Morpheus Data Parsing and Plotting A New Approach to Large Data Set Analysis with Open Source Tools

J.R. Leeman

August 10, 2011

1 Abstract

Project Morpheus is a technology demonstration project focusing on an innovative propulsion system, autonomous guidance, navigation, and control (GNC), and streamlined development techniques enabling space vehicles development on short timelines without a loss of reliability. Currently the project has two craft, the Pixel lander (Risk Reduction -1) and the Morpheus 1.0 lander (Risk Reduction 2). Both machines have flown under their own power and RR-2 is being prepared for its first free flight test which involves liftoff, translation, and safe landing. The guidance, navigation, and control (GNC) for the lander was developed on a rapid timeline and produced in a very modular fashion so it is easily adaptable to changes in the vehicle structure or entirely different vehicles.

Large data sets produced by the vehicle systems are difficult to mine for useful information. Most software packages are not capable of handing datasets with many millions of points. Open source software was produced

using python that allowed graphing and data parsing from the Morpheus GNC files. The software also produces reports of major flight events from the Autonomous Flight Manager (AFM) output. Using these tools engineers are able to spend time solving vehicle issues, not parsing data to produce the graphs and reports used to diagnose problems.

2 Introduction

The Morpheus lander produces volumes of output data, which is difficult to analyze and graph in an efficient way. A new software package was needed to standardize the plots used by project engineers, produce useful reports that can give an overview of the flight at a glance, reduce the time engineers spent plotting data, and prevent duplication of work by multiple engineers.

Data comes from two sources: flight tests/static tests with the vehicle and numerical simulation of flights. Both produce data in a very similar format with the exception that truth data is known in the simulations. At a minimum around 1000 variables are brought in through the CSV files (Table 1). Currently the data is brought into the trick data plotter (trick dp), MATLAB, or even into excel. Trick dp is good for plotting simulation data, but not equipped to plot flight data, resulting in slightly different looking plots to be produced for simulation data. MATLAB is an effective tool for plotting the data, but it carries an expensive license fee to just be used for plotting and there is no repository of plotting scripts for the project. Excel does not handle such large data sets well and is prone to crashing easily.

There was also no report style output from the flight data to easily allow a user to glance at the report and learn times of important changes in the vehicle, monitor vehicle strike counters, and monitor basic true/false flags in the data. Such a report needs to be produced in a format easily readable on multiple devices (computers, smart phones, tablets, etc) to allow field data analysis.

By producing such software there will be a reduction in duplication of work allowing increased man-hours to be dedicated to data analysis. Using the same plotting package for all data also ensures that all plots look similar in format and style reducing the chance of data display influencing or biasing decisions made on that data.

3 Python

Python is an multi-platform, open source interpreted programming language. The language is relatively new, appearing the 1991 and designed by Guido van Rossum. The data analysis toolkit was developed in python due to its open source nature, readability, and rapid development capability. Though not used for this application, the scripts developed could be easily linked to Fortan or C programs to increase the speed of computations preformed before data analysis and graphing takes place.

Open source software is gaining popularity in scientific circles. By making the source code of an analysis code available it allows others error check the results and build off of/improve the software. Open source software also makes the financial impact of data analysis disappear as expensive software licenses are not required.

The high-level nature of the Python language also makes the source code very readable. Simple commands can be easily followed and replace many lines of C code (at the expense of computational efficiency). While doing post flight data processing computation speed is not of upmost importance, but rapid development is. The project benefits from using graphs and reports as tools to diagnose performance of the vehicle, but time to develop those tools should be as small as possible. Python allows such rapid development and is very easy to learn/alter.

4 Software Design

Modular software design has always been encouraged in scientific communities to make source code reusable and reduce development time. The CSV Toolkit was designed to be reusable with different vehicle configurations with little to no modification. Plotters, data parsers, configuration parsers, report generators, and coordinate conversions make up the toolkit (Table 2).

