DSENDS: Multi-mission Flight Dynamics Simulator for NASA Missions

Jonathan Cameron*, Abhi Jain, Dan Burkhart, Erik Bailey, Bob Balaram,
Eugene Bonfiglio, Håvard Grip, Mark Ivanov, Evgeniy Sklyanskiy
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109

Increasingly complex space missions require powerful and flexible simulation environments in order to design, analyze, and operate the missions. NASA's Jet Propulsion Laboratory has created the DSENDS simulation environment that addresses these needs for a wide range of space missions. In this paper, we describe the DSENDS simulation environment and the key architectural components that make DSENDS a useful simulation and analysis framework. We also overview a variety of NASA missions and flight experiments that are using DSENDS.

Nomenclature

ARRM	Asteroid Rendezvous Robotic Mission, also known as Asteroid Redirect Mission (ARM)
COMPASS	Core Operations, Mission Planning, and Analysis Spacecraft Simulation
DEM	Digital Elevation Map
DSENDS	Dynamics Simulator for Entry, Descent and Surface landing
EDL	Entry, Descent, Landing
GNC	Guidance, Navigation, and Control
LDSD	Low-Density Supersonic Decelerator
MDNAV	Mission Design and Navigation
MSL	Mars Science Laboratory
RCS	Reaction Control System
SIAD	Supersonic Inflatable Aerodynamic Decelerator
SLS	Space Launch System

I. Introduction

Closed-loop flight dynamics simulations play a critical role in the development of flight mission concepts and flight systems, and during mission operations. With the increasing complexity of flight systems, high-fidelity simulations can be complex and expensive to develop. *Dynamics Simulator for Entry, Descent and Surface Landing (DSENDS)* is a multi-mission flight dynamics modeling and simulation tool developed for modeling of atmospheric Entry, Descent, and Landing (EDL) spacecraft at the DARTS Lab at NASA's Jet Propulsion Laboratory (JPL).[1] The goal of *DSENDS* is to promote simulation model reuse and manage simulation complexity, while addressing the diverse fidelity and high-performance needs for mission applications. Although *DSENDS* has roots as far back as the Cassini space probe in the early 1990s, *DSENDS* has evolved significantly in recent years and includes new and versatile features to enable application to many types of space missions.

DSENDS has been, and is currently in active use by several flight missions and technology development efforts including the Mars Phoenix, Mars Science Lab (MSL), InSight landers, the Low-Density Supersonic

*Senior Member of Technical Staff, Mobility and Robotics System Section

Decelerator (LDSD) technology demonstrator, precision landing technology development and asteroid retrieval mission concept development (ARRM).[2,3] A recent collaboration with Johnson Space Center (JSC) Flight Operations Directorate team uses *DSENDS* as a foundational toolkit for ascent, descent, and rendezvous operations for the International Space Station (ISS), Space Launch System (SLS) and future flight missions.

In this paper we describe key architectural features of *DSENDS* that enable its multi-mission use and several case studies that illustrate its application. While *DSENDS* simulations use C++ for fast performance speed, an extensive Python front-end provides analysts and developers with a convenient and sophisticated command line and scripting environment. The organizing structure for simulations within *DSENDS* is a hierarchy of object-oriented and configurable assemblies. Assemblies represent functional subsystems which can be contained within other assemblies, and can contain sub-assemblies of their own. For instance, a spacecraft assembly may contain assemblies for sensors, actuators, fuel manifolds, and these in turn can contain sub-assemblies for thrusters, valves, *etc.* The assemblies are configurable and parameterized to support a variety of configurations and parameter needs. Individual assemblies can be unit tested independently before use within simulations. Assemblies are responsible for the creation of component models, bodies, nodes and frames and setting up the data-flow between models. They also provide a home for sub-system specific methods. We provide an overview of key assemblies in the *DSENDS* library that are available for mission-specific simulation development.

The *DSENDS* simulation environment is one of a family of application domains that build upon the *Dshell* simulation framework.[4] The key simulation domains supported by *Dshell* framework are shown in Figure 1 and include robotics (*RoboDarts*[5]), ground vehicle simulations (*ROAMS*[6]), and spacecraft simulations (*DSENDS*). Although this paper focuses on *DSENDS*, the architecture under girding *DSENDS* is common to all of these applications and the component models from the different domains are interoperable.

