Koviz User's Guide

Version 1.0

December 2022

National Aeronautics and Space Administration
Johnson Space Center
Houston, Texas

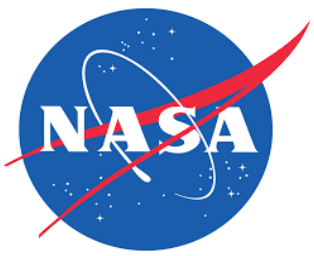

Table of Contents

Introduction

Koviz is a simulation data visualization tool. It is designed especially for Trick monte carlo data analysis, comparing simulation runs, analyzing data spikes and creating report quality plot booklets. Koviz can be run interactively via the GUI or can be run in batch. Koviz supports Trick binary data and CSV. Koviz offers a real-time analysis report for Trick real-time data recordings. Koviz also offers a plugin-like functionality, external programs, to transform simulation data. Koviz can also be synced with video so one can view video alongside associated data. The usage for koviz is:

```
UNIX Prompt> koviz <RUN_dirs|MONTE_dir> [DP_file1...] [options]
```

To see a list of options:

```
UNIX Prompt> koviz <ENTER>
```

RUN Directories

RUN directories contain log files from a simulation run. Koviz is efficient at coplotting multiple RUN directories:

```
UNIX Prompt> koviz RUN*
```

MONTE Directories

Trick Monte carlo directories, typically prefixed by MONTE, contain a set of RUNs and a monte carlo inputs file. The Koviz GUI correlates RUN inputs to plot curves:

```
UNIX Prompt> koviz MONTE_dir
```

DP Files

The data product (DP) specification file syntax is shown below. All keywords are case insensitive. There are three main specifications in the file: PLOTS, TABLES, and PROGRAM. PLOTS refers to one or more pages of XY plots. TABLES refers to one or more pages of ASCII tables. The DP file may contain any number of Pages and Tables. A page may also have any number of plots. PROGRAM refers to an external program – a user created shared library dynamically linked into koviz for manipulating recorded data for display. A DP file can be created from the koviz GUI using the main menu's “Save As DP” option.

DP File Syntax

[Title]

[PLOTS:

PAGE <page_number>: “<page_title>”

[FOREGROUND_COLOR: “#rrggbb|color_name”]
[BACKGROUND_COLOR: “#rrggbb|color_name”]

PLOT <plot_number>: “<plot title>”

[X_AXIS_LABEL: “<x_axis_label>"]
[Y_AXIS_LABEL: “<y_axis_label>”]
[X_MIN_RANGE: <x_min_value>]
[X_MAX_RANGE: <x_max_value>]
[Y_MIN_RANGE: <y_min_value>]
[Y_MAX_RANGE: <y_max_value>]
[PLOT_X_SCALE: “log|linear”]
[PLOT_Y_SCALE: “log|linear”]
[PLOT_RATIO: “square”]
[GRID: <on|off>]
[RECT: <x%> , <y%>, <width%>, <height%>]
[PRESENTATION: <compare|error|error+compare>]
[MAJOR_X_TICS: <comma separated list of floats>]
[MINOR_X_TICS: <comma separated list of floats>]
[MAJOR_Y_TICS: <comma separated list of floats>]
[MINOR_Y_TICS: <comma separated list of floats>]

X_VARIABLE: “<param_name>”

[LABEL: “<var_label>”]
[UNITS: “<var_units>”]
[SCALE_FACTOR: <scale_value>]
[BIAS: <bias_value>]

Y_VARIABLE: “<param_name>”

(Same attributes as X_VARIABLE)
[SYMBOL_STYLE: <symbol_style>]
[SYMBOL_SIZE: <symbol_size>]

[LINE_STYLE: <line_style>]
[LINE_COLOR: “#rrggbb|<color_name>”]

[Y_VARIABLE: ...]

]

[TABLES:

TABLE <table_number: “<table_title>”

VARIABLE: “<param_name>”
(Same attributes as X_VARIABLE in PLOT)
[VARIABLE: ...]

]

[PROGRAM: “<program_name>”:

IN: “<param_name1>” “<param_name2>” ...
OUT: “<param_name1>” <param_name2>” ...