When wanting to produce a standard suit of plots data is first passed to a parser/formatter. A user defined configuration file is also passed to the parser/formatter. The configuration file contains requests to store and plot certain variables. Requested data is polled from the input files, combined, and output as a single CSV file. The combined CSV file is then passed to the reader/plotter. This module also uses the configuration file and produces the requested plots. The output is stored as PNG, PDF, PS, or JPG (Fig.1).

The AFM report generator works in a similar way. The flight data file is passed to the same parser/formatter as above. The output of the parser is then given to a change detector. The change detector monitors certain variables and marks important changes. Variables such as change in flight mode, vehicle strike counters, and true/false flags are considered. Important changes are passed to an HTML writer, which produces as HTML report that can be opened with any web browser on a desktop or mobile device (Fig.2).

A graph report tool is also included in the package. This tool simply looks for graphs produced by the plotter and writes an HTML page to

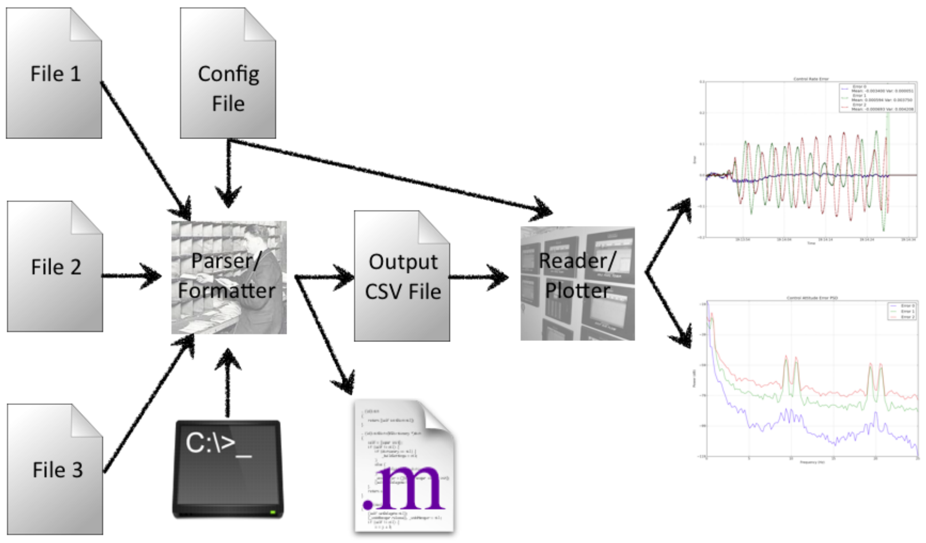


Figure 1: Flow of data in the CSV plotter portion of the toolkit. Note that the parser/formatter may also be controlled from the command line as well as the configuration file. There is also an option to produce a .m data file so existing MATLAB scripts can be used.

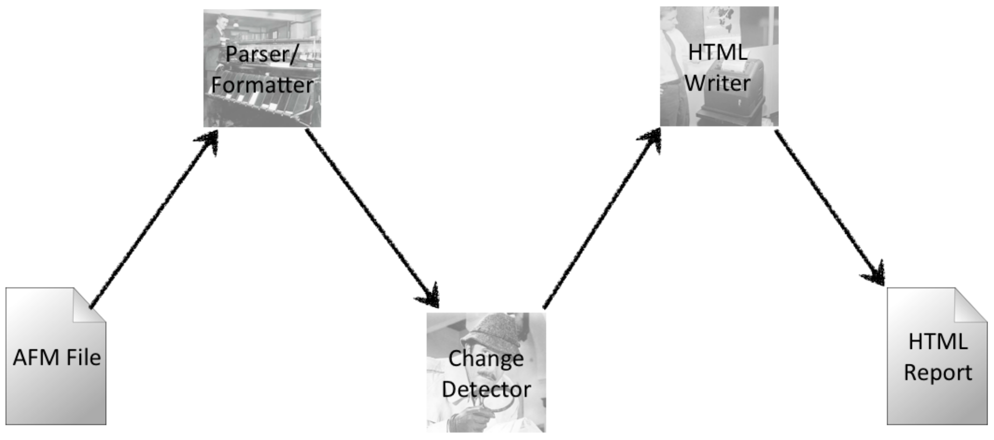


Figure 2: Flow of data in the AFM report portion of the toolkit.

display them on any web browser. With the addition of FTP capability the graphs and reports could be automatically be posted to a server immediately after a flight.