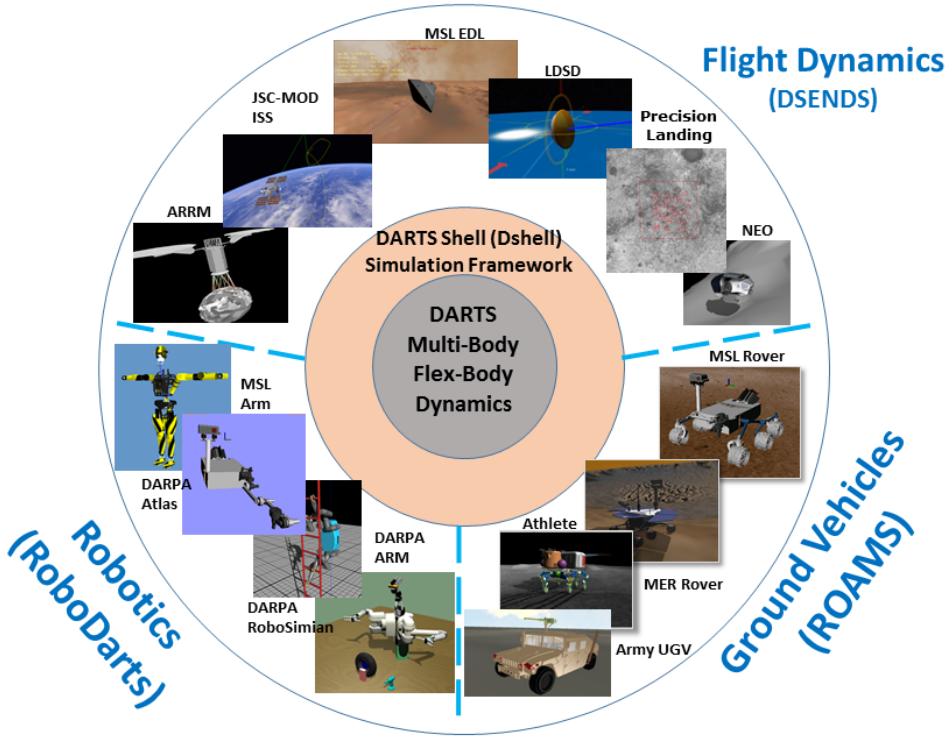

Figure 1. DARTS Lab Multi-Domain Simulations

The first part of this paper describes some of the main features of the current version of the *Dshell* framework and details of specific adaptations for *DSENDS*. In the second part of this paper we present an overview of key mission uses of the *DSENDS* toolkit including NASA flight missions such as ISS, Space

Launch System (SLS), MSL, Phoenix, Insight, and ARRM missions. For further details of the use of *DSENDS* and related software, please see the DARTS Lab publication list.[7]

II. Dshell Simulation Framework

DSENDS and related robotics and ground vehicle simulations make use of *Dshell* framework features such as: applicability to a wide range of systems, re-usability of components, object-oriented design, fast simulation speed, extensive read/write access to simulation variables, portability, modularity, *etc.* *Dshell* includes several core component layers illustrated in Figure 2. At the center of this figure is the *Dshell* module that handles numerical integration, model creation, managing and executing a model data-flow at run-time, *etc.* *Dshell* relies on modules such as *Dspace* for visualization, *SimScape* for terrain models, *DARTS* for multibody dynamics models, and a library of reusable component *Dshell models*. *DARTS* uses algorithms from the *Spatial Operator Algebra* multibody dynamics methodology[8] for solving the equations of motion and related dynamics computations. In the rest of this section we describe some of these modules that are part of the *Dshell* framework.

