Beam Physics Note
TRI-BN-20-06R
April 15, 2020

A Quick TRANSOPTR Primer

Olivier Shelbaya

TRIUMF

Abstract: This document is intended as a quick primer on elementary use of the beam envelope code TRANSOPTR for new users. An example use case of TRANSOPTR is presented, allowing for a brief overview of how the code is employed and what its output looks like today.

4004 Wesbrook Mall, Vancouver, B.C. Canada V6T 2A3 Tel: 604 222-1047 www.triumf.ca

Introduction

I intend this document to serve as a quick reference for new users to TRANSOPTR, myself having become more familiar over the past three years. This is not an in-depth overview of the theory behind TRANSOPTR's (optr's) physics or operating principles, which can be found in the literature, including [1, 2, 3, 4]. Unlike commercial codes who offer an easy to use visual interface, TRANSOPTR has by both its nature and history remained entirely interface free. This imposes upon the beginner a steeper learning curve and also requires familiarity with Unix-like operating systems.

On the other hand, its spartan nature means users are able to customize it as they see fit, and expand it if necessary. While the learning curve for optr is quite steep, the proficient user has at his disposal a tool which is equally lightweight as it is powerful. The absence of complicated user-experience driven interfaces makes the code particularly versatile and has undoubtedly allowed for its continued development and relevance over the last 40 years at TRIUMF, since the original inception at Chalk River Nuclear Laboratories.

Finally, it is emphasized that TRANSOPTR is written in FORTRAN. Consequently the user should be aware of variable naming conventions in the code and in particular the sensitivity between real (floats) and integers. Starting variable names with certain letters also implicitly assigns the variable type, so caution should be exercised.

Local TRANSOPTR Installation

There are two ways to use optr: old-school and HLA; this document covers the former. For the record, the HLA (high-level application) use of TRANSOPTR requires access to and configuration of the TRIUMF gitlab service on the local filesystem, which can be obtained via the TRIUMF HLA team. TRANSOPTR's source is in FORTRAN and a proposed installation consists of the following:

```
oshelb@oshelbx1:~/hla/transoptr$ pwd
/home/oshelb/hla/transoptr
oshelb@oshelbx1:~/hla/transoptr$ ls -lh
total 60K
-rwxr-xr-x  1 oshelb oshelb  621 Mar  7 11:20 default.gnu
-rwxr-xr-x  1 oshelb oshelb 1.8K Mar  7 12:25 envelope.gnu
drwxr-xr-x 15 oshelb oshelb 4.0K Mar  7 11:20 example
-rw-r--r--  1 oshelb oshelb 3.3K Mar  7 11:20 readme.md
-rwxr-xr-x  1 oshelb oshelb 6.6K Mar  7 11:20 runoptr.sh
drwxr-xr-x  2 oshelb oshelb 4.0K Mar 18 15:29 src
-rw-r--r--  1 oshelb oshelb  29K Mar  7 11:20 VERSION_LOG.txt
oshelb@oshelbx1:~/hla/transoptr$
```

Figure 1: Default TRANSOPTR installation. Note that I'm showing the git-controlled version, which I just said we weren't discussing. The installation is identical in either cases.

A tarball with the above can be obtained upon request from the TRIUMF Beam Physics

group. optr has the following dependencies:

```
gnuplot
gfortran
getopt
```

For Mac users:

```
‘xcode-select --install‘
‘xcodebuild -license‘
```

An optr environment variable should be defined. As an example, on a bash shell in `.bashrc`:

```
OPTRDIR=/path/to/transoptr/directory
export OPTRDIR
```

It should then be possible to run optr from the command line by defining an alias:

```
alias optr=$OPTRDIR/runoptr.sh
```

Note that running optr with the flag `-c` re-compiles all of the source code (then runs it). The source files are found in the subdirectory `src/` shown in fig. 1. On first execution, you must use `-c`, same if you've modified the source code: you'll need to compile once to see the changes take effect.

Example Driven `TRANSOPTR` Notes

The goal of this exercise is to image the output mass selection slit of the ISAC Offline Ion Source (OLIS) dipole magnet, aiming to produce a certain transverse twiss parameter match criterion at the OLIS emittance rig. There are 6 quadrupoles in this example, IOS:Q1 to Q6, however we're only going to be changing Q4, Q5 and Q6. As such, we assume that a well defined tune already exists, and will not cover principles of beam optics - we're only trying to make the beam do something specific.