Coordinate conversion is also an import part of data analysis. While the CSV Tools package only has a basic coordinate converter it was designed to be expanded easily. The data parser module is reused to pull in data, preform numerical calculations, and write out a new file. Currently quaternion to Euler angle (eq.1), and radian to degree conversions (eq.2) are supported.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix} \tag{1}$$

$$\text{degrees} = \text{radians} \cdot \frac{180}{\pi} \tag{2}$$

5 Software Use

While describing the use of the software is beyond the scope of this technical paper, it has been summarized in the CSV Tools Repository Guide included with the software. The software is run from a user specified configuration file. This is a plain text file written in easy to read non-programming language (Table 3). The data to be used is specified and typed. The graphs desired are also specified.

There are three types of graphing options in the package. Plotting parameters that can be set in the configuration file are summarized in table 4. A standard x-y plot of two variables is easy to produce and is called a *std* plot. Plots with time stamps on the x-axis are also easy to produce with the *timex* plot type (Fig.3). Finally, power spectral density plots are

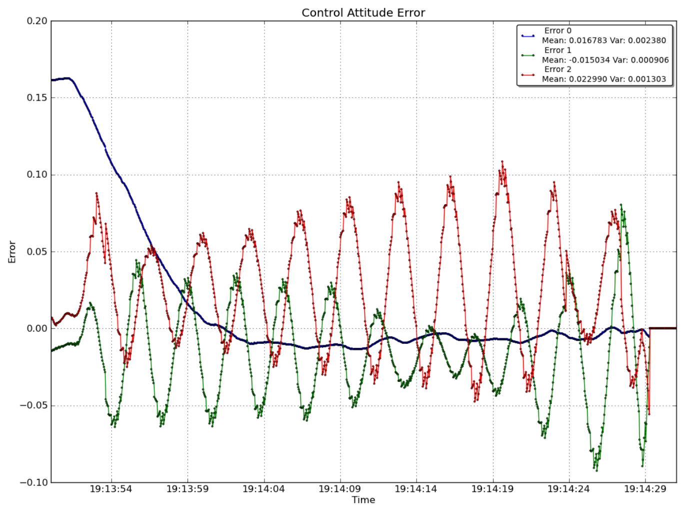


Figure 3: An example timex type plot. The standard type plot is identical, but with a variable other than time of the x-axis.

computed and produced from single channel data by invoking the *psd* plot type (Fig.4). The power sprectral density tool is very versatile with possible parameters designated in table 5.

Multiple variables from different files may be displayed on the same graph, as well as figures with multiple subplots. Optional legends on the figures not only show how each variable is plotted, but also show basic statistics on the data allowing quick evaluation of that particular parameter. LaTeX notation may be used when specifying all parameters for plot display.

The AFM and graph reports are automatically generated by the associated code without need for a configuration file. An example AFM report is shown in figure 5.

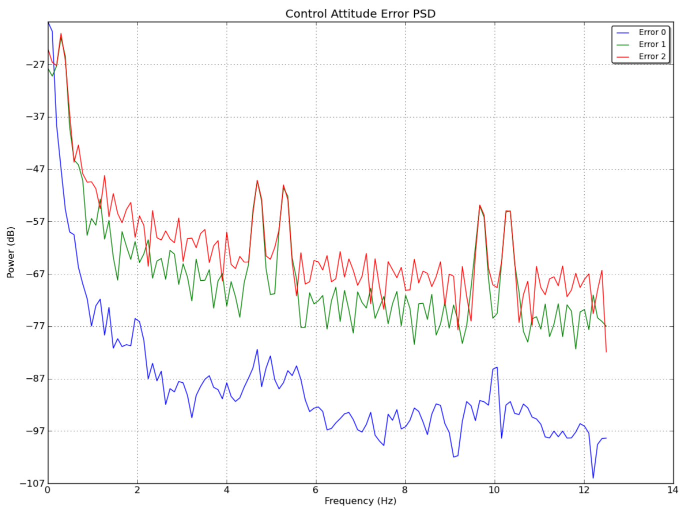
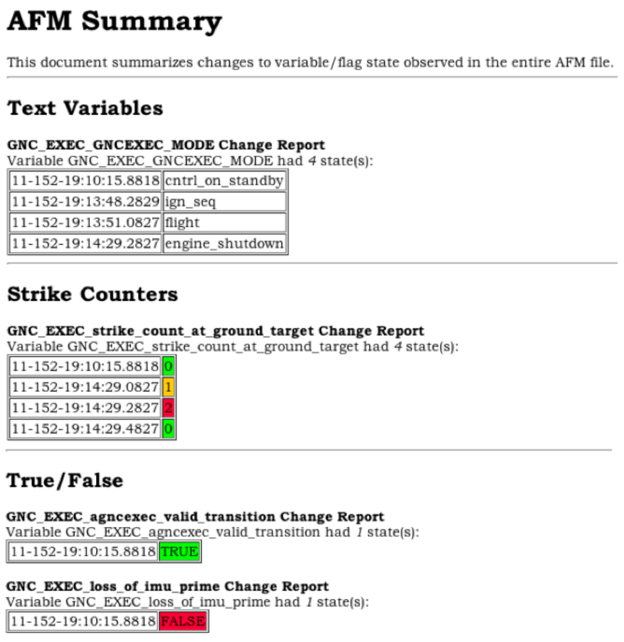


Figure 4: An example plot of power spectral density.



AFM Summary

This document summarizes changes to variable/flag state observed in the entire AFM file.

Text Variables

GNC_EXEC_GNCEXEC_MODE Change Report
Variable GNC_EXEC_GNCEXEC_MODE had *4* state(s):

11-152-19:10:15.8818	cntrl_on_standby
11-152-19:13:48.2829	ign_seq
11-152-19:13:51.0827	flight
11-152-19:14:29.2827	engine_shutdown

Strike Counters

GNC_EXEC_strike_count_at_ground_target Change Report
Variable GNC_EXEC_strike_count_at_ground_target had *4* state(s):

11-152-19:10:15.8818	0
11-152-19:14:29.0827	1
11-152-19:14:29.2827	2
11-152-19:14:29.4827	0

True/False

GNC_EXEC_agncexec_valid_transition Change Report
Variable GNC_EXEC_agncexec_valid_transition had *1* state(s):

11-152-19:10:15.8818	TRUE

GNC_EXEC_loss_of_imu_prime Change Report
Variable GNC_EXEC_loss_of_imu_prime had *1* state(s):

11-152-19:10:15.8818	FALSE

Figure 5: An example portion of an AFM report generated from Morpheus flight data.

6 Conclusions

A versatile plotting package was produced that met all project goals. The Morpheus team is less reliant on expensive commercially licensed software and can produce automated plot sets to rapidly share. The open source nature of the software and its location in the Morpheus tools subversion repository allows all team members to access and run the software with standard or custom configuration files.

By reducing duplication of efforts, team members can spend more time diagnosing problems with vehicle flight software and less time trying to produce tools to aid their analysis. By introducing a simple configuration file format powerful analysis capability is available to non-programmers on the team. While the plotting package is not meant to replace MATLAB, it is meant to reduce the reliance on it. The adage of the right tool for the right job applies more than ever to data analysis.

File Name	Description	Number of Variables
AFM	Autonomous Flight Manager	45
ALTIMETER	Laser Altimeter	11
CNTRL	Control Subsystem Outputs	24
GPS	Global Positioning Data	12
GUID	Guidance Commands	53
IMUPRE	IMU Preprocessing Data	22
LN200	Output from the Litton IMU	86
SIGI	Output from the SIGI IMU	315
NAVKF	Kalman Filtered Navigation Output	93
PROP	Propulsion Subsystem Output	188
NAVFP	Fast Propagation Navigation Output	174
UPP	Universal Pointing Package (Nav)	56
	Total	1079

Table 1: The GNC files from Morpheus of interest and their associated number of variables.

afm_report.py	Produce an HTML file highlighting important changes in variable and strike counter states.
ConfigParse.py	Read file and plotting parameters from the configuration text file and pass them as python dictionaries and lists to allow easy plotting parameter setup.
Coordinate.py	Convert data between vehicle coordinate systems.
CsvFormat.py	Read a csv file and allow the user to write a new csv file containing only a subset of the existing data. This script may be called on its own to run in interactive mode or its functions may be accessed by other scripts to automate the plotting process. Also can accomidate the merging of data from multiple CSV files.
DataPlotter.py	Given the data and desired headers, plot layouts, etc produces the plots and saves them in the desired format.
graph_report.py	Produce an HTML file to display all graphs produced and directed to the output directory.
RunPlots.py	A simple control script that accesses all the other script's utility. Given a text configuration file the existing data is formatted, plotted, and saved according to the users instructions. This is the tool that automates the standard suite of plots.

Table 2: The python scripts that compose the CSV tools repository and a brief description of their purpose.

file_name	File name of the csv file to read data from.
time_start	The start time of data to be read/plotted. May be entered as row number (starting at 0) or as a time string exactly matching one of those in the file.
time_end	The end time of data to be read/plotted (not enabled in this version of CSV Tools). May be entered as row number (starting at 0) or as a time string exactly matching one of those in the file.
num_header	Number of header lines in the original csv file.
prefix	Add a prefix to the variable names. Used when the user desires to read in the same file, but with different parameters. Defaults to none.
decimate	Decimate the data. Given as in integer value (i.e. decimate every 10th data point). Defaults to 1 for recording all data points.
time_var	The name of the variable to use for time slicing with the time start/end cuts. Defaults to the first column of the file.
var	Name of a variable to include in the formatted file/plot as it appears in the *ORIGINAL* file header.

Table 3: Input file parameters recognized in the text configuration file for RunPlots.py.

PLOT	Designates a new plot configuration.
fig_name	Name with which to save the figure. Also determines type (png, jpg, pdf, etc).
position	Position of the graph on the figure in a rows, columns format. For one graph on a figure 1,1 is still necessary.
x_var	The name as it appears in the file header of the independent variable to plot.
y_var	The name as it appears in the file header of the dependent variable to plot.
x_label	Set label to appear under x axis.
y_label	Set label to appear under y axis.
rad_to_deg	Convert the y variable from radians to degrees for plotting.
title	Set the graph title (individual title if subplots are used).
legend	Set as True or False to provide a legend entry to the specific variable. Defaults to True.
END	Designates the end of the plot configuration and readies the code for a new plot.

Table 4: Key parameters used to configure plots and plot layout as recognized in the text configuration file for RunPlots.py.

Num_FFT	Defines the number of data points (integer) to be used in each block for FFT. Powers of 2 are the most efficient computationally, but any even number will run. *Default: 256*
Fs	Sampling frequency *Default: 50*
detrend	Calls a built in matplotlib or custom user defined function to detrend the data. Matplotlib options are *detrend_none* (default), *detrend_mean()*, and *detrend_linear()*.
window	Uses existing windowing options or takes a window vector of length NFFT. Windows avaliable include *window_hanning()* (default), *window_none()*, *numpy.blackman()*, *numpy.hamming()*, *numpy.bartlett()*, and those avaliable in the scipy.signal toolkit. User may have to change call slightly based on how packages are imported.
overlap	Number of points (interger) overlap between FFT blocks. *Default: 0*
padding	Number of points to pad data segment for FFT (integer). While not improving resolution includes more points in the actual PSD plot allowing smaller peaks to be evident. *Default: set to Num_FFT*
sides	Which side of the PSD to return: *'default'* returns one side for real data and both for complex data, *'onesided'* and *'twosided'* force one and two sided returns.
freq_scale	Sets if resulting densities are scaled by the scaling frequency resulting in a unit of Hz$^{-}1$. *Default: True*
Fc	Center frequency of the signal (integer) offsetting the x extents of the plot. *Default: 0*

Table 5: Parameters used to configure power spectral density plots as recognized in the text configuration file for RunPlots.py.