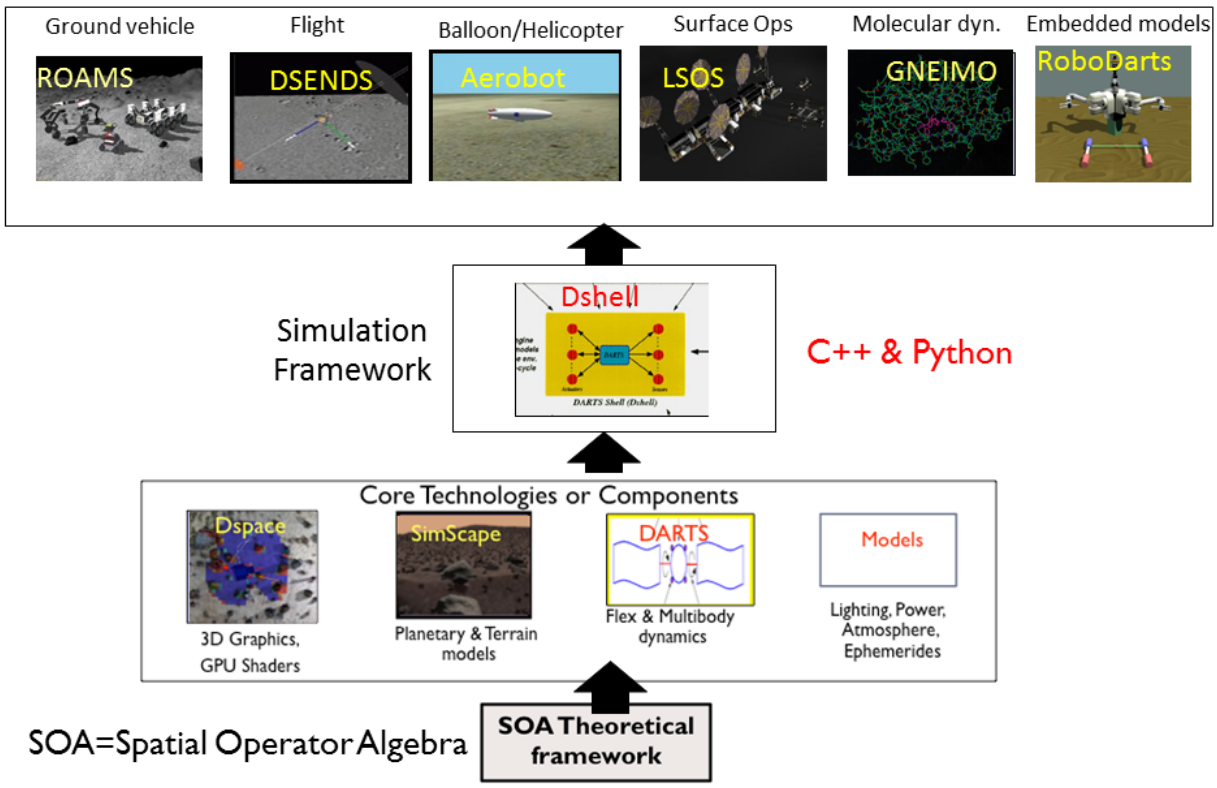

Figure 2. DARTS Lab Simulation Architecture

II.A. Frames

Coordinate systems are an essential part of any physical simulation. For instance, there is usually a global reference coordinate system for any simulation. Bodies require a reference coordinate system in which the moments of inertia are defined. Bodies typically have points of interest which we call "nodes" for locations such as a body's center of mass, points where forces can be applied, or points where positions and velocities are needed. Determining the relative positions and velocities between various locations and coordinate systems is an operation that is often needed.

The *Dshell* system defines a *frame* class as a coordinate system having its origin at a specific location. Further, *Dshell* defines bodies, sensor/actuator nodes, planetary bodies, and other points of interest as specializations of frames (as shown in Figure 3). Frames are a foundational layer of the *Dshell* simulation environment and *Dshell* implements a tree of all the frames in the simulation. Since all of the frames are part of the same tree, it is straightforward to traverse the connecting path in the tree between any pair of frames to compute the relative transform (position with rotation), velocity, and acceleration between two frames regardless of whether they represent bodies, nodes, planetary bodies, or other types of frames. Since the relative frame API is simple and consistent, the need for user code for computing relative transforms and related quantities is significantly reduced and eliminates this source of error and the burden of validating the corresponding code.

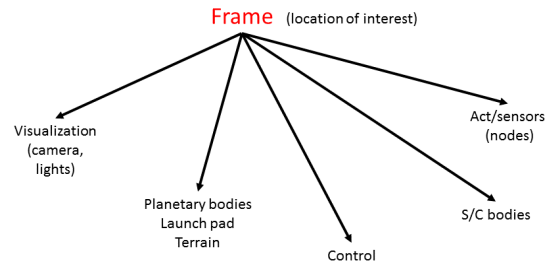

Figure 3. All points of interest in the simulations are types of *Frames*

The frame tree uses lazy evaluation to make the computations very efficient. In other words, the relative transforms between frames are only computed on demand, and only if the cached values are stale. For example, the relative position between two adjacent limbs in a multi-limb robotic arm depends only upon the relative joint angle for the two limbs. The transform between the two limbs is unchanged and can be cached and used until the joint angle changes. In the event that the joint angles change, the transform is recomputed when the relative transform is requested. Any subsequent relative frame computations that traverse that joint can use the cached relative transform without recomputing it.

