OPERATION SOFTWARE FOR COMMISSIONING OF KEKB LINAC PROGRAMMED WITH SAD

N. Akasaka, H. Koiso, and K. Oide
KEK, High Energy Accelerator Research Organization, 1-1 Oho, Ibaraki 305, JAPAN

Abstract

SAD was originally developed for accelerator design. In order to use it as a language for KEKB operation software, Python interpreter and EPICS channel access interface were embedded into it. Tkinter, which encapsulates Tcl/Tk in Python, provides graphical user interface (GUI). Hardware instruments are accessed through EPICS channel access routine. In this paper, some of practical tools used in the commissioning of KEKB linac are presented.

1 INTRODUCTION

SAD (Strategic Accelerator Design) started as a code for accelerator design at KEK in 1986[1]. Its functionality has been increasing since then, which includes: optics matching, particle tracking, Taylor map, spin calculation, *Mathematica*-style functions, EPICS channel access functions and Tcl/Tk[2] interface via Python[3]. Among these, the last three are important for accelerator control: *Mathematica*-style function for programming control logic, EPICS channel access for controlling hardware, and Tcl/Tk interface for GUI. Currently, hardware in KEKB linac is accessed by executing shell commands within SAD, since EPICS is not introduced there yet.

Merits of using SAD for accelerator control are:

1) Unification of model/control/GUI in a single script.
   A large accelerator such as KEKB requires tight relationship between a computer model and observation/control of the beam. SAD ensures direct relationship between them, so that an accelerator physicist or an operator can write a necessary software only with SAD.
2) Immediate and powerful execution of a program.
   SAD is an interpreter language and easy to learn/try programming. *Mathematica*-style functions for list-manipulation, functional operations, symbolic and numerical math, etc., make a program very compact and highly structured.
3) Libraries with object-oriented features.
   Although SAD is not a fully object-oriented language, it still has some of object-oriented features in its libraries. For instance, a beam line is identified as an BeamLine object, which accepts many functions such as Join, Delete, Insert, Times, etc.

Considering the fact that most commissioning tasks need an accelerator model, we choose SAD as the most suitable tool for the commissioning of KEKB according to the first merit stated above. Although implementing accelerator modeling capability of SAD in other interpreter languages such as Java, Python, Tcl/Tk, *Mathematica* etc., might have some possibility, it would require significant time and effort, and we do not think there is enough reason to do that within the very limited time before the commissioning of KEKB. Even when no accelerator model is necessary, SAD does not fall behind with these interpreter languages.

2 ACCELERATOR MODEL

SAD has many functions which are related to accelerator models. Only a small part of them is introduced in this chapter.

2.1 Definition of the beam line

A beam line is defined in SAD as a series of components like drift space, magnets from 2-pole (bend) to 2^{20}-pole, solenoids, and accelerating cavities etc. Magnets and accelerating cavities can be spatially overlapped. Alignment errors (offsets and rotations), fringe field for magnets, position dependence of accelerating voltage for accelerating cavities, can be assigned if necessary. `BeamLine[]`, which takes beam line components as its arguments, creates a beam line object, and `ExtractBeamLine[]` returns the current line objects or a beam line object specified by its argument.

2.1 Optics-related functions

`CALCULATE` command calculates the optics using the current values of the components. SAD accepts this type of non-*Mathematica*-style commands. These are rather historical, but very convenient for interactive use. `Twiss[]` function returns the value of calculated optical functions (α_x, β_x, etc.) at the entrance of a component. `GO` command executes optical matching. The matching conditions can be imposed on both geometry and optical functions by `FIT`, `FIX`, and `FREE` commands.

2.2 Plotting functions

`OpticsPlot[]` function creates a plot of calculated optical functions or any given list of values. The abscissa is the distance in the beam line measured from its entrance. A schematic beam line is automatically drawn below the plot. This drawing can be used for selecting components by binding events to them. When the data is given as a list, it is a nested list of the position or component name in the lattice and the value in the form like:

```
{{SPA1B8, 0},{SPA1C5, 0},.....}.
```

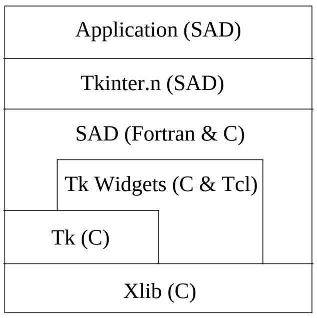


Fig. 1 Levels of graphics processing in SAD.

3 GRAPHICAL PROGRAMMING

SAD outputs graphics on an X-window server. Figure 1 shows the levels of graphics processing.

3.1 *Tcl/Tk*

Tcl/Tk is one of the most widely used toolkits for building GUI. One can easily place common user interface components like buttons or text entries and customize their behavior for the application. Applications can send SAD expressions to another application on the same X-server and receive their result. This is realized by directly calling Xlib functions instead of using send command of Tk, which only sends Tcl commands.