We assume in this example that the user has installed optr as discussed and has made a separate folder in which to work:

```
mkdir /home/oshelb/optr_example/
```

And obtained (or better yet wrote!) two files: `data.dat` and `sy.f` - the system file representing OLIS, shown in Appendix A.

```
oshelb@oshelbx1:~/optr_example$ pwd
/home/oshelb/optr_example
oshelb@oshelbx1:~/optr_example$ ls -lh
total 8.0K
-rw-r--r-- 1 oshelb oshelb  959 Apr  2 11:19 data.dat
-rw-r--r-- 1 oshelb oshelb 2.3K Apr  2 11:19 sy.f
oshelb@oshelbx1:~/optr_example$
```

Figure 2: Working directory with both requisite optr files: `data.dat` and `sy.f`.

As this document is a primer, no further discussion of the elements in `sy.f` is presented. An overview of the code's system elements - subroutines that represent the sequential optical elements in the beamline - can be found in [3].

`data.dat`, the input datafile, contains all simulation starting parameters and element setpoints that optr will use for its computation. **Note: line numbers on the left and dotted lines added here for reference:**

```
1...............0.0612 0.0 0.0 27944.8 1.0 1.6e-19 !
2...............-1 5 0.01 0.0001 !
3...............0 0.0 1.0 0.0 !
4...............0.16 0.03125 0.16 0.03125 0.0 3.4e-11 !
5...............1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 !
6...............2
7...............1 2 0.92 3 4 0.92
8...............6
9...............1788.0     0.0        10000.0    0 ! IOS:Q1:POS:VOL QE1 V
10..............3432.0     0.0        10000.0    0 ! IOS:Q2:POS:VOL QE2 V
11..............4413.0     0.0        10000.0    0 ! IOS:Q3:POS:VOL QE3 V
12..............4959.0     0.0        5000.0     0 ! IOS:Q4:POS:VOL QE4 V
13..............3475.0     0.0        5000.0     0 ! IOS:Q5:POS:VOL QE5 V
14..............0.0        0.0        5000.0     0 ! IOS:Q6:POS:VOL QE6 V
15..............0.001 20
16..............10 0.0 0.95 20
```

Commenting in the file is permitted: everything to the right of the ! character is ignored by optr. A line-by-line summary of the parameters is presented. All inputs are assumed to be real numbers (float), unless specified as an integer (int). Floats and integers should not be mixed.

data.dat line 1:

```
[energy (MeV)], [momentum], [b*rho], [mass (MeV/c^2)], [chargestate], [bunch-charge(C)/Cur (A)]
```

Line 1 contains mass, momentum, energy and charge. We note that in this example, items 2 and 3 are 0 with the input energy specified. Regarding the final parameter, if optr is run in mode-5 (see next line), the user specifies bunch charge in C, any mode below 5 expects a current in A. Both cases are for space charge computation.

data.dat line 2:

```
[iprint (int)], [IVOPT (int)], [initial Runge-Kutta stepsize], [RK error tolerance (relative)]
```

The integer iprint, when set to -1, produces the so-called `TRANSOPTR` slim output, featuring a consolidated output beam envelope file: `fort.envelope`. If iprint $>$ 0, the original optr output file structure is featured, which is left to the reader to explore, notably by having a look at `$OPTRDIR/src/main.f`. The Runge-Kutta engine used to solve the envelope equation features an adaptive step size scheme which is based on the Kutta-Merson method, whose error estimate is compared to the user specified value for the integration. More details on optr's RK engines can be found in [5].

