A Hybrid Parachute Simulation Environment for the Orion Parachute Development Project

James W. Moore[1]
Jacobs ESCG, Houston, TX, 77598

A parachute simulation environment (PSE) has been developed that aims to take advantage of legacy parachute simulation codes and modern object-oriented programming techniques. This hybrid simulation environment provides the parachute analyst with a natural and intuitive way to construct simulation tasks while preserving the pedigree and authority of established parachute simulations. NASA currently employs four simulation tools for developing and analyzing air-drop tests performed by the CEV Parachute Assembly System (CPAS) Project. These tools were developed at different times, in different languages, and with different capabilities in mind. As a result, each tool has a distinct interface and set of inputs and outputs. However, regardless of the simulation code that is most appropriate for the type of test, engineers typically perform similar tasks for each drop test such as prediction of loads, assessment of altitude, and sequencing of disreefs or cut-aways. An object-oriented approach to simulation configuration allows the analyst to choose models of real physical test articles (parachutes, vehicles, etc.) and sequence them to achieve the desired test conditions. Once configured, these objects are translated into traditional input lists and processed by the legacy simulation codes. This approach minimizes the number of sim inputs that the engineer must track while configuring an input file. An object oriented approach to simulation output allows a common set of post-processing functions to perform routine tasks such as plotting and timeline generation with minimal sensitivity to the simulation that generated the data. Flight test data may also be translated into the common output class to simplify test reconstruction and analysis.

Nomenclature

C_D	=	drag coefficient
C_DS	=	drag area
qbar	=	dynamic pressure

I. Introduction

THE test program to design and validate the parachute recovery system for the NASA Crew Exploration Vehicle (CEV) includes ambitious objectives and complex test techniques.[5,6] Engineers perform trajectory simulations prior to the tests to examine the effectiveness of these techniques and ensure the safety of the test. While the details of each test are unique, engineers typically perform a similar set of analyses for every test. Over the years, multiple simulations have been developed to meet specific needs. Each simulation has its own input methods and output files. The result is that engineers need to be skilled in using different tools to perform the same task. Time and effort that could be spent analyzing data is diverted to translating engineering concepts into tool-specific simulation inputs and decoding tool-specific output into the parameters of interest. A key analysis technique is to compare the results of simulations to actual flight test data. Flight data brings additional format, synchronization, and noise reduction issues that further complicate the analyst's task.

This paper will briefly describe a set of MATLAB object classes that have been developed to simplify the engineer's interaction with these multiple simulation tools and test data. These tools construct a hybrid simulation[7] environment that consists of simulation tools based in several programming languages. The simulations are brought together into a single Parachute Simulation Environment (PSE) that maintains the versatility and reliability of the multiple legacy simulations while providing the trajectory analyst with a common way to interact with simulation input and output. Some of the parachute simulations used by the CPAS project will be briefly introduced to explain the tasks performed by the PSE. The PSE includes tools for configuring simulations and analyzing output. Both of these functions will be described in detail. Finally, limitations and future improvements will be discussed.

[1] Analysis Engineer, Aerothermal and Flight Mechanics, 455 E. Medical Center Blvd., Webster, TX.

II. Legacy Simulations

The CPAS project employs four parachute trajectory simulations, three of which will be adapted to the simulation environment. The PSE effectively translates the input and output used by several simulations into a common format. To understand how the PSE works it is helpful to review the simulations that it manages.

DSS[4] (Decelerator System Simulation) is a legacy 6 Degree-of-Freedom (DOF) parachute trajectory simulation based on the UD233A simulation used by the Space Shuttle Solid Rocket Booster parachute project. DSS is written in Fortran and user input is performed via pre-configured text files. The DSS executable produces binary and text format output files. DSS is the highest fidelity NASA-maintained simulation used by the CPAS project.

DSSA[2,3] (Decelerator System Simulation Application) is a 6-DOF parachute trajectory simulation based on DSS that includes the capability to model the extraction of a test platform from an aircraft. Like DSS, this simulation is written in Fortran. DSSA employs a spreadsheet front end to configure the input file for a compiled executable. After execution, DSSA displays trajectory plots based on text and binary data files generated by the executable.