]

DP Line and Symbol Styles

SYMBOL_STYLE:

None|Square|Circle|Star|XX|Triangle|
Solid_Square|Solid_Circle|Thick_Square|Thick_Circle|
Number_[0-9]

SYMBOL_SIZE:

Tiny|Small|Medium|Large

LINE_STYLE:

“No_Line|Thick_Line|X_Thick_Line|Fine_Dash|Med_Fine_Dash|
Long_Dash|X_Long_Dash|Dot_Dash|2_Dot_Dash|3_Dot_Dash|
4_Dot_Dash|Plain|Dash|Scatter”

LINE_COLOR:

“#rrggbb|<color_name>”

DP File Example

Example barebones DP file:

```
PLOTS:

PAGE 1: "Page Title"

    PLOT 1: "Ball X Position"

        X_VARIABLE: "sys.exec.out.time"
        Y_VARIABLE: "ball.state.out.position[0]"
```

Session Files

To launch koviz with a set of RUNs and DPs, the -session command-line option may be used:

UNIX Prompt> koviz -session <session_file>

The syntax for the session file is as follows:

[Title]
[DEVICE: <File <file_name>|Terminal>]
[FREQUENCY: <cycle_time>]
[TIME_MATCH_TOLERANCE: <delta_time>]
[PRESENTATION: <Compare|Error|Error+Compare>]
PRODUCT: <DP_filename>
[PRODUCT: <DP_filename_n>...]
RUN: <RUN_directory>
[RUN: <RUN_directory_n> …]
[T[1-4]: <title1-title4>]
[C[1-7]: <line color 1-7>]
[L[1-7]: <legend label 1-7>]
[LS[1-7]: <linestyle 1-7>]
[S[1-7]: <symbol 1-7>]
[FG: <foreground color>]
[BG: <background color>]
[START: <start_time>]
[STOP: <stop_time>]
[ORIENT: <portrait|landscape>]
[SHIFT: <RUN shift times>]
[TimeName: <time_name>]
[MapFile: <map_filename>]
[LEGEND: <on|off>]
[ShowTables: <on|off>]
[EXCLUDE: <exclude pattern>]
[FILTER: <filter pattern>]
[Video: <filename>]
[VideoOffset: <time>]
[ShowPageTitle: <yes|no>]
[ShowPlotLegend: <yes|no>]
[PlotLegendPosition: <ne|e|se|s|sw|w|nw|n>]

The DEVICE: statement specifies the visualization device for data output. Device types are Terminal (the default) and File. When Terminal is specified, the koviz GUI will be displayed. When File is specified the output is re-directed as PDF to the <file_name> file.

The FREQUENCY: statement specifies the delta time between data points when loading curve data. <cycle_time> is specified in seconds. If the optional FREQUENCY statement is unspecified or if the <cycle_time> is less than the recorded data cycle time, all recorded curve data will be loaded. If the <cycle_time> is larger, curve data points will be culled.

The TIME_MATCH_TOLERANCE: statement specifies the maximum allowed delta between the time stamps for two sets of data points used in generating the error plot. The default value is 1.0e-6. Koviz does a best match for timestamps within the tolerance.

The PRESENTATION: statement is only useful when two RUNs are specified. If presentation is “error”, a difference curve between two RUNs will be displayed. The points in the difference curve are matched by the TIME_MATCH_TOLERANCE. Interpolation is not done. If presentation is “compare”, curves from each RUN will be shown together. Finally, if presentation is “error+compare”, three curves will be shown: the difference curve alongside the curves from each RUN.

The PRODUCT: statement specifies a data product (DP) file. One or more DP files must be specified. DP files specify the simulation parameters to display, and the display attributes for each parameter. See DP Files for more detail.

The RUN: statement specifies a simulation RUN directory from which to retrieve logged data, <RUN_directory>. One or more RUNs must be specified.

The SHIFT: statement is used to shift RUN time(s). If there is a single RUN, specify a double value e.g. SHIFT: 13.56. If there are multiple RUNs, use a comma delimited list of RUN:<shift_value> specs e.g. SHIFT: RUN_a:13.56,RUN_b:77.28,RUN_c:23.0.

The TimeName: statement is used when time is not the default “sys.exec.out.time” e.g. TimeName: myTimeName.