II.B. DARTS multibody dynamics

A key backbone module for the *Dshell* simulator is the *DARTS* multibody dynamics module for solving the equations of motion.[8] *DARTS* creates an instance of the multibody world which is populated with component bodies by individual *Dshell* assemblies. The bodies are organized into a topological graph. One of the salient features of *DARTS* is its use of a minimal coordinates formulation for the multibody dynamics model. The minimal coordinates approach results in a dynamics model of small size when compared with the more commonly used absolute coordinates dynamics models. A significant advantage of the minimal coordinates model is that most of the inter-body articulation constraints are eliminated from the dynamics model, and this typically allows for the use of ordinary differential equation solvers instead of the more complex differential-algebraic solvers and constraint error management techniques required with absolute coordinate modeling approach. The apparent hurdle presented by the added complexity of the minimal coordinates model is handled by the use of the theory and algorithms from the *Spatial Operator Algebra (SOA)* methodology[8] that provides low-cost, O(N) structure-based algorithms that are the fastest available for solving the multibody equations of motion. These algorithms extend to rigid as well as deformable bodies. For multibody systems with internal loops, the system has graph instead of tree topology, *DARTS* includes an implementation of the novel constraint embedding technique that allows the use of the ordinary differential equation methods even for these systems.[9] *DARTS* also has dynamics solvers for non-smooth dynamics for contact and collision dynamics.

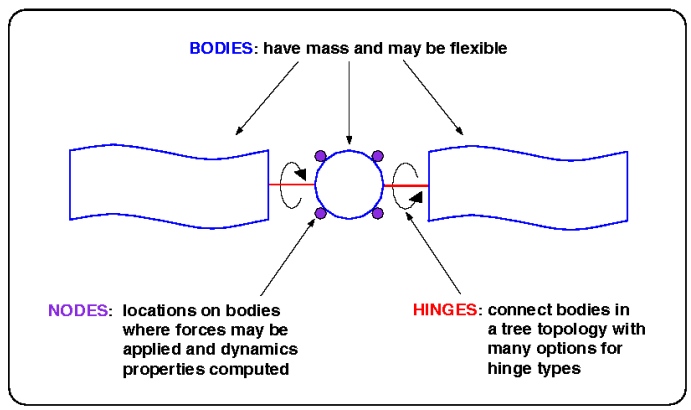

Figure 4. DARTS multibody dynamics solver.

DARTS allows for topological changes to the multibody model during run-time from the detachment and attachment of bodies, as well as the deletion and creation of bodies. The structure-based nature of the *DARTS* algorithms allows them to seamlessly adapt to such topological changes. The structure-based nature of the dynamics algorithms also allows the easy restriction of the operation of dynamics computations to "subgraphs" of the overall multibody model. This can be very useful for restricting computations to individual vehicles in a multi-vehicle simulation or to component sub-assemblies such as robotic arms or propulsion systems.

II.C. Dshell component models library

Dshell models are one of the key building blocks of *Dshell* simulations. *Dshell models* are low-level, component simulation models that interact with the *DARTS* multibody system or model subsystems independent of the multibody system (such as a battery). System devices such as thrusters, IMUs, encoders, spring-dampers, *etc.* are typically modeled as individual *Dshell models*. Models are interconnected to form a data-flow (see Figure 5). The execution of the data-flow at run-time is carried out by a sequential execution of the model methods. The connectivity between the models determines the model execution order (so that upstream models are evaluated before downstream models). When the model inter-connections form a closed loop, the user can define a virtual cut to allow the model sorting to determine the model execution order. *Dshell models* are written in C++ for fast execution speed and support user defined parameters, inputs, outputs, scratch variables, discrete states, and continuous states. The *Dshell model* C++ code builds upon a layer of auto-generated C++ boiler-plate code for peeking/poking at model variables, selecting model verbosity, state rollback, *etc.*