Further, note that when `iprint` is a positive integer, setting it to either of the following produces output translations of the `TRANSOPTR` sequence into other codes. That is, if `iprint` =

1. output to `TRANSOPTR` system file
2. output to `GIOS`
3. output to `COSY`
4. output to `TRANSPORT`

data.dat line 3:

```
[bool (int)], [s-offset (cm)], [s-units(1=cm)], [Bs units(1=Kg)]
```

The initial boolean allows for the activation of an external s-oriented magnetic field, where s- is the Frenet-Serret accumulated arclength of the reference particle. **Parameter2 is the**

starting offset of the simulation's longitudinal co-ordinate, independent of parameter-1 setting. The two last floats allow for unit conversion, for example imperial. If Bs is enabled, the magnetic field is read in from the file `fort.2`, containing tabulated s vs. Bs data, which must be supplied by the user. The field starts at the specified s-offset.

data.dat line 4:

```
[2*x rms], [2*Px rms], [2*y rms], [2*Py rms], [2*z rms (bun. len.)], [2*Pz rms (dp/p)]
```

Line 4 specifies the starting bunch parameters, in terms of the 2rms size of the distribution in the six canonical phase space coordinates. These are the elements σ_{ii} of the beam matrix. The transverse beam dimensions are in units of length and the canonical momenta in angles. Units are specified at line 5.

data.dat line 5:

```
[x dim. (1.0=1cm)], [Px dim. (1.0=1rad)], [y dim. (1.0=1cm)], [Py dim. (1.0=1rad)],
[z dim. (1.0=1cm)], [Pz dim. (dp/p in rad)]
```

This line controls unit definitions. Default dimensions for x,y,z are cm, meaning if the entry in line 5 is set to 1.0. Dimensions for Px,Py,Pz are radians. For instance, if one wishes to use inches for the x-dimension, the first entry in line 5 would read 0.3937, which is the factor (1"/2.54cm). Regarding the longitudinal coordinates z,Pz, we clarify that in `TRANSOPTR` they are, by definition:

$$z = \beta c \Delta t \tag{1}$$

$$Pz = \frac{\Delta E}{\beta c} \tag{2}$$

The base units for Pz, when set to 1.0 produce units in radians.

data.dat line 6:

```
[number of rij correlation params.]
```

data.dat line 7:

```
1(int) 2(int) [r12] 3(int) 4(int) [r34] I J [rij]
```

The correlation coefficients for the I,J elements of the σ-matrix. Note, in the present example, line 7 only contains `r12` and `r34`, the correlation coefficients for x,Px and y,Py, though other I J correlations can be provided, meaning line 6 needs to be updated as well. These parameters are important as they define the orientation of the phase space ellipse containing the rms distribution of the beam.

data.dat line 8:

```
[number of tuneable element parms. (int)]
```

This line tells optr how many elements in `sy.f` read in values from `data.dat`, via the `COMMON` block. The integer on line 8 is the number of lines below occupied by tuneable parameters, whose definition is entirely up to the user. In the example file shown here, the 6 OLIS quadrupoles' voltage setpoints are listed in sequence. Note that parameters from top to bottom in `data.dt` are passed to the `COMMON` block variables from left to right. For instance, in `sy.f`, the block reads as:

```
      COMMON/BLOC1/QE1,QE2,QE3,QE4,QE5,QE6
```

with QE1 to 6 being stored in `data.dat` lines 9 to 14.

data.dat tuneable element lines:

```
[setpt], [min value], [max value], [bool: optimize? (int)]
```

Each tunable element in the file needs a user specified setpoint. In the present OLIS example, as the optr subroutine `EQ` which models electrostatic quadrupoles expects an input voltage in kilovolts, the specified voltages from `data.dat` are scaled by 10^{-3} in `sy.f`'s `EQ` calls, visible in Appendix A. The minimum and maximum values for the setpoint are needed if optr is called to perform an optimization. The optimization call, which will vary the element setpoint to achieve a user-specified match in `sy.f`, is enabled with the final boolean entry on the line, 1 for optimization and 0 for none.

A comment regarding what is meant by *tuneable element*: in `TRANSOPTR`, this does not only mean control system variable such as quadrupole voltage, but can also mean physical parameters such as drift lengths, element lengths, apertures, etc..

data.dat line N-1 and N:

```
[CLIM], [MAXIT]

[METHOD],[TEMPTR],[DEC],[IITER]
```

These last two lines supply parameters for the simulated annealing optimization routine [3]. CLIM is the convergence criterion. If the relative change or the size (whichever is smaller) of the minimization parameter CHI is less than this number it is declared converged. Usually CLIM can be set to 1.E-4 in mode 4 or 5, but can be down to 1.E-7 in mode 3; it is the square root of the precision when doing a Runge-Kutta step. Would be 1.E-8 for double precision (`optr -d`).