The MapFile: statement is used to tell koviz where the map file is located. Maps are described later in this document, but in a nutshell maps are used to map RUNs with different parameter names so that the RUNs can be compared and difference plotted.

The EXCLUDE: and FILTER: statements either cull out or filter for logfiles within the RUN directories. This can help speed up data loading and/or show a subset of all variables logged.

External Programs

The DP PROGRAM specification provides a means for transforming data. Users build a program that is dynamically linked into Koviz for manipulating data specified in the DP specification file.

External Program DP File Specification

The <program_name> argument is a full path to a program which accepts the INs and generates the OUTs. This program must adhere to strict interface requirements. This program will be dynamically linked into the data products, which implies it must be built under specific guidelines. Only ONE external program may be defined per DP file.

The IN parameters are specified as a space delimited list of simulation variable names and external program OUT token names. The OUT list is a user defined name list which provides a unique token name for each of the external program output arguments. These output tokens may be used throughout the product specification file wherever a simulation variable name is used.

Inputs will be cast to doubles going to the external program, and outputs must be doubles as well.

As an example, a user would like to plot the rss of a vector. The product specification file (DP_* file) might look like the following (NOTE: all strings surrounded in quotes):

External Program DP File Example

PROGRAM: “/home/kvetter/test/ext_programs/librss.so”

IN:

“Chaser.stVeh.stFFrDock.vecF_Frame[0]”
“Chaser.stVeh.stFFrDock.vecF_Frame[1]”
“Chaser.stVeh.stFFrDock.vecF_Frame[2]”

OUT:

“vecF_Frame_rss”

PLOTS:

PAGE 1: "External Program Example"

PLOT 1: “RSS”

X_VARIABLE: "sys.exec.out.time"
Y_VARIABLE: "vecF_Frame_rss"

External Program Source Code Example

The following is an example external program source file (rss.c) which takes three inputs and generates a single output. Bold font indicates code that is mandatory (i.e. “kovizProgram” is the function name that koviz expects for external programs).

```
#include <math.h>

void kovizProgram( double* in, int numIn, double* out, int numOut )
{
       out[0] = sqrt(in[0]*in[0] + in[1]*in[1] + in[2]*in[2]);
}
```

Building Example External Program On Linux

```
% cc -fPIC -c rss.c
% ld -shared -o librss.so rss.o -lm -lc
```

External Program Building

The external program created by the user must be built so that it can be dynamically linked into koviz.

In order to build, take the following steps according to platform.

Linux

Step 1. cc -fPIC -c <myprogram1>.c (compile all individual object this way)

Step 2. ld -shared -o <myprogram>.so <myfunction1>.o <myfunction2>.o... [<myLib>.a] -lc

MacOS X

Step 1. cc -c <myprogram1>.c (compile all individual object this way)

Step 2. cc -bundle -o <myprogram>.so <myfunction1>.o <myfunction2>.o ... [<myLib>.a] -lc

<myprogram>.so is the name that needs to be specified in the DP specification file. Set LD_LIBRARY_PATH so that koviz can find the shared object.

Try the following to see if the newly create shared object has unresolved dependencies:

UNIX% nm <myprogram>.so

External Program Summary

To use an external program:

Step 1. Build a DP spec file with the program name, inputs and outputs.

Step 2. Write an external program

Step 3. Build the external program.

Step 4. UNIX Prompt> koviz DP_external_program RUN_name

External Program Problems And Caveats

- Can’t load shared library!!! Ensure that $LD_LIBRARY_PATH is set so that koviz can find the *.so file. The external program (*.so program) may have unresolved dependencies. Try “nm” on the external program, and look for “U”s.
- Koviz will not scale or bias X values with external programs.
- External programs convert everything to doubles, and only accept and output doubles.
- External programs have no notion of unit conversion.