DTVSim (Drop Test Vehicle Simulation) is a 2-DOF parachute trajectory simulation intended for conceptual planning of test trajectories. This simulation includes parachute inflation models and is appropriate for early assessment of parachute loads as well as available altitude studies, programmer sizing, and reefing ratio selection. DTVSim is MATLAB-based and inputs are configurable through a MATLAB/Java-based user interface. After execution, output is plotted in the same user-interface.

Sasquatch[1] is a footprint prediction tool that uses wind profiles and trajectory information to plot dispersed touchdown circles against drop zone boundaries. While not a trajectory integrator, Sasquatch interacts with data generated by the other simulations. The PSE can facilitate this interaction.

In addition to the simulations mentioned above, the parachute manufacturer Airborne Systems uses a proprietary simulation named DCLDYN[8] that provides an independent check on simulation results. DCLDYN is not part of the PSE but is mentioned here to provide a complete list of trajectory simulations used on the CPAS project.

As would be expected of simulations developed at different times and for different purposes, each simulation uses a different input method and output data is presented differently. DSSA and DTVSim provide their own custom methods for reviewing output. Direct access to the output data is available for all three simulations but the file formats are different. Also, the input and output variable names corresponding to a particular physical quantity are typically different across the three simulations. DSS and DSSA employ Fortran common blocks. Users familiar with this practice will recall that this allows the same memory location to represent different variables depending on the particular subroutine being executed. Moreover, within the DSS and DSSA input file, variable names and common block locations are re-used for each parachute in a test sequence. This allows the re-use of code but presents opportunities for error. Engineers familiar with the use of legacy simulations will, no doubt, be acquainted with similar complications.

Re-coding these tools in a more modern language would be time-consuming and would require extensive validation. The existing simulations have proven to be accurate and have a track record of success. The PSE aims to retain the reliability of the legacy simulations while providing a simpler and less error-prone interface. Pairing an interpreted language with an existing compiled simulation has proven to be useful in this situation.[7]

III. Intuitive Simulation Configuration

The most complex of the simulations described above requires hundreds of user-inputs to execute. Properly selecting such a large number of inputs is problematic. A detailed understanding of the simulation code is required to fully understand the importance of all the user inputs. It is increasingly difficult to maintain this level of competency as simulation codes age, documentation grows out-of-date, and the importance of some of the simulation functions decline. The traditional approach to this problem is to start with something that worked before and update the user inputs that are known to be important for the analysis in question. Since the CPAS project is a parachute development program, many of the parachute modeling parameters and test vehicle sensitive inputs change from test to test. This limits the applicability of previous simulation input files to upcoming tests. With this approach to simulation re-configuration, CPAS trajectory designers have found that they need to track roughly one hundred inputs in order to properly configure a simulation for a single test. The consequence of missing one of these inputs could be re-performing an analysis that takes days to complete.

An alternative approach to configuring simulation inputs is to assemble the inputs in a more intuitive way using off-the-shelf components. This approach mirrors the way tests are assembled in the physical world. Engineers have a test objective that they desire to meet for a specific test. They survey the available test vehicles, parachutes, and other hardware that can be used to achieve this test objective. Once these components are selected, engineers assemble the pieces into a test article. By selecting (or in some cases building) the physical components, test engineers implicitly define the associated mass properties, dimensions, aerodynamic properties, parachute inflation parameters, and characteristics that will affect the test trajectory. The PSE attempts to mimic this test design process within the simulation configuration process using object-oriented programming techniques.