MAXIT is the maximum number of iterations. The optimizer will stop if it is exceeded. Usually set to some number between 100 and 1000. For testing, it can be useful to set MAXIT to 1, causing optr to only run once.

METHOD is a flag, which has to be > 0. If it is 10, some simulated annealing parameters are set automatically.

TEMPTR is the starting temperature for the annealing procedure. If set to zero, simulated annealing is not used at all. If it is 1., all free parameters are allowed to wiggle around during annealing by a range equal to the allowed range of the parameter, set in that parameter's input line. If already near optimum, temperature can be set to 0.1, reducing the range. If the optimum setting is unknown, setting the temperature to 10 will maximize chances of finding the minima. This effectively means it goes through several iterations where all optimization parameters are basically set to random values (within their allowed range).

DEC and IITER set the cooling scheme for the temperature: For a given temperature, the parameters are allowed to fluctuate randomly and at each fluctuation, CHI is evaluated by running through `sy.f`. The number of evaluations of CHI at one temperature is IITER, after which the temperature is multiplied by DEC and another iteration started. Regarding their use, quoting Baartman directly [6]:

"Through many many calculations with many different numbers of parameters from 3 to 30, I discovered a rule. If there are lots of free parameters, I think it's clear that you need lots of evaluations or else the parameter space is not properly sampled. It turns out that IITER should be exponential in number of parameters p and I use IITER=2^p. So for 5 parameters, IITER=32. Further, for more parameters, you also need slower cooling. I use DEC=1.-1./IITER. This means for 5 parameters, DEC=1-1/32=0.96875."

"If you set METHOD=10, both DEC and IITER are set to these values above, and the last two entries in the last line of data.dat are ignored."

Running TRANSOPTR

We can now run optr in our local directory, which will give us the OLIS beam envelope, up to the emittance rig. Upon execution, TRANSOPTR computes the 2rms beam envelopes of the system, outputting several files in the working folder, shown below.

```
oshelb@oshelbx1:~/optr_example$ pwd
/home/oshelb/optr_example
oshelb@oshelbx1:~/optr_example$ ls -lh
total 1.6M
-rw-r--r-- 1 oshelb oshelb  959 Apr  2 14:22 data.dat
lrwxrwxrwx 1 oshelb oshelb   39 Apr  2 14:57 envelope.gnu -> /home/oshelb/hla/transoptr/envelope.gnu
-rw-r--r-- 1 oshelb oshelb 1.1M Apr  2 14:57 fort.envelope
-rw-r--r-- 1 oshelb oshelb  490 Apr  2 14:57 fort.label
-rw-r--r-- 1 oshelb oshelb 5.4K Apr  2 14:57 fort.xml
-rwxr-xr-x 1 oshelb oshelb 404K Apr  2 14:56 optr
-rw-r--r-- 1 oshelb oshelb  43K Apr  2 14:57 optr.eps
-rw-r--r-- 1 oshelb oshelb 2.7K Apr  2 14:56 sy.f
oshelb@oshelbx1:~/optr_example$
```

Figure 3: Working directory after optr execution.

In brief, the output files correspond to:

- `fort.label`: output `ascii`-formatted element label vs element position along the beamline for plotting
- `fort.xml`: output XML formatted sequence file (for use with TRIUMF-HLA framework)
- `fort.envelope`: full output beam envelope, including beam energy, reference time, all terms of the σ-matrix and cumulative point-to-point transfer matrix.
- `envelope.gnu`: `gnuplot` script file for beam envelope printing
- `optr.eps`: output `gnuplot` encapsulated postscript (.eps) image file with beam envelope, as defined in `envelope.gnu`

An overview of the output files is given in Appendix B, particularly when running `iprint` $>$ 0 (`data.dat` line 2). The beam envelope from `optr.eps` is shown in fig. 4. The figure is simply plotting values from `fort.envelope`.