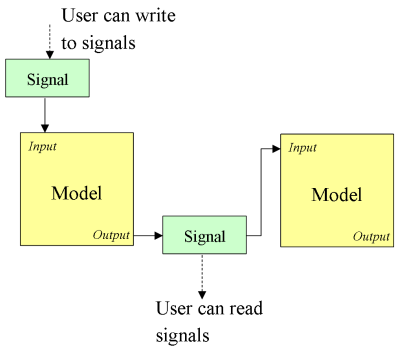

Figure 5. Dshell models and signals

Dshell models are initialized by parameters provided by the user during simulation setup. *Dshell models* use "signals" to send data between *Dshell models* as shown in Figure 5. Signal data can be accessed by users for inspection and logging and for setting their values.

Dshell models contain instances of body nodes and hinges that they are attached to, to allow them to interact with the multibody dynamics model. The models can obtain location and state information from the nodes and hinges, and, in turn, apply forces and torques at these nodes and hinges. The narrow interface between the *Dshell models* and the *DARTS* body instances helps compartmentalize and modularize the model implementations, and avoid added complexity from unnecessary cross-coupling across the device models and the *DARTS* multibody dynamics model.

II.D. Subsystem Assemblies

Although it is possible to create a full simulation by creating individual component *Dshell models*, this can be a complex and error prone process due to the large number of models and inter-connects for even moderate size simulations. *Dshell* uses *Assemblies* to organize and simplify the creation of complex simulations. In contrast with *Dshell models* which represent are low-level models, assemblies represent higher-level reusable models that are containers for multiple bodies, *Dshell models*, signals and other sub-assemblies. Figure 6 describes the decomposition of an example simulation involving multiple spacecraft into a hierarchy of assemblies. Note that the multiple thrusters and fuel tanks are multiple instances of the respective thruster and fuel tank assembly classes.

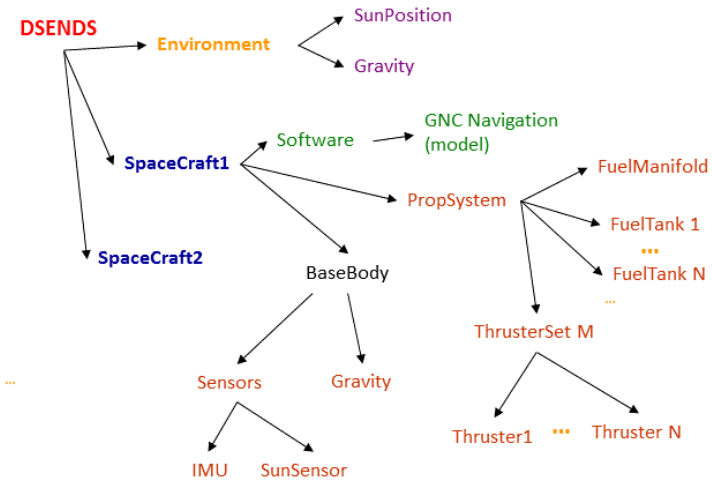

Figure 6. Example assembly hierarchy for a Dshell simulation

Assemblies use input *configuration* data to for selecting variations of models created by the *Assembly* and the topology of the constructed models. For instance, the *ThrusterSet* assembly receives configuration information about the types and number of thrusters. It then loops through and creates a *Thruster* assembly for each of the thrusters. In the process, the *ThrusterSet* assembly passes the thruster type down to each *Thruster* assembly as configuration data. Each *Thruster* assembly then constructs the desired thruster and connects its input and output signals appropriately.

Assemblies also use *context* data to define how the assembly relates to it parents assembly, models, and bodies. For instance, the *ThrusterSet* assembly passes a unique *DARTS* node object as context input to each child *Thruster* assembly that is used as an attachment point for the thruster.

Once the full simulation is constructed, parameters are used to set *Dshell model* parameters values. Since assemblies are hierarchical each assembly is responsible for using its input parameters and passing on selected parts to its children assemblies. Thus the *ThrusterSet* assembly uses its input parameter data to update the parameters of its own *Dshell models*, and passes parameter data down to its children *Thruster* subassemblies so that they can update their *Dshell model* parameters. This process also helps manage the parameter fan-out process so that parameter variables that are shared by multiple assemblies only need to be updated in one place at the upper level assembly instead of at each of the lower-level assemblies.