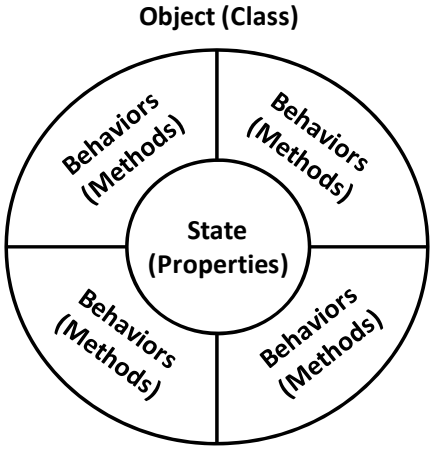

Figure 1. Object schematic. *MATLAB implements object-oriented concepts by defining classes. The state of the object is stored in its properties. The object behavior is described by special functions called methods. Interaction with the data is performed via the methods.*

The PSE is built on a set of MATLAB objects that model the properties of test vehicles, parachutes, sensors, and simulations. As with other object-oriented languages, MATLAB can represent physical objects with virtual objects. Figure 1 shows a typical schematic of a virtual object. In the MATLAB implementation of object-oriented programming, objects are represented by 'classes', the state of an object is stored in its 'properties', and the behavior of an object is determined by its 'methods'. The data associated with an object is stored in its properties and access to the properties should be performed through the methods. Moving from the abstract object to specific types of objects, the PSE defines classes that are useful in generating simulation inputs. These include parachute, vehicle, sensor, and simulation objects. Figure 2 provides a schematic of these objects, including sample properties and methods. The generic vehicle object has properties such as mass, inertia tensor, aero coefficients, and attach points. Parachute objects have properties such as mass, diameter, drag coefficient, riser and suspension line lengths, reefing ratios, and inflation parameters. Generic sensor objects are models of the sensor output rather than the physical sensors themselves and have many available properties such as position, velocity, acceleration, load, temperature, wind velocity, etc. Sensor objects are more important in post-flight analysis and will be discussed later. Simulation objects have properties such as atmosphere models, and other information that allows MATLAB to build an input set. For example, the DSS object includes the Fortran common block locations associated with each input variable.

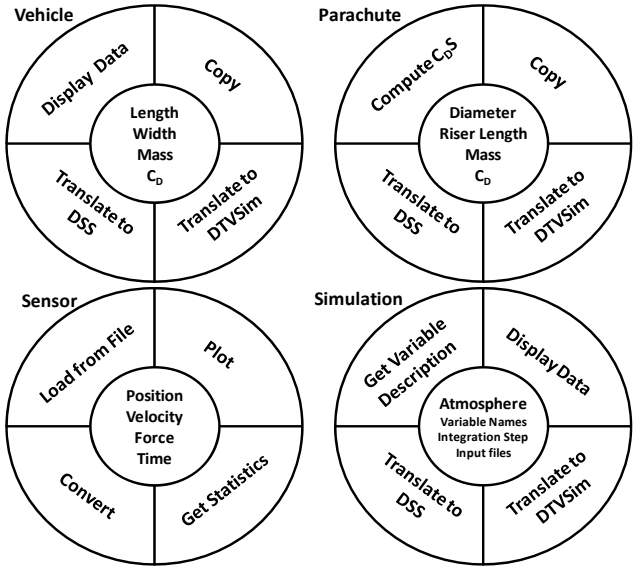

Figure 2. Parachute Simulation Objects. *The physical objects that make up a parachute test are implemented in an object-oriented simulation configuration environment. The particularities of the individual simulations are also coded as objects.*

The first step in reducing the number of inputs tracked by the analyst is to build some common instances of these generic objects that assign particular values to the object properties. This represents another step from the abstract toward the concrete. For example, in Fig. 3, an instance of the parachute object might be created to model the CPAS Main parachute. The inflation parameters might be set to the latest model values, the physical dimensions might be set to the values in the design documentation. The CPAS Main instance of the parachute object is then saved for

later use. Now instead of verifying that twenty or so simulation inputs associated with the Main parachute are correctly typed into an input file, the engineer just needs to identify the baseline Main parachute as the test parachute. A similar process is repeated for the common test vehicles. Instances of common sensor objects are a convenient place to store information about sensor precision and typical output units.