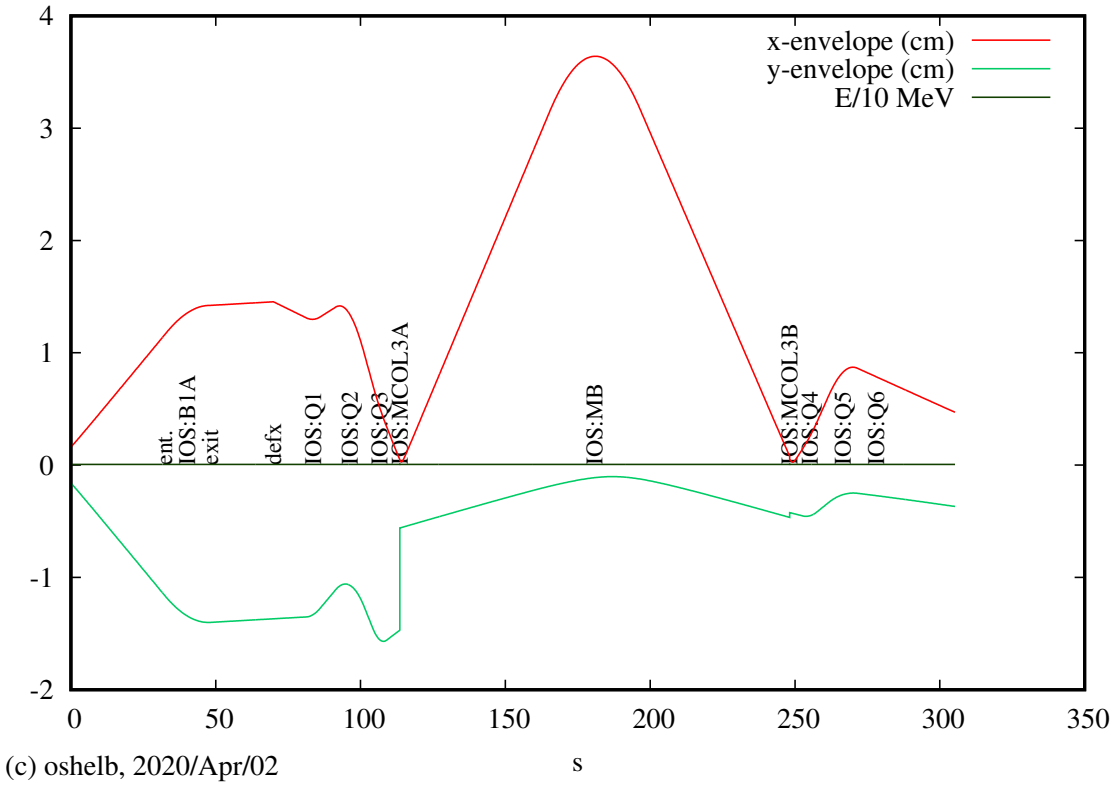

Figure 4: `TRANSOPTR` envelope simulation from the OLIS microwave source up to the OLIS emittance rig. A mass 30, charge 1 beam at 61.2 keV is shown.

`TRANSOPTR` Optimizations

The original goal of this example was to fit twiss parameters at the emittance rig. This is accomplished in two steps: first, fitting functions must be placed in `sy.f`, at the appropriate location s along the beamline. In our example, this is simply the end of the system file. For this case, we're going to use the function `TWISSMATCH`, however the user can also use `fit` or `fitarb` if desired. For this example, since we want to image the slit `IOS:MCOL3B`, we first insert two `TWISSFIND` calls in `sy.f` at the marker `! Slit IOS:MCOL3B`:

```
! Slit IOS:MCOL3B
call TWISSFIND(1,axi,bxi)
call TWISSFIND(3,ayi,byi)
call slit(0.05,0.5,wo,'IOS:MCOL3B')
```

```
! ILT:EMIT
call TWISSMATCH( 1, axi, 100.0*bxi, 1., 1)
call TWISSMATCH( 3, bxi, 100.0*byi, 1., 1)
```

The `TWISSMATCH` call is placed at the commented location of the OLIS emittance rig in `sy.f`. As we've stated, we want to only optimize Q4 to Q6, so the optimize booleans are set to 1 in `data.dat`. Note that we've chosen to magnify the beta functions β_x and β_y. Once these lines are added to the system file and data.dat is set to optimize the quads, we run optr in the command line, and after a few iterations obtain our solution, shown in fig. 5.

