Koviz User's Guide

Version 1.0

March 2018

National Aeronautics and Space Administration
Johnson Space Center
Houston, Texas

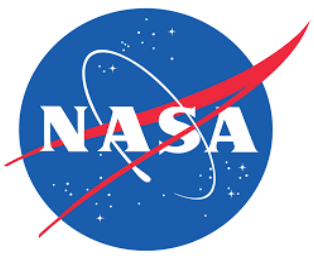

Table of Contents

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. 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 have up to 6 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>]

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”

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 -sessionFile command-line option may be used:

UNIX Prompt> koviz -sessionFile <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> ...]

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.

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.

External Programs

The DP PROGRAM specification provides a means for transforming data. Users build a program that is dynamically linked into Trick data products 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 parameter 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 parameter 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 post process a set of spacecraft attitude Euler angles, but only the 3x3 attitude matrix was recorded. An external program can be used to transform the 9 attitude matrix elements into 3 Euler angles. The product specification file (DP_* file) might look like the following (NOTICE: all strings surrounded in quotes):

External Program DP File Example

PROGRAM: “my_utilities/dp_euler_pyr_extract”

IN:

“orbiter.rot_dyn.out.R_inertial_to_body[0][0]”
“orbiter.rot_dyn.out.R_inertial_to_body[0][1]”
“orbiter.rot_dyn.out.R_inertial_to_body[0][2]”
“orbiter.rot_dyn.out.R_inertial_to_body[1][0]”
“orbiter.rot_dyn.out.R_inertial_to_body[1][1]”
“orbiter.rot_dyn.out.R_inertial_to_body[1][2]”
“orbiter.rot_dyn.out.R_inertial_to_body[2][0]”
“orbiter.rot_dyn.out.R_inertial_to_body[2][1]”
“orbiter.rot_dyn.out.R_inertial_to_body[2][2]”

OUT:

“pitch”
“yaw”
“roll”

PAGE 1: "Pitch-Yaw-Roll"

PLOT 1: “Attitude Matrix To PYR”

X_VARIABLE: "sys.exec.out.time"
Y_VARIABLE: "pitch"
Y_VARIABLE: "yaw"
Y_VARIABLE: "roll"

External Program Source Code Example

The following is an example external program source file which takes nine inputs and generates three outputs for the attitude matrix to euler angle example above. Bold font indicates code that is mandatory (i.e. “kovizProgram” is the function name that koviz expects for external programs).

```
int kovizProgram( double* in, int numIn, double* out, int numOut )
{

	// Declarations
	void euler231( double ang[3] , double mat[3][3] ) ;
	double mat[3][3] ;
	double pyr[3] ;

	// Load attitude matrix
	mat[0][0] = in[0];
	mat[0][1] = in[1];
	mat[0][1] = in[2];
	mat[0][1] = in[3];
	mat[0][1] = in[4];
	mat[0][1] = in[5];
	mat[0][1] = in[6];
	mat[0][1] = in[7];
	mat[0][1] = in[8];

	// Extract PYR angles using math library
	euler231( pyr, mat );

	// Load program outputs
	out[0] = pyr[0];
	out[1] = pyr[1];
	out[2] = pyr[2];

	return(0);

}
```

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 -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
Zoom In	Mouse Middle Click and Drag Zoom Box
Zoom Out	Mouse Right Click
Pan	Mouse Left Drag In Middle Of Plot
Scale	Mouse Left Drag In Outter Extremity Of Plot
Full/Page View	Mouse Left Click Plot White Space
Toggle Y Axis Units	Mouse Wheel Over Y-Axis Unit In Curly Braces {unit}
Suspend Live Coordinate Updates	Keypress Shift While Moving Mouse Off Plot
Toggle Presentation	Keypress Spacebar
Print Current Coordinate	Keypress Period

Maps

Maps are used for two special cases. Maps make comparing two simulation RUNs whose parameter names do not match possible. Maps also allow one to specify RUN:parameter pairs to coplot across multiple RUNs. 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,...”

What follows are three examples. The first compares simulation x-position when x-position names are not the same. The second coplots variables xyz from RUNs abc respectively. The third example shows how to map one RUN to another using a file.

UNIX Prompt> koviz -map “x=spots.vehicle.pos.x=dynamics.veh[0].posX” RUN_spots RUN_dyn

UNIX Prompt> koviz -map “v=RUN_a:x=RUN_b:y=RUN_c:z” RUN_a RUN_b RUN_c

UNIX Prompt> koviz -mapFile “spots2dyn.kvz”

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
```