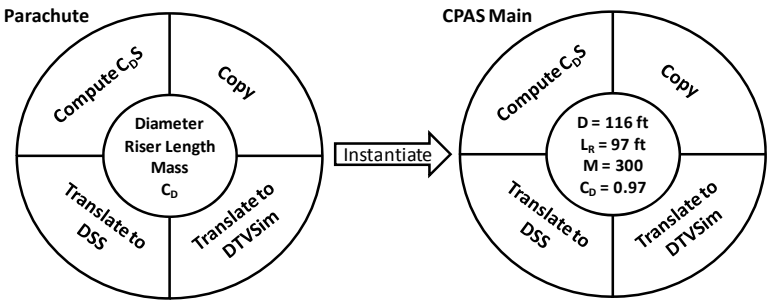

Figure 3. Instantiation schematic. *Specific instances of the generic objects contain the data associated with the physical object to be modeled. Here, an instance of the parachute object corresponding to the CPAS Main parachute is created.*

Each object has methods that operate on its properties. For sensor objects, these methods might detect and remove bias from the acceleration property or convert measured inertial vehicle velocity and wind information into airspeed. The primary methods for the vehicle and parachute objects are those that translate the internally stored properties into the input formats of the three simulations.

With these objects and their methods, a set of simulation inputs can be built "from scratch" rather than modifying an existing input set. This reduces the likelihood of mistakenly carrying forward an inappropriate input from and older analysis. The process of building the input set is similar to building the physical test article. Figure 4 shows an example. The test vehicle and parachutes are selected from the common off-the-shelf objects. At this point the default properties can be overridden with test specific conditions, if required. Each object simulation translation method is then called to convert the MATLAB object into a list of text inputs in the case of DSS and DSSA, or a

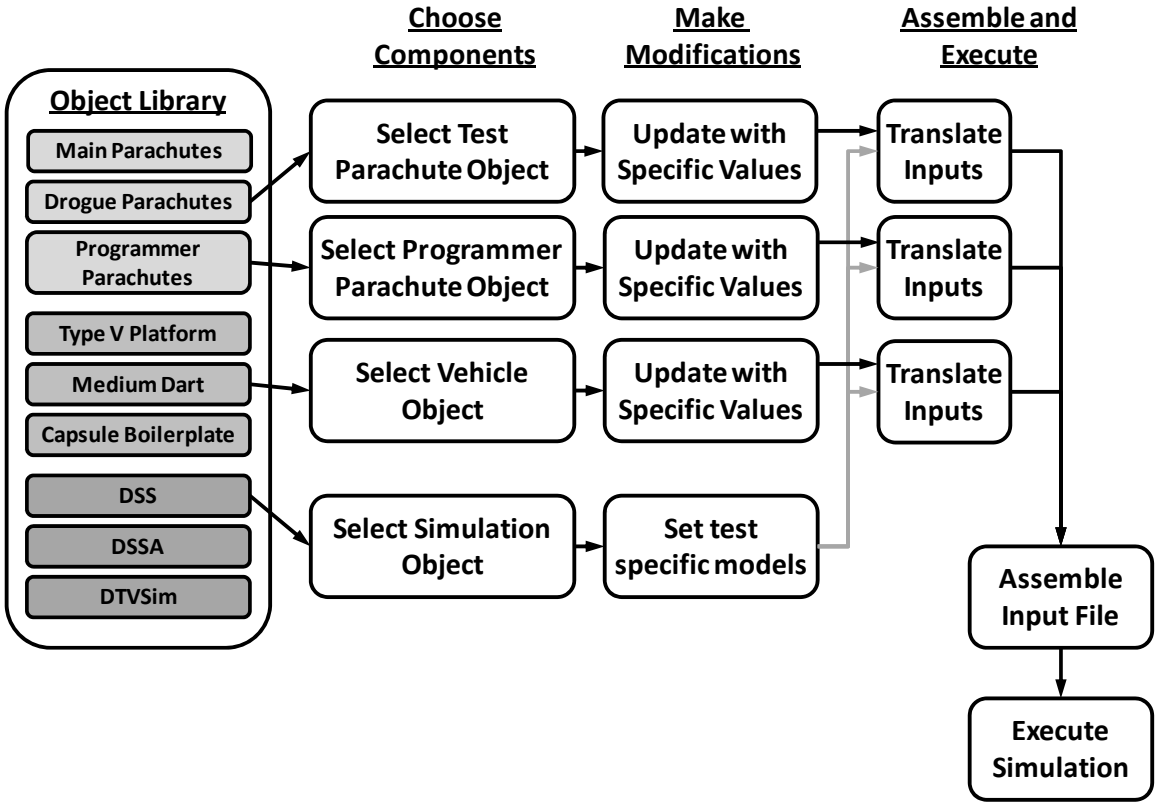

Figure 4. Simulation build sequence. *Test articles are selected from off-the-shelf objects. Any required modifications to the selected objects are made, then the objects translate themselves into input values for the legacy simulation. The inputs are then compiled into a single file for execution.*