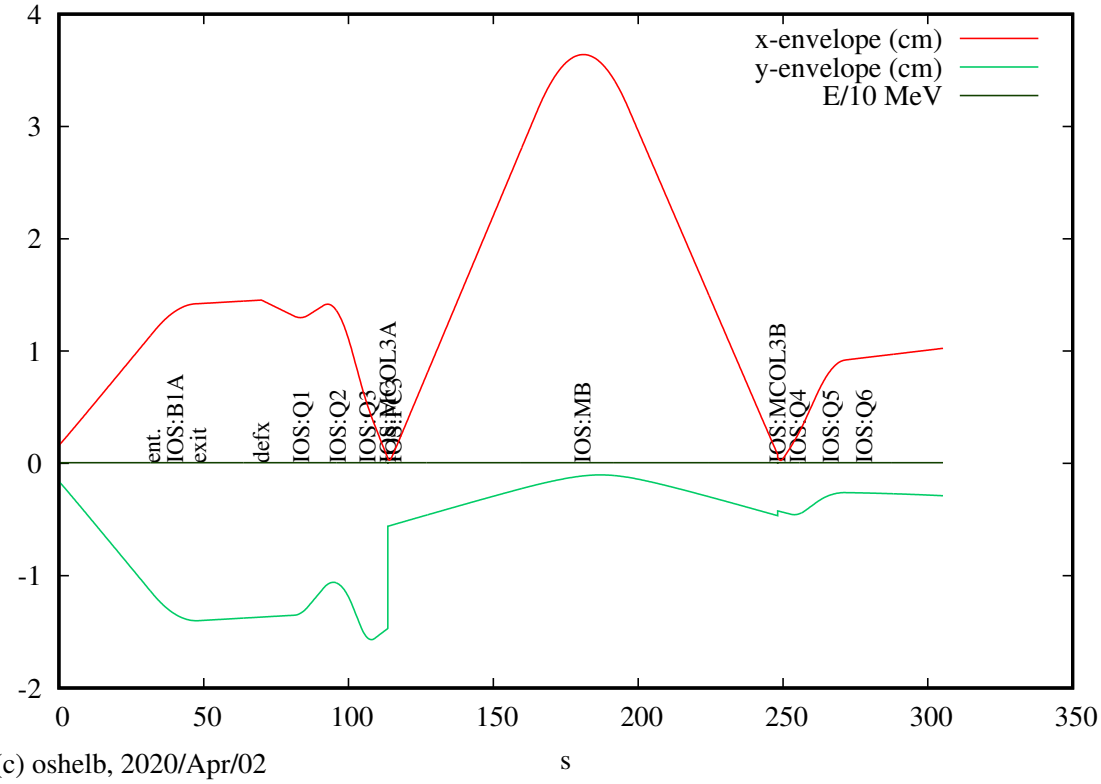

Figure 5: `TRANSOPTR` envelope simulation from the OLIS microwave source up to the OLIS emittance rig. A mass 30, charge 1 beam at 61.2 keV is shown. The slit has been imaged at the OLIS emittance rig following optimization.

Finally, `TRANSOPTR` prints out the optimized parameters after the fit is complete:

```
Runge-Kutta engine= 0, Final step=      0.160cm, EPS= .100E-03
```

```
Symplectic.Error/EPS; Avg.=    0.079, St.Dev.=    0.066, Max. =    0.216

No. of iterations =    2
  20 Final run sqrt(CHI)=  0.1671861E+02
 Fixed Parameter:   1788.
 Fixed Parameter:   3432.
 Fixed Parameter:   4413.
Varied Parameter:   4570.
Varied Parameter:   2632.
Varied Parameter:   3.295
```

The list which is printed to terminal shows each of the specified tuneable parameters and their final values if the optimization converged. In this case, the last three parameters correspond to the voltages for Q4, Q5 and Q6. We note that Q6 can effectively remain unpowered.

Acknowledgements

Many thanks to Rick Baartman, Thomas Planche and Paul Jung for the many discussions about the code and the countless times they've helped me understand! Rick Baartman further supplied the description for simulated annealing, which was edited into the text on July 28, 2020. Appendix C was added on December 4th, 2020.

References

[1] Heighway EA and Hutcheon RM. *Transoptr - A second order beam transport design code with optimization and constraints*. Technical report, Atomic Energy of Canada Limited, 1981.

[2] Sacherer FJ. *RMS Envelope Equations With Space Charge*. Technical report, CERN, 1970.