Thus assemblies have the desirable software characteristics of encapsulation, information hiding and modularity that not only simplify the modeling complex systems, but also enhance re-usability by providing a well defined interface to their contained models and sub-assemblies and making them easier to use by other higher-level assemblies. Assemblies can also be unit tested for desired behavior prior to using within larger simulations.

II.E. DVar data access

An important capability of any simulation is the ability to read or modify data variables within the simulation. *Dshell* simulations involve a large set of data variables that users need to access within simulations. A common need is for logging and/or plotting data from simulations for analysis. Also, a user may need to initialize or modify data variables during run-time.

Reading and modifying data variables within the simulation in a uniform way is made possible using the *DVar* data access layer within *Dshell*. *DVar* provides a data tree that includes all simulation data variables. The location of each data item in the *DVar* tree is represented by an "address", a unique dot-separated string identifier that can be used to access the data variables. The *DVar* data access layer supports normal read and write operations for the data variable.

The types of data variables that automatically get registered in the *DVar* data tree include all *DARTS* multibody variables; all *Dshell model* scratch, state, and parameter variables, as well as all signals. The uniformity of the *DVar* data representation makes it easy to access *DVar* objects for reading, modifying, logging, run-time data plotting, introspection, and population of graphical user panels in a convenient and consistent way.

II.F. SimScape terrain models

Dshell uses a terrain modeling toolkit called *SimScape* to model planetary bodies, local terrain patches, DEMs (digital elevation maps) and irregular shapes (such as asteroids) using meshes.[10] *SimScape* has two primary purposes: (1) to provide tools to import terrain data from a wide range of sources into terrain models that *Dshell* simulations can use at run-time (see Figure 7); and (2) to provide a library used at run-time to support loading terrains and execute queries for topography of terrains, intersection of rays with terrain surfaces, slopes, visualization, and more. *SimScape* is often used in *Dshell* simulations to model spacecraft sensors such as LIDARs, cameras, altimeters and proximity sensors as well as for landing and terrain interaction models. *SimScape* stores its terrain data using the *Hierarchical Data Format (HDF5)* data storage format for efficient and flexible access.[11] The HDF5 data format is hierarchical and allows *SimScape* to deal with large data sets since selected data slices can be loaded efficiently on demand at run-time. HDF5 supports "striding", accessing every nth row or column of data to sub-sample the data.

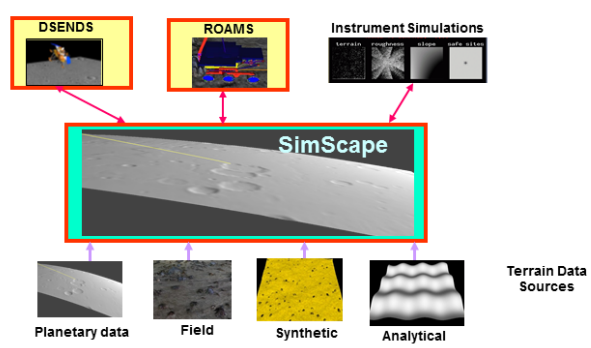

Figure 7. SimScape terrain environment toolkit to facilitate the use of terrain models in a range of simulation applications.

II.G. Geometry Data and Visualization

Simulations often need geometry data for the various bodies in the system for visualization as well as collision and contact dynamics computations. A geometry manager module, called *DScene*, is used by *Dshell* to isolate the main simulation from the consumer modules. The simulation defines geometrical shapes for the component bodies via shape primitive such as cuboids, cylinders, *etc.* as well as via general mesh formats used by graphics applications. *DScene* supports a plugin architecture where geometry consumers can register themselves with *DScene* as shown in Figure 8. The plugin architecture allows *Dshell* simulations to to add in new geometry manipulation modules with custom interfaces without impacting the core *Dshell* simulation itself. At run time *DScene* creates a corresponding geometry world in each of the registered plugin clients, and syncs up their positions and orientations with that in the simulation world.