3.2 *Programming style*

In the case of creating a button in a new window, the SAD script would be as follows:

```
w = Window[];
b = Button[w,
      Text->“Hello”,
      Command:>Print[“Hello, World!”]];
```

The result is shown in Fig. 2. The first line creates a window and the next 3 lines create a button whose label is “Hello”. The first argument of `Button[]` is the parent of the button. The button is automatically packed in the parent window `w`. The attributes of the button, `Text` and `Command` in this case, are passed as (delayed) rules. When this button is clicked, it executes the SAD function `Print[“Hello, World!”]` and prints “Hello, World!” on the console. Any SAD function can be specified for `Command` attribute.

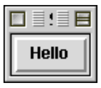


Fig. 2 Hello World example in SAD.

4 EXAMPLES

4.1 *KBFrame*

KBFrame is written to provide a default main window with a menu bar, status line, progress bar, etc. The code below creates a window shown in Fig. 3:

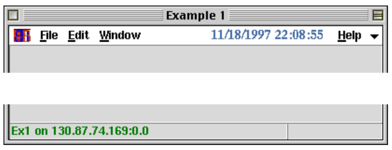


Fig. 3 KBFrame before application-specific components are placed.

```
w = KBMainFrame[
      ”Ex1”,f,Title->”Example 1”];
```

A frame is created with the name `f`, which is used for arranging application-specific components in it. The downward arrow on the right of the menu bar is the task menu, with which one can switch between SAD applications running on the same X-server.

KBFrame provides convenience functions that arrange components without explicit use of geometry managers. This function assumes fixed parameter values for geometry management so that the user doesn’t have to care about it at the expense of some of the flexibility of Tcl/Tk ‘pack’ geometry manager. The components can be arranged in the main application window or in a dialog box created for them.

KBFrame is used in the applications presented below.

4.2 *Klystron status*

Figure 4 is the panel for displaying klystron status in A-C sectors of the linac. When one or more klystrons are tripped, it brings itself on top of the other windows and (optionally) warns the operator with synthesized voice. The voice is generated on a Macintosh with text-to-speech, which reads input strings aloud. UDP packets are used for the communication between SAD and the Macintosh.

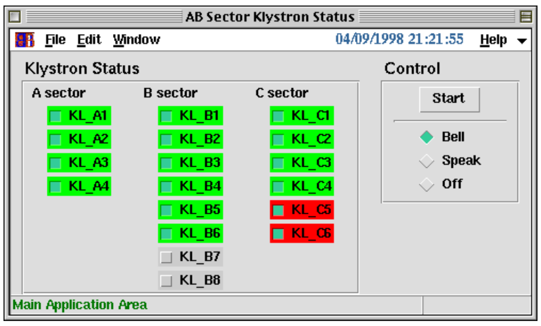


Fig. 4 Klystron status display.

4.3 *BPM display*

Figure 5 shows the BPM monitoring panel. This is an example which uses an accelerator model in SAD. The displayed data from top to bottom are horizontal beam position, horizontal steering strength, vertical beam position, vertical steering strength, and bunch current. The plots and the accelerator line below is drawn by `OpticsPlot[]`. The value of BPM reading can be displayed by clicking on the corresponding data point in the plot.

4.4 *Optics matching and emittance measurement*

Twiss parameters and emittance can be measured from the beam sizes with different strength of a Q-magnet. The panel in Fig. 6 shows the result of fitting of the measured data. Calculated twiss parameters and emittance are displayed at the top of the panel. The measurement conditions (changed Q and observed screen) are specified in another panel by selecting components in the schematic beam line drawn by `OpticsPlot[]`.

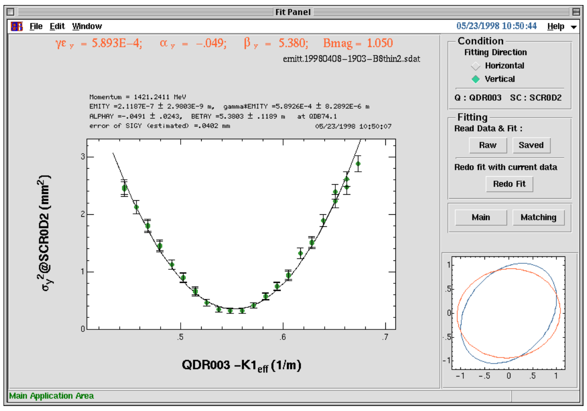


Fig. 6 Emittance calculation panel.

4.5 *Orbit history*

Figure 7 is a display of the fluctuation of BPM readings. The deviation from the time-average of each BPM reading is plotted as a function of time. If BPM readings don't change at all, the displayed plot becomes a perfectly flat plane. This plot is created by a 3D plotting function `ListBirdsEyePlot[]`. SAD has other plotting functions not explained above, which include: `ColumnPlot[]` (bar graph), `FitPlot[]` (fit arbitrary function to given data and plot the result), and `ListPlot3D[]`.

