
% scan values in microrad
set values_to_scan
    {-100 -75 -50 -25 0 25 50 75 100}
% number of steps, equal to values
set nr_time_steps_short
    [llength $values_to_scan]
% nr long time steps (long-term loop not used)
set nr_time_steps_long 1
% specify beam line and beam beam interaction
set use_main_linac 0
set use_bds 1
set use_beam_beam 1
% output directory
set dir_name "QD0_rollscan"
```

The code file `qd0_rollscan_code.m`, in Octave language, consists of the following self-explaining PLACET commands:

```
% static mode, store initial roll
if (sim_mode == static_mode)
qd0_roll_start =
   placet_element_get_attribute(
   'electron', index_qd0(electron), 'roll')
end

% short time mode, apply new roll setting
if (sim_mode == short_term_mode)
% new roll setting
qd0_roll =
   values_to_scan(time_step_index_short) +
   qd0_roll_start
% apply to beamline
placet_element_set_attribute(
   'electron', index_qd0(electron),
   'roll', qd0_roll)
end

if (sim_mode == long_term_mode)
% nothing to be done here
end
```

The simulation in PLACET can be started by:

```
placet run.tcl qd0_rollscan_settings.tcl
```

The output data can be plotted with one of the Python scripts that are provided:

```
import TrackingAnalysis
a=TrackingAnalysis.MeasurementStation
   (directory="../QD0_rollscan/")
a.lumiScanPlot(-100,100,25,label=
   'QD0 Roll scan [$\mu$rad]',
   plotname='QD0Roll')
```

And finally, the output is shown in Fig. 3, which shows the CLIC peak luminosity in $cm^{-2}s^{-1}$ as a function of the roll of the last focusing quadrupole QD0.

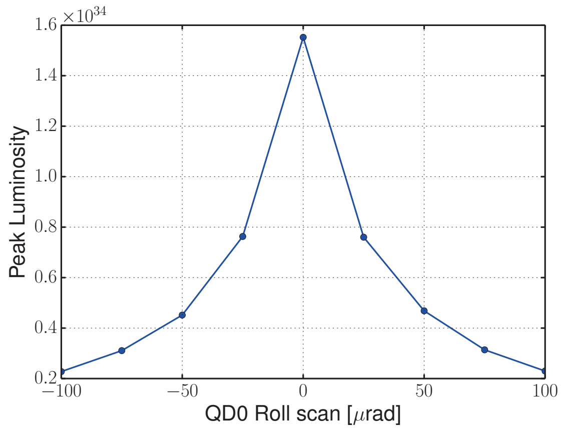

Figure 3: Peak luminosity in $cm^{-2}s^{-1}$ versus QD0 roll.

INSTALLATION

To be able to use LinSim, it is necessary to install PLACET. If beam-beam simulations should be performed, also GUINEA-PIG has to be available. The installation instructions for PLACET and GUINEA-PIG can be found in reference [1] and in the LinSim manual [11]. The installation for LinSim as well as a comprehensive overview is described in the manual [11]. The manual includes a list of the implemented algorithms and imperfections, an explanation of the input parameters, a list of the variables available for usage in user code files, the description of the output files, an overview of the analysis scripts, and some additional examples. Furthermore, a chapter is provided on how to add a new linac, apart from the four already implemented, to the LinSim framework.

REFERENCES

[1] D. Schulte *et al.*, “The PLACET project", CERN, http://clicsw.web.cern.ch/clicsw

[2] A. Latina *et al.*, “Evolution of the tracking code PLACET”, in *Proceedings of IPAC, Shanghai, 2013*, p. 1014.

[3] D. Schulte, “Study of Electromagnetic and Hadronic Background in the Interaction Region of the TESLA Collider", PhD thesis, Universität Hamburg, 1996.

[4] J. K. Ousterhout, “Tcl and the Tk Toolkit". Addison-Wesley Professional Computing Series, 1994. ISBN: 0-201-63337-X.

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE029

[5] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, "GNU Octave: a high-level interactive language for numerical computations", https://www.gnu.org/software/octave/doc/interpreter

[6] Y. Renier, P. Bambade, and A. Sery. "Tuning of a 2D ground motion generator for ATF2". Technical Report LAL/RT 08-18, CARE/ELAN document-2008-005, ATF-08-10, LAL, 2008.

[7] ATF2 Proposal, KEK Report 2005-2.

[8] M. Aicheler, P. Burrows, M. Draper, T. Garvey, *et al.*, "A Multi-TeV Linear Collider Based on CLIC Technology," (2012).

[9] "FACET User Facility", http://facet.slac.stanford.edu

[10] T. Behnke, J. E. Brau, B. Foster, J. Fuster, M. Harrison, *et al.*, "The International Linear Collider Technical Design Report - Volume 1: Executive Summary, (2013), arXiv:1306.6327 [physics.acc-ph].

[11] LinSim manual, http://clicsw.web.cern.ch/clicsw/LinSim/LinSim.pdf