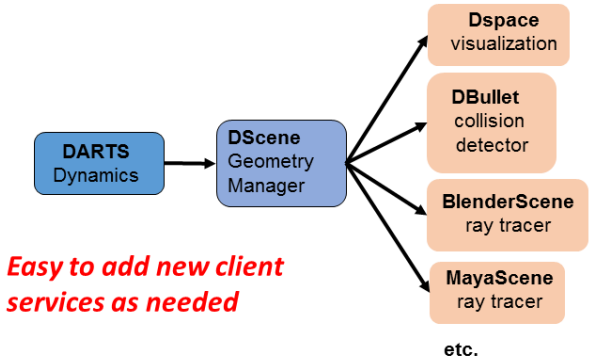

Figure 8. The DScene geometry manager and example geometry client modules.

One such *DScene* plugin in the *Dshell* environment supports 3D graphics visualization and is called *Dspace*. *Dspace* is implemented using the Ogre3D open-source visualization library that is also used in gaming applications.[12] For collision detection, *Dshell* uses a *DScene* plugin based on the open source Bullet engine.[13] *Dshell* also supports using *DScene* geometry plugins for higher-fidelity visualization based on ray-tracing techniques.

For large *SimScape* terrain models, *Dspace* includes a GPU-based continuous level of detail algorithm for seamlessly managing the rendered polygon count for the smooth real-time visualization of the terrain data.[14]

II.H. Simulation Infrastructure

Dshell implements all run-time code in C++ for best performance. For convenience and flexibility, *Dshell* uses the Python interface for constructing the simulation, gluing together all the C++ parts, and initializing all the run-time parameters. This is made possible by using the SWIG[15] tool that takes C++ headers and auto-generates Python binding code for accessing C++ objects and functions from Python (Figure 9). Even though much of the simulation setup is done in Python, the *Dshell* simulation can be configured as a stand-along C++ library with API calls so that it can be embedding into other run-time environments. Also, external C, C++ and FORTRAN libraries can also be linked into *Dshell models*.

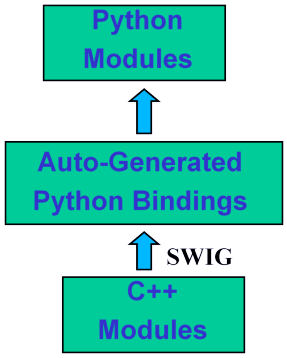

Figure 9. SWIG generates Python bindings for Dshell C++ classes

The *Dshell* simulation can be run as a server, with an external process controlling the *Dshell* simulation functions. *Dshell* can also be run as a client to interact with external server programs such as Matlab. *Dshell* includes a wide range of numerical integrators, from simple fixed-step Runge-Kutta to variable-step ordinary differential equation integrators as well as differential algebraic equation solvers for systems with constraints. *Dshell* provides integrators for smooth problems as well as integrators suitable for stiff problems. *Dshell* also provides explicit and implicit integrators as well as propagators for non-smooth contact dynamics.

III. DSENDS

The *DSENDS* simulator builds upon the *Dshell* framework, and contains a wide range of models and assemblies to support space missions including thrusters, fuel tanks, gravity, aerodynamics, spring-dampers, motors, encoders, IMUs, cameras, robotic arms, and more. An overview of some of the available assemblies is shown in Table 1. In applications to specific missions, these generic assemblies and associated *Dshell models* are typically tailored for the specific needs of missions.

Table 1. Examples of supported generic assemblies for DSENDS simulations

Assembly **type**	**Description**
General	
Vehicle	Creates a spacecraft vehicle (with support for a variety of spacecraft functionality)
Mechanisms	
Arm	Creates a robotic multi-link arm
Encoder	Adds an encoder assembly to a joint
ExternalDisturbanceActuator	Adds a user specified disturbance force and torque on a specific node on a body in the simulation
ExternalDisturbanceMotor	Adds a user-specified disturbance to joint torques or forces
Link	Creates a single link of an arm or a serial link mechanism
Motor	Adds a "motor" model to control the behavior of a joint. This assembly supports a number of different motor types including DC motors, prescribed motion, spring-dampers, and gearing.
PIDController	Creates a PID controller
ProfileUnit	Implements a joint motor that moves the joint through a trapezoidal profile
Environment	
Site	Provides information about the terrain surface site such as location
Target	Base for various target assemblies. "Targets" refer to the planetary body that is the landing target.
TargetSpice	Creates a planetary body associated with predetermined orbital motion that is controlled by NAIF Spice (see http://naif.jpl.nasa.gov/naif/)
Aerodynamics	
Aerodynamics	General-purpose aerodynamics
Atmosphere	Adds spline-based atmospheric profiles
SplineWind	Adds a spline-based wind model
Gravity	
ConstantGravity	Adds constant user-provided gravity
AsphericalGravity	Gravity model for non-spherical planetary bodies using "J" terms
GeneralGravity	Allows the user to specify changing gravity using signals
PolyhedronGravity	Gravity model for an irregular body defined by a polyhedral mesh
IMU	
IMU	Adds a basic IMU
NoisyIMU	Adds an IMU with noise characteristics
Propulsion	
FuelManifold	Connects fuel tanks and thrusters
FuelTank	Fuel tank that changes mass properties as fuel is used
PropulsionSystem	Constructs a full propulsion system with fuel tanks, manifolds, and thrusters
ThrusterFixed	Fixed-thrust thrusters with on-off inputs
ThrusterThrottled	Variable-thrust thrusters with thrust level inputs
ThrusterSet	Constructs a set of thrusters with similar characteristics