[3] Richard Baartman. TRANSOPTR: Changes since 1984. Technical Report TRI-BN-16-06, TRI-UMF, 2016.

[4] Baartman R. *Fast Envelope Tracking for Space Charge Dominated Injectors*. In *Proceedings of LINAC 2016 Conference*, pages 1017–21, 2016.

[5] Richard Baartman. Runge-Kutta Customization for TRANSOPTR. Technical Report TRI-BN-18-09, TRIUMF, 2018.

[6] R. Baartman. personal communication.

[7] DV Gorelov, PN Ostroumov, and RE Laxdal. Use of the lana code for the design of a heavy ion linac. In *Proceedings of the 1997 Particle Accelerator Conference (Cat. No. 97CH36167)*, volume 2, pages 2621–2623. IEEE, 1997.

[8] Laxdal RE. *Design Specification for ISAC HEBT*. Technical Report TRI-DN-99-23, TRIUMF, 1999.

[9] R Baartman. Emittance convention. *TRIUMF, Vancouver, Canada, Internal report, TRI-BN-15-07*, 2015.

Appendix A: OLIS sy.f - system file

```
SUBROUTINE TSYSTEM
COMMON/SCPARM/QSC,ISC,CMPS
COMMON/MOM/P,BRHO,pMASS,ENERGK,GSQ,ENERGKi,charge,current
COMMON/BLOC1/QE1,QE2,QE3,QE4,QE5,QE6

CMPS=0.314   ! Number of cm per step, for plotting only
wo=1.0 ! Weight aberration from optical elements

! startOf_olis_transoptr
call drift(11.7623,".")
! IOS:XCB1
call drift(3.887,".")
! IOS:YCB1
call drift(16.5609,".")
! Electrostatic dipole IOS:B1A
call Eedge(25.379,36.0,1.0,0.212,3.81,wo)
call Ebend(25.379,36.0,1.0,'IOS:B1A')
call Eedge(25.379,36.0,1.0,0.212,3.81,wo)
call drift(15.4919,".")
! IOS:YCB1A
call drift(6.272,".")
! Deflector IOS:XCB1A
call deflectx(5.08,9.0,0,0)
call drift(4.607,".")
! endOf_ios_mws
call drift(6.702,".")
! Electrostatic quadrupole IOS:Q1
call Equad(-0.001*QE1,2.54,4.892,wo,'IOS:Q1')
call drift(5.3148,".")
! Electrostatic quadrupole IOS:Q2
call Equad(0.001*QE2,2.54,9.8704,wo,'IOS:Q2')
call drift(2.8178,".")
```

```
! Electrostatic quadrupole IOS:Q3
call Equad(-0.001*QE3,2.54,4.892,wo,'IOS:Q3')
call drift(4.56,".")
! Slit IOS:MCOL3A
call slit(0.05,0.5,wo,'IOS:MCOL3A')
call drift(1.951,".")
call marker('IOS:FC3')
call drift(12.724,".")
! IOS:YCB3
call drift(36.898,".")
! Magnetic dipole IOS:MB
call edge(0.0,29.9887,60.0,0.0,0.317,2.0,5.08,0.0,wo)
call bend(29.9887,60.0,0.0,'IOS:MB')
call edge(0.0,29.9887,60.0,0.0,0.317,2.0,5.08,0.0,wo)
call drift(36.6619,".")
! IOS:YCB4
call drift(14.928,".")
! Slit IOS:MCOL3B
call slit(0.05,0.5,wo,'IOS:MCOL3B')
call drift(4.544,".")
! Electrostatic quadrupole IOS:Q4
call Equad(-0.001*QE4,2.54,4.892,wo,'IOS:Q4')
call drift(4.0688,".")
! Electrostatic quadrupole IOS:Q5
call Equad(0.001*QE5,2.54,9.8704,wo,'IOS:Q5')
call drift(1.5288,".")
! start_q6_tomography
call drift(2.541,".")
! Electrostatic quadrupole IOS:Q6
call Equad(0.001*QE6,2.54,4.892,wo,'IOS:Q6')
call drift(10.218,".")
! IOS:YCB6
call drift(14.564,".")
! ILT:EMIT
call print_transfer_matrix
return
end
```

Appendix B: Original TRANSOPTR FORTRAN Output Files

optr will output the following files if the first integer in `data.dat` line 2 is greater than 0.