Plot Interaction

Select Curve	Mouse Left Click Curve
Drag Curve	Mouse Left Click Curve to select Then Keypress Crtl + Mouse Left Drag
Drag Curve Onto Another Curve	Mouse Left Click Curve to select Then Keypress Alt+Ctrl+Mouse Left Drag making init points of both curves close enough to “snap”
Zoom In	Mouse Middle Click and Drag Zoom Box
Zoom Out	Mouse Right Click or Keypress Escape
Pan	Mouse Left Drag In Middle Of Plot
Scale	Mouse Left Drag In Outter Extremity Of Plot
Toggle Single/Multiplot View	Mouse Left Click On Y-Axis Label or Plot Title
Toggle Y Axis Units	Mouse Wheel Over Y-Axis Unit In Curly Braces {unit}
Toggle Linear/Logscale	Mouse Wheel over x or y tic values
Suspend Live Coordinate Updates	Keypress Shift While Moving Mouse Off Plot
Toggle Presentation	Keypress Spacebar
Print Current Coordinate & Dx,Dy	Keypress Period
Measure Distance On Plot	Keypress Alt + Mouse Left Drag (updates koviz status bar)
Hop To Prev/Next Plot Coord	Keypress Left/Right Arrow (after curve selected)
Place Marker On Plot Coord	Keypress Comma
Change X Axis Variable	Options->Drag-n-Drop. Drag-n-drop variable onto x-axis label.
Frequency Domain Toggle	Keypress f
Butterworth Filter	Keypress b
Savitzky-Golay Filter	Keypress g
Flip Curve About Y-Axis	Keypress -
Derivative Of Curve	Keypress d
Integral Of Curve	Keypress i (use entry box for initial value, default to 0.0)

Maps

Maps are used to compare simulation RUNs whose variable names do not match. A map is either contained in a file or can be specified directly on the commandline. The map is a comma delimited set of key-values in the following form:

"key1=value1=value2...,key2=value1=value2,..."

Here is an example of comparing simulation position and velocity names differ.

UNIX Prompt> koviz -mapFile "spots2dyn.kvz" RUN_spots RUN_dyn

UNIX Prompt> cat spots2dyn.kvz

posx = spots.vehicle.pos[0] = veh[0].dynamics.posX,
posy = spots.vehicle.pos[1] = veh[0].dynamics.posY,
velx = spots.vehicle.vel[1] = veh[0].dynamics.velY,
vely = spots.vehicle.vel[1] = veh[0].dynamics.velY

Maps may also be used to scale and bias variables. If the logged data has no units, units may be applied in the map file as well. Here is an example of map file with scale, bias and unit specs:

```
ForceX = Fx               = inertialForceZ {N},
ForceY = Fy bias(-12.25)  = inertialForceX {N},
ForceZ = Fz scale(-1.0)   = inertialForceY {N}
```

Video

Introduction

Koviz uses mpv as the backend video player. Any video that mpv can play e.g. *.mp4, *.avi can be tied to data.

Building Koviz with MPV Support

To build koviz with video support, the mpv development libraries must be installed. On CentOS7, this requires the “Nux Dextop” RPM repository. Once Nux is enabled, something like the following will install the mpv dependencies and build koviz with video.

```
% sudo yum install  mpv-libs mpv-libs-devel mpv
% cd <koviz-repo>
% make distclean
% qmake-qt5
% make
```

Data Synchronization

Normally, video and data are alignable by a time offset. The time offset is given to koviz either on the commandline or in a file. Finding the time offset is a manual process unless the video and data were recorded together by a time synchronized system. For example, consider a GoPro camera recording a video of a person striking a loadcell with a hammer. The offset must be set to a value that aligns the hammer strike event in the video to the spike in the loadcell logged force data. Once a single event is synchronized, the video and data are completely synced – assuming the normal case where the video and data are real-time. Finding the exact video frame of the event that corresponds to the exact time in the data may not be possible. For instance, the hammer strike may not be seen in the video, only before and after the strike. In practice, this is okay, but is something to be aware of if the analysis requires that kind of fidelity.