III.A. DSENDS Testing and Validation

The *DSENDS* environment is covered by a large regression test suite. The tests range from low-level (unit) tests to many system level tests. Some of these tests are in C++ but most are python test scripts.

The *DSENDS* environment has been validated in several ways. The basic aerodynamics functionality has been validated several ways against the POST2 simulation tool (from Langley Research Center).[16] Various other parts of the *DSENDS* environment have been validated against other aerospace simulation tools. Finally, the *DSENDS* has been validated by several Mars and in Earth lander missions.

IV. Applications of DSENDS

In the following sections, we describe several applications of *DSENDS* to NASA missions and flight experiments. These examples illustrate the use of *DSENDS* in a variety of mission phases including planning, design, implementation, and operations.

IV.A. COMPASS

Over the last five years, JPL has collaborated with the Flight Dynamics Division (FDD) within the Flight Operations Directorate (FOD) at NASA's Johnson Space Center. The collaboration started at the end of the Shuttle program when the FDD team needed a new simulation architecture to meet evolving future needs. The FDD team started with *DSENDS* and added their domain-specific models to create a tool called *COMPASS*.[17] *COMPASS* is used to provide flight-operations tools for the International Space Station, future Exploration Missions, and beyond.

The collaborative partnership between JPL DARTS Lab and the JSC FOD team has proved to be very beneficial for everyone for two important reasons. First, since *COMPASS* is based on *DSENDS*, FDD has benefited from previous and active JPL development and maintenance. Secondly, improvements to *DSENDS* required by FDD to support operational needs have resulted in new simulation capabilities while improving the current framework.

IV.B. Mars Science Laboratory

The most recent planetary science mission to Mars was Mars Science Laboratory (MSL) with the Curiosity rover, which launched November 26, 2011 and landed at Gale Crater on August 6, 2012. Curiosity is a rover with a significantly larger and more advanced landing payload than any previous Mars lander mission. In addition, MSL was the first use at Mars of a complete closed-loop entry Guidance Navigation and Control (GN&C) system, including guided entry with a lifting body (via center of gravity offset) to greatly reduce dispersions during the Entry, Descent and Landing (EDL) phase. The hyper-sonic entry guidance enables the entry body to fly out the remnant delivery error from the final Trajectory Correction Maneuver (TCM) and other sources, resulting in less than a 25 km 20 km landing error relative to the selected Gale Crater landing target.

The JPL MSL EDL trajectory analysis team had two major simulation actions for operations. The first was high-fidelity trajectory and simulation verification analysis of the POST2 simulation from the Langley Research Center, the primary performance simulation tool for MSL EDL analysis.[18,19] This verification analysis required including the full suite of EDL simulation models including dispersed inputs for Monte Carlo analysis of EDL. The second was EDL trajectory simulation to support cruise TCM targeting and maneuver design, which required modeling of the EDL dynamics without closed-loop GNC. Both are covered in the following section. The simulation chosen for MSL was *DSENDS*. *DSENDS* provides the framework for modeling various aero-assisted simulations (EDL, aero-braking and aero-capture) with varying complexity from simple systems with single bodies and 3 degrees-of-freedom (DOF) dynamics to multibody, flexible systems.

IV.B.1. MSL Verification and Validation (V&V)

One role of the *DSENDS* simulation for MSL was an independent verification and validation (V&V) of POST2, the primary performance simulation, from EDL start to parachute deploy. This was a natural end