LIAR - A NEW PROGRAM FOR THE MODELLING AND SIMULATION OF LINEAR ACCELERATORS WITH HIGH GRADIENTS AND SMALL EMITTANCES*

SLAC-PUB-7300

R. Assmann, C. Adolphsen, K. Bane, T.O. Raubenheimer, R. Siemann, K. Thompson

Stanford Linear Accelerator Center, Stanford, CA 94309, USA

Abstract

Linear accelerators are the central components of the proposed next generation of linear colliders. They need to provide acceleration of up to 750 GeV per beam while maintaining very small normalized emittances. Standard simulation programs, mainly developed for storage rings, do not meet the specific requirements for high energy linear accelerators. We present a new program LIAR ("LInear Accelerator Research code") that includes wakefield effects, a 4D coupled beam description, specific optimization algorithms and other advanced features. Its modular structure allows to use and to extend it easily for different purposes. We present examples of simulations for SLC and NLC.

1 THE LIAR PROJECT

The computer program LIAR is a numerical simulation and tracking program for linear colliders. The LIAR project was started at SLAC in August 1995 in order to provide a computing and simulation tool that specifically adresses the needs of high energy linear colliders. LIAR is designed to be used for a variety of different linear accelerators. It has been applied for and checked against the existing Stanford Linear Collider (SLC) as well as the linacs of the proposed Next Linear Collider (NLC). It is anticipated that LIAR will be applied to a broad range of problems for the various linear colliders that are studied at this time. Interested accelerator physicists from all laboratories are invited to join the LIAR project and both to use the program and to contribute new features. In this paper we point out some concepts and highlights of the program. A complete description is found in [1]. The physics concepts are described there in detail.

2 COMPILER AND COMPUTER REQUIREMENTS

The LIAR code is mainly written in standard FORTRAN 77. It, however, takes advantage of the STRUCTURE and RECORD extensions that are available in most FORTRAN compilers. The code is stand-alone apart from a few system calls (random number generator, time) that might need to be adjusted to the actual computer system. No specific libraries are required for the compilation. The present LIAR version 1.4 is compiled and run under AIX 3.2 on IBM workstations. Porting to other UNIX platforms is a straight forward process, as long as FORTRAN compilers are available that support the STRUCTURE and RECORD extensions. Porting to non-UNIX platforms might require minor changes to the file handling. The computer should be powerful enough to handle large blocks of memory (≈64 MByte).

3 INTERNAL DATA STRUCTURES

The internal data structures are designed in such a way that easy data access is guaranteed. An INCLUDE statement like

```
INCLUDE 'lattice.inc'
```

will give access to all beamline information, including names, s-locations, magnetic fields, Rf phases, gradients, misalignments, Twiss parameters, beam offsets, emittances and many more quantities.

Let's consider an easy example. The beamline is defined in a structure ELEMENT that has pointers to the BPM structure. The following loop would print the vertical emittance at the BPM's for the whole beam and the first bunch:

```
    DO I = 1, NUM.ELEMENT
      IF (ELEMENT(i).IS_BPM) THEN
        ibpm = ELEMENT(i).POINTER
        WRITE(outlun,*)
1               BPM(ibpm).YEMITALL,
2               BPM(ibpm).YEMIT(1)
      ENDIF
    END DO
```

The emittance information at the BPM's is updated automatically during each tracking. Another hypothetical example is to misalign all quadrupoles vertically by 1% of the vertical beta function:

```
    DO I = 1, NUM.ELEMENT
      IF (ELEMENT(i).IS_QUAD) THEN
        iquad = ELEMENT(i).POINTER
        QUAD(iquad).DY =
1         0.01 * TWISS(i).BETAY
      ENDIF
    END DO
```

During the next tracking the new quadrupole misalignments will be used. Those examples illustrate how easy it is to access and manipulate the internal beamline information without a detailed understanding of other parts of the program. Complete reference information about the internal data structure can be found in [1].

* Work supported by the Department of Energy, contract DE-AC03-76SF00515

Work supported by Department of Energy contract DE–AC03–76SF00515.