Commandline -video and -videoOffset options

The simplest way to quickly tie a single video to data is to use the -video and -videoOffset options. For example:

```
% koviz RUN_loadcell -video hammerstrike.mp4 -videoOffset 12.345
```

Session Files

A session file may work better if there is a need to save the video and offset for future use. Example

session file (session_hammer):

```
Session – Hammer Video Sync
RUN: RUN_loadcell
PRODUCT: DP_forces
video: Videos/hammerstrike.mp4
videoOffset: 12.345
```

To run:

```
% koviz -session session_hammer
```

Video Directory Within RUN

Another way, especially useful when there are multiple RUNs tied to separate videos, is to create a "video" directory in the RUN dir. The video file, e.g. hammerstrike.mp4, is placed in the "video" directory. An optional file, explicitly called "video-offset.txt", contains the video offset for the video. For the hammertime example, the directory structure would look like:

```
RUN_loadcell
    loadcell_data.csv
    video
        hammerstrike.mp4
        video-offset.txt (containing 12.345)
```

If the RUN directory is setup this way the following will bring up the data and the associated video:

```
% koviz RUN_loadcell
```

Multiple RUNs with Multiple Videos

Using the video directory within a RUN directory technique, a set of RUNs can be tied to their associated videos. Once all RUNs have their associated videos in place, koviz can be launched with the entire set of RUNs. This makes it so one may hop from video to video by simply clicking on a curve of interest.

Blender

Introduction

Koviz is able to synchronize its plots to Blender animations via a koviz Blender plugin. The start/end times of the koviz simulation run data are mapped to the begin/end frames of the Blender animation. After selecting a koviz curve, mouse movement updates koviz live time interactively. The koviz live time is sent to Blender while the mouse moves. Blender interpolates the koviz time to the animation frame and updates the Blender scene. The sync works from Blender to koviz as well. With mouse/keyboard focus on the Blender window, running the Blender animation will send koviz time updates and update koviz plot live time.

Blender Addon Setup

First, add koviz to Blender's addons. Example:

```
% cd $HOME/.config/blender/2.91/scripts/addons/modules
% ln -s <koviz-home>/python/koviz.py koviz.py
```

Animate Blender Default Cube

As an example, make an animation of Blender's cube:

1. Bring up blender:
   % blender
2. Right click > Insert Keyframe > Location
3. Slide animation frame to 250
4. Move cube up 6: KeyPress gz6
5. Right click > Insert Keyframe > Location
6. Run animation by mouse clicking |< followed by >
7. Save as cube.blend
8. Close blender

Sync Koviz To Default Cube Animation

Blender argument order matters. The -P option must follow the model.

```
% blender cube.blend -P <koviz-home>/blender/koviz-hello-world.py
# Should see “Listen for koviz connection”
```

Run koviz on a RUN of your choosing. If the RUN has 250+ points, that’s best, for example the Trick ball sim has 300 points and works.

```
% koviz RUN_a
# Koviz should print “Connected to host=127.0.0.1 port=64053!
```

To see koviz time synced to the Blender animation:

1. In koviz, click some variable
2. Mouse select the curve
3. Move mouse to see Blender animation sync
   !! If it does not sync, click off of koviz curve, reselect curve and try moving mouse again.

To see Blender animation sync to koviz:

1. Select curve in koviz (if not already done)
2. Click Blender window so Blender has focus
3. Run Blender animation by mouse clicking the play button >

Sync Blender To Koviz With ES's Vismo

The Engineering Structures group have tied their vismo Blender tool to koviz. If you are lucky enough to have vismo, here is an example. The -k option tells vismo to sync using koviz:

```
% cd <rddr-nds>
% . .Trick_user_profile ; # To put vismo.py in $PATH
% cd <rddr-nds>/sims/rddr/nds/SIM_nds
% trick-CP
% ./S_main* SET_checkout/SET_m1u/RUN_checkout_MaxJackknife/input.py
% vismo.py -k SET_checkout/SET_m1u/RUN_checkout_MaxJackknife
    # Listen for koviz connection...
% koviz SET_checkout/SET_m1u/RUN_checkout_MaxJackknife
     # Should see "Connected to host=127.0.0.1 port=64053!"
```

To see the koviz to Blender sync:

1. In the koviz search box, type "Fqu.daActuator"
2. Scroll over all 6 vars to coplot 6 actuators on single page
3. Select a curve and move mouse to see data synced to docking animation

To see the Blender to koviz sync:

1. In the Blender window click the |< to put the animation at the beginning
2. Press > to play
3. The koviz live time should update along with the animation
4. Pause the Blender animation by pressing the | | pause button
5. Press the keyboard right and left arrow keys to step the animation and watch koviz update frame by frame