fort.1

```
[label] [s (cm)] [x-envelope (cm)] [y-envelope (cm)] ...
```

2rms envelope output computation file, see `main.f` for full sequencing of elements. The file also contains the transfer matrix. We note that `fort.3` contains the element positions in 3D space.

fort.8

System file (`sy.f`) translation into other optics codes. For `data.dat` line 2, entry 1 (see `iprint`) equal to positive integer:

1. output to `TRANSOPTR` system file
2. output to `GIOS`
3. output to `COSY`
4. output to `TRANSPORT`

fort.81, fort.83

Both files contain output Twiss parameters for the x,y envelopes, respectively.

fort.9x

The files `fort.91`, ..., `fort.96` are generated and represent the ith transfer matrix row elements. For instance, `fort.91` contains the elements M11, M12, M13, M14, M15, M16 at each point in the computation.

Appendix C: Conversion of Longitudinal Emittance Units

This appendix is intended to make clear the conversion between longitudinal emittance units, in particular from energy-time to cm·rad, as used in TRANSOPTR. The original LANA simulations from Gorelov *et al.* [7], which were used in Ref. [8], make use of the former convention. For more details, see [9]. The longitudinal emittance is quoted as ϵ_z = $1.6\pi\mathrm{keV/u}\cdot\mathrm{ns}$, while the Twiss parameter β_z is in $\mathrm{deg}/\%$ for 105MHz. The conversion to cm, rad is performed as follows, where we assume a mass of 22u and an energy of 200 keV/u. We first note the use of π in the quoted emittance, which we disregard from the computation. We first ccompute the relativistic β factor for the 200 keV/u case:

$$K = (\gamma - 1)mc^2 \tag{3}$$

$$(\gamma - 1) = 2.147 \times 10^{-4} \tag{4}$$

$$\beta = 0.0207 \tag{5}$$

The emittance is then normalized by the full relativistic momentum of the reference particle:

$$\epsilon_z = 1.6\pi\frac{\mathrm{keV}}{\mathrm{u}}\cdot\mathrm{ns} \tag{6}$$

$$\epsilon_z = 35.2\mathrm{keV}\cdot\mathrm{ns} \tag{7}$$

$$\epsilon_z = 35.2\frac{\mathrm{keV}}{\beta\gamma\mathrm{mc}}\cdot\mathrm{ns}\left(\frac{1\mathrm{MeV}}{1000\mathrm{keV}}\right) \tag{8}$$

$$\epsilon_z = 0.0025\mathrm{cm}\cdot\mathrm{rad} \tag{9}$$

We've obtained the final longitudinal emittance by multiplying out the resulting units at Eq. (3). Next, for the Twiss-β_z function, ref. [8] presents the units of β_z in degrees at 105MHz, so we compute the corresponding time duration of one degree at that frequency:

$$\Delta t = \frac{1}{f}\frac{1}{360^\circ} = 2.646 \times 10^{-11}\mathrm{s} \tag{10}$$

we know in TRANSOPTR that phase space coordinate 5 is expressed as $\Delta z = \beta c\Delta t$, so we convert one degree of the above period to a length scale, where β is the relativistic velocity parameter:

$$\Delta z = 0.0207 \cdot 3 \times 10^{10}\frac{\mathrm{cm}}{\mathrm{s}} \cdot 2.646 \times 10^{-11}\mathrm{s} = 0.0164\mathrm{cm} \tag{11}$$

We now know that 1° RF at 105MHz corresponds to 0.0164cm for this particular beam velocity. The β_z conversion from deg/% to cm·rad then proceeds as follows:

$$\beta_z = 78.7\frac{\text{deg}}{\%}\left(\frac{1\%}{10\text{mrad}}\right)\left(\frac{0.0164\text{cm}}{1\text{deg}}\right) \tag{12}$$

$$\beta_z = 0.1291\frac{\text{cm}}{\text{mrad}} = 129.1\frac{\text{cm}}{\text{rad}} \tag{13}$$

We note that α_z is dimensionless and therefore requires no conversion, while γ_z may now be found using the standard Twiss parameter definition:

$$\gamma_z[\frac{\text{rad}}{cm}] = \frac{1+\alpha^2}{\beta_z} \tag{14}$$