4 THE LIAR COMMAND LANGUAGE

The goal to make LIAR easily extendable required a modular structure of the different parts of the program. It should be possible to add subroutines (commands in the LIAR language) without major modification to other parts of the program. Subroutine parameters are therefore passed through a command language. The main program knows only the name of a subroutine and passes the control with whatever parameters have been defined. It is the task of the subroutine to process and use the parameters properly. When the LIAR program is started without command line parameters then it enters an interactive mode:

```
 ___      ___
|||      |||
|||      ___   ______    ____
|||      |||  _||||||   _||||
|||      |||  |||  |||   |||
|||____   |||  |||__|||_  |||
||||||   |||  ||||||||   |||    Version 1.4
-----------------------------------------------
L I near   A ccelerator   R esearch  code

S L A C   1996    ---    Ralph Assmann
                         Karl Bane
                         Tor Raubenheimer
                         Kathy Thompson


LIAR>
```

and commands can be entered. Alternatively LIAR can be run in batch mode, reading its input from a file. Let's consider the example of a simple SLC simulation. We go through it in steps, explaining the purpose of every part. For most commands many more parameters than shown are available. However, if parameters are not specified explicitely, then they are set to their default values. Command lines are automatically continued if an input line is ended with a comma.

In the first two commands we initialize all internal data structures to zero, we switch off the debug level and we set the output unit to the standard output:

```
RESET, all
SET_CONTROL, debug   = 0,
             outlun  = 6
```

Then we read the lattice from a transport input deck and define the injection energy:

```
READ_TRNS, infile  = 'input/slc.trns',
           energy  = 1.19
```

In the next step the Rf-phases for BNS damping are defined and the lattice is rescaled accordingly. Note that two different Rf-phases are specified along the linac with the switch at 800 m:

```
SET_RF, energy   = 50.,
        scale    = .t.,
        phase1   = 22.,
        lswitch1 = 800.,
        phase2   = -16.5
```

Next we calculate the Twiss parameters, specifying the initial values:

```
CALC_TWISS, betax   = 3.40,
            betay   = 3.08,
            alphax  = 0.156,
            alphay  = 0.066
```

After that we need to define the initial conditions for the beam setup. We define an initial beam offset y, the injection energy and energy spread, the normalized emittances and the Twiss values:

```
SET_INITIAL, y        = 200.d-6,
             energy   = 1.19,
             espread  = 0.014,
             nemitx   = 3.0d-5,
             nemity   = 0.3d-5,
             betax    = 3.40,
             betay    = 3.08,
             alphax   = 0.156d0,
             alphay   = 0.066d0
```

Everything is prepared now to set-up the beam. We specify the bunch population, the bunch length, the number of bunches, the number of slices per bunch, the number of mono-energetic beam ellipses per slice and the bunch spacing:

```
SET_BEAM, current  = 4.d10,
          blength  = 1100.d-6,
          nb       = 1,
          ns       = 20,
          nm       = 3,
          bspace   = 0.
```

Next we have to define the wakefields. They are read in from design specific input files. We only define short-range wakefields:

```
SET_SR_WF, file = 'input/srwf_slc.dat'
```

Before we track the beam through the lattice, we misalign all quadrupoles vertically by 200 μm rms:

```
ERROR_GAUSS_QUAD, name     = '*',
                  y_sigma  = 200.e-6
```

Now we can finally track the beam through the lattice:

```
TRACK
```

and measure the normalized beam offsets at the BPM's:

```
MEAS_BPM, file = 'output.data',
          norm = .t.
```

The output is saved into a file. As already mentioned many more parameters and commands are available in LIAR. For example the command TRACKC would have tracked the beam through the lattice while applying a 1-to-1 trajectory correction. The complete reference information is available in the LIAR manual [1].

During the execution of the commands LIAR provides extensive information. For example the progress of the tracking is indicated by a “trackometer”:

```
T R A C K O M E T E R :

0    10    20    30    40    50    60    70    80    90    100 %
|____|____|____|____|____|____|____|____|____|____|
^^^
```