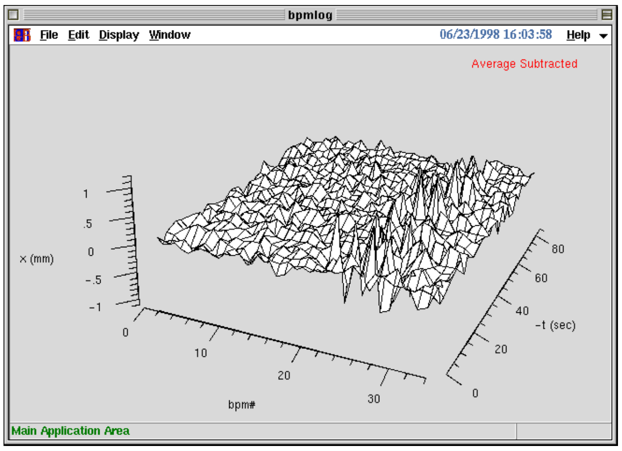


Fig. 7 Orbit history panel.

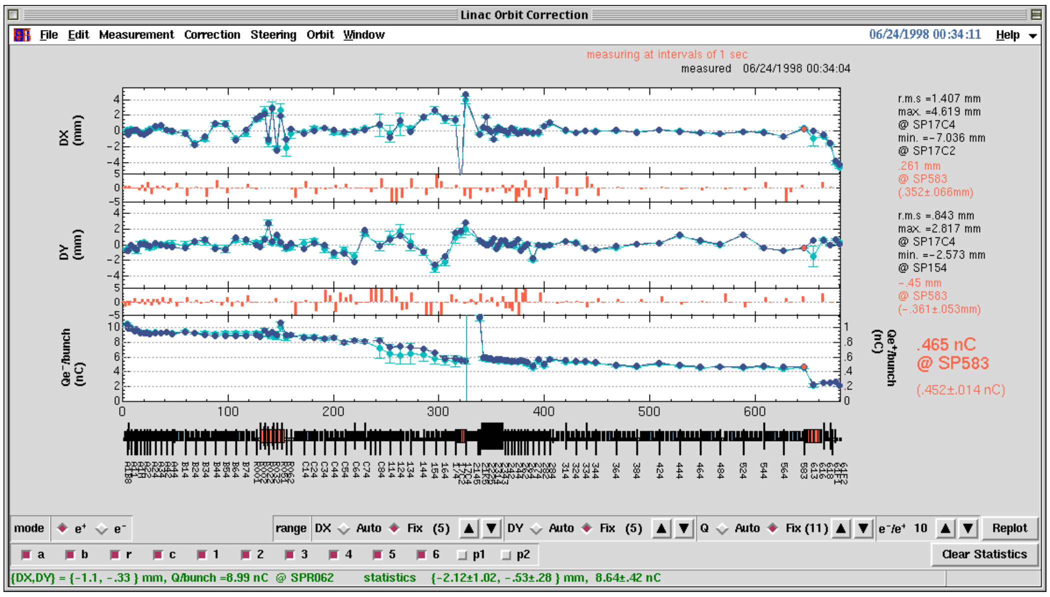
5 ACKNOWLEDGMENT

The authors wish to thank N. Yamamoto for initial implementation of EPICS channel access and Tcl/Tk in SAD, K. Furukawa and N. Kamikubota for preparing shell commands to access hardware, and KEKB linac commissioning group for cooperation.

6 REFERENCES

[1] http://www-acc-theory.kek.jp/SAD/sad.html.

[2] "Practical Programming in Tcl and Tk", Brent Welch, Prentice Hall.



Linac Orbit Correction
File Edit Measurement Correction Steering Orbit Window
06/24/1998 00:34:11 Help
measuring at intervals of 1 sec
measured 06/24/1998 00:34:04
DX (mm)
DY (mm)
Qe⁻/bunch (nC)
r.m.s =1.407 mm
max. =4.619 mm
@ SP17C4
min. =-7.036 mm
@ SP17C2
.261 mm
@ SP583
(.352±.066mm)
r.m.s =.843 mm
max. =2.817 mm
@ SP17C4
min. =-2.573 mm
@ SP154
-.45 mm
@ SP583
(-.361±.053mm)
.465 nC
@ SP583
(.452±.014 nC)
mode e⁺ e⁻
range DX Auto Fix (5) DY Auto Fix (5) Q Auto Fix (11) e⁻/e⁺ 10 Replot
a b r c 1 2 3 4 5 6 p1 p2
Clear Statistics
{DX,DY} = {-1.1, -.33} mm, Q/bunch =8.99 nC @ SPR062 statistics {-2.12±1.02, -.53±.28} mm, 8.64±.42 nC

Fig. 5 BPM display.