simulation-specific data structure in the case of DTVSim. These sets of inputs are then assembled in the appropriate sequence and saved as an input file for the legacy simulations. The simulation may then be executed in the usual manner.

An additional benefit of this approach is that it allows a single test configuration to be translated into multiple simulations with minimal changes to the configuration script. This feature could be useful for simulation comparisons or for improving fidelity from 2-DOF to 6-DOF as the test planning process matures.

The simulation configuration function of the PSE is still under development and has seen limited use. Engineers should not undervalue the importance of deep understanding of the models and algorithms employed by a trajectory simulation. However, the intuitive configuration process used by the PSE will hopefully lower the learning curve for new analysts and allow meaningful analyses to be performed quickly and with minimal error and re-work.

IV. Common Output Interface and Analysis Methods

The common output functionality of the PSE was the first component to be developed and has more extensive capabilities than the simulation configuration functionality. The motivation for developing this interface is the observation that many parachute test analyses are interested in several common parameters. For example, many pre-flight test predictions focus on time histories of parachute riser loads, dynamic pressure, and altitude. Post-flight analyses are commonly interested in the same parameters as well as the calculation of drag coefficient and inflation characteristics. While the output of the typical CPAS analysis will be similar, namely, plots and peak values of these parameters, the input data to the analysis comes in at least four different formats. There are three different output formats corresponding to the three simulations, with many output parameters having a different name in each simulation. In addition, test data may have a different format, base unit, and variable name depending on the sensor collecting the data. For example, CPAS uses multiple sensors that collect acceleration data. Each sensor has its own format and may have an unsynchronized time channel.

This situation has resulted in a collection of custom analysis and plotting scripts that were tailored for specific sensors or simulations. In addition, each analyst might have a personal library of analysis tools that is not easily transferrable to other engineers. The goal of the common output interface was to make analysis scripts applicable to multiple simulation tools, transferrable to other analysts, and re-usable for multiple tests.

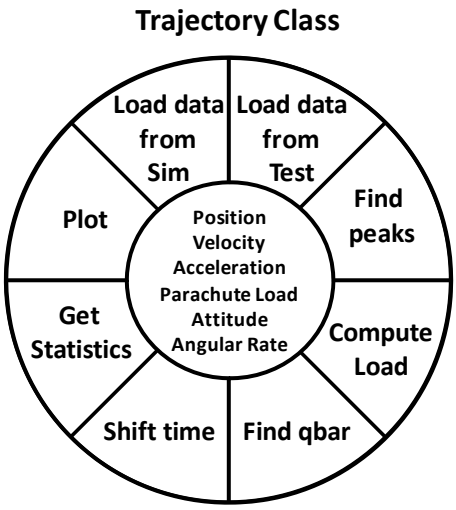

Figure 5. The Trajectory class. *The trajectory class provides a common structure for simulation output and flight test data. Methods are included to perform common analysis tasks and standardize output plots.*

The PSE common output interface employs another set of MATLAB object classes to manage data and perform common calculations. Figure 5 depicts the generic object called a "trajectory", including a few representative properties and methods. There are over one hundred trajectory properties and roughly seventy methods. The trajectory class has properties associated with all the common simulation output variables and sensor readings. These properties include trajectory data such as position and velocity as well as mass properties, parachute parameters, and environmental data.

Trajectory objects include methods that instantiate the object properties with arrays of trajectory data extracted from the simulations or from the recorded test data. For each of the simulations there is a method to load data into the property arrays. In a sense, the trajectory object "knows" how to fill itself with data from each of the simulations. There are also methods to populate the property arrays with the common sensor and balloon data formats as well as generic data sources like text files and spreadsheets. The process of converting data from the various sources into a common structure is an important step in creating re-usable analysis scripts.

Another important component of the PSE is a streamlined plotting capability. Several methods are also included that extract characteristics of the data such finding local peaks and mean values. The data label and unit information is contained within the object. The trajectory object has a plotting method that builds on the basic MATLAB plot

command by automatically adding titles, labels, legends, error bars, text annotations, and timeline events. This allows the analyst to concentrate on the data while ensuring that plots are free of typographical errors and are in a consistent format regardless of the team member who generated the plot. Any trajectory parameter may be plotted against any other parameter. Figure 6 shows a sample plot of parachute load vs. time generated with the following MATLAB commands:

```
DSS = trajectory('name', 'DSS');            % Construct a Trajectory object named 'DSS'
DSS.loadDSS('Sample.mat');                  % Load simulation data
DSS.get_peak_loads('time',{[20 30],[40 50],[60 70]});  % Find peaks in the provided timeframes
DSS.trajplot('chute_load','xlim',[0,100],'annotate','chute_load')  % Plot and annotate
```

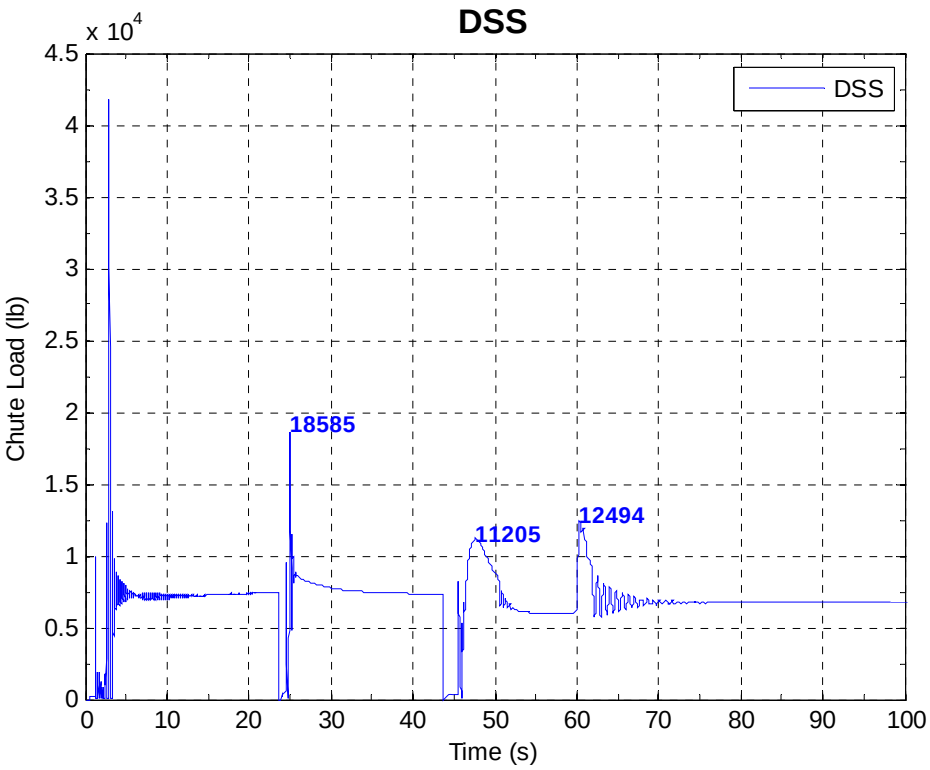

Figure 6. Sample trajectory plot. *The trajectory class allows analysts to quickly generate standardized plots using the same basic command for any simulation or set of test data.*

The trajectory class provides a simple means to add similar trajectory data from other sources. In this way, co-plots are generated with the same command that generates regular plots. If a trajectory object contains multiple trajectories, the plot command will automatically co-plot the data. For example, an engineer might load data from a pre-flight prediction simulation into a trajectory object and then with another command, load flight data from a sensor output file, and then call the plot command to compare the pre-flight prediction to the actual data.

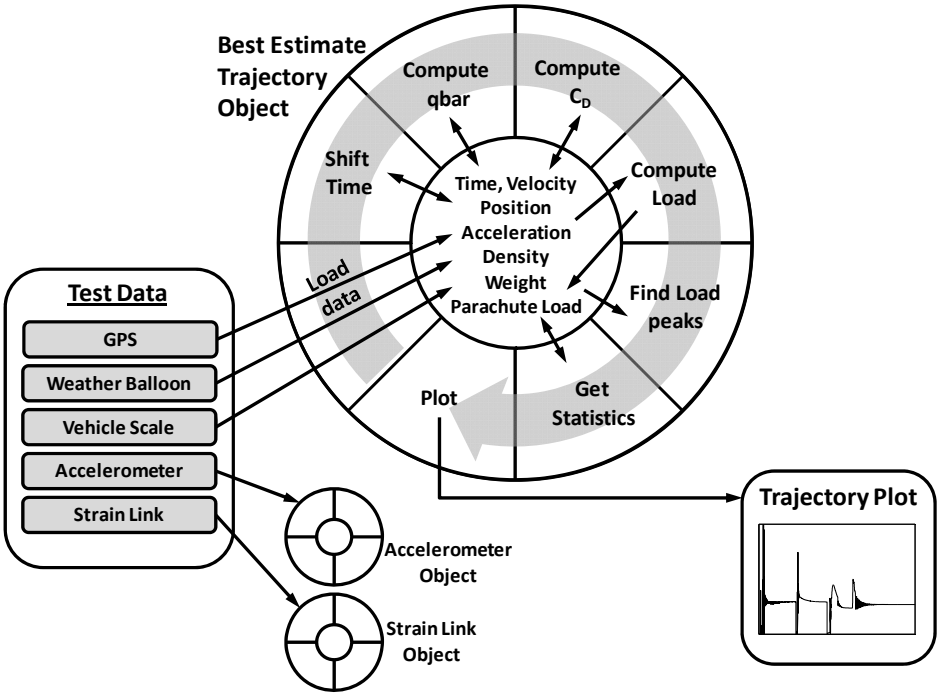

Figure 7. Sample test data processing. *The physical instruments are modeled as objects. The interaction of the methods with the object properties shows how raw data might be transformed into a Best Estimate Trajectory.*

The trajectory class was originally designed for use with simulation data but it was easily extended to accommodate actual flight data. The sensor object class is a super-class that inherits the trajectory object parameters and methods. It extends the trajectory class by adding methods to load data from the standard CPAS avionics data files, as well as text files and spreadsheets. The sensor class includes a special timing data structure that is used to interpret discrete events recorded by the avionics system. These events include a pin-pull that occurs when the test article clears the aircraft ramp and the cut-away or mortar-fire commands issued by the avionics system. The analyst may also add information from a video timeline of the test to assist in data synchronization.

In addition to reading and plotting data, the trajectory and sensor objects include methods that perform various calculations common to parachute analysis. Many of these computations are simple but prone to error if not applied carefully. Locating the code in a common object class reduces the likelihood of a calculation error. The important algorithms are coded in one place where they can be validated and then re-used. These calculation methods include:

- converting between various time formats and references and synchronizing data
- finding the average and extrema in a user-defined section of the data
- computing airspeed from collected vehicle velocity and wind data
- applying measured air density to compute dynamic pressure and equivalent airspeed
- converting between altitude references, including finding the local ellipsoid-to-geoid conversion
- computing latitude and longitude from Cartesian position
- computing parachute riser loads from accelerometer data and inferring C_DS
- computing C_D from steady state rate-of-descent, weight, and vehicle C_DS
- fitting data to empirical formulae

These functions are used to process sensor data for post-flight analysis and for comparison to simulation data. Uncertainty in each parameter is stored along with the actual data array. The calculation methods also propagate the uncertainty in the original data to uncertainty in the resulting parameter. This allows error bars to easily be added to any plot using the plot method described above.

Figure 7 is a schematic of flight data processing with the PSE. The physical instruments may be modeled as sensor objects such as an accelerometer object, strain link object, and a Best Estimate Trajectory (BET) object. The detail in the BET object shows how the interaction of the sensor methods with the object data might transform the raw flight data into a final product. Weight, atmosphere, and GPS data are loaded into the object. Moving in a clockwise direction, the *Shift Time* method synchronizes the data with other objects or with a video timeline. The

Compute qbar method calculates the dynamic pressure using the measured atmospheric density and an airspeed derived from the wind (balloon) and vehicle (GPS) velocity properties. The *Compute* C_D method calculates the parachute C_D using the object weight and velocity properties. The *Compute Load* method populates the parachute load property using the weight and acceleration properties. The *Find Load Peaks* and *Get Statistics* methods extract key characteristics of the data set. Finally the *Plot* method generates a standardized plot as shown in Fig. 6.

Several methods have been included to perform basic data clean-up and processing. These methods include functions to identify and remove spurious data spikes, normalize data to a user-defined mean value, and removing sections of data that are not of interest. These functions are convenient in post-flight analysis since they can be applied to reduce noise in the data, remove biases, and delete data that was collected by a sensor before or after the time of interest.

Finally there are several data export methods that make it convenient to create data files that can be read by other tools. Trajectory data can be reformatted to simple text files that are easily readable by many software packages. Data collected from one of the trajectory simulations can be reformatted as input to the Sasquatch[1] footprint tool. Analysts are commonly asked to provide a text-based trajectory timeline that highlights key events that occur during a test. A method is included to extract the key sequence events from the simulation data and use this information to build a timeline file. Any trajectory object with latitude, longitude, and altitude data can be exported to Google Earth™. These functions speed the transfer of data between tools and standardize the handoff of data between analysis groups.

V. Conclusion

The common output aspect of the simulation environment described in this paper has been used for trajectory predictions and post flight analysis on many of the Generation II CPAS drop tests. It has facilitated the extension of a Monte Carlo tool originally built for DSSA to the other NASA maintained parachute simulations. The generic nature of the data structure and plotting capability has proven to be versatile and has allowed analysts to extract pertinent information from very large data sets without the need to resort to custom analysis codes and scripts. The PSE tool will continue to be refined as new analysis needs develop.

References

1. Bledsoe, K., "Development of the Sasquatch Drop Test Footprint Tool", *21st AIAA Aerodynamics Decelerator Systems Technology Conference*, Dublin, Ireland, May 2011 (submitted for publication)
2. Cuthbert, P.A., "A Software Simulation of Cargo Drop Tests," *17th AIAA Aerodynamic Decelerator Systems Technology Conference*, Monterey, California, May 2003, AIAA Paper 2003-2132.
3. Cuthbert, P.A., Conley, G. L., Desabrais, K. J., "A Desktop Application to Simulate Cargo Drop Tests," *18th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, Munich, Germany, May 2005, AIAA Paper 2005-1623.
4. Moog, R.D., et al., "Parachute Simulation User's Guide Computer Program UD233A," *Martin Marietta Corp., Denver Aerospace Division*, Denver, Colorado, USA, Feb. 1986.
5. Morris, A.,et al., "Summary of Generation II CPAS Parachute Performance", *21st AIAA Aerodynamics Decelerator Systems Technology Conference*, Dublin, Ireland, May 2011 (submitted for publication)
6. Morris, A.,et al., "Summary of CPAS Test Techniques", *21st AIAA Aerodynamics Decelerator Systems Technology Conference*, Dublin, Ireland, May 2011 (submitted for publication)
7. Nemeth, S. M., "Hybrid Simulation Technology: The Next Step in the Evolution of Spaceflight Simulations", *SpaceOps 2008 Conference*, Heidelberg, Germany, May 2008, AIAA Paper 2008-3334
8. Taylor, A. P., Murphy, E., "The DCLDYN Parachute Inflation and Trajectory Analysis Tool – An Overview", *18th AIAA Aerodynamic Decelerator Systems Technology Conference and Seminar*, Munich, Germany, May 2005, AIAA Paper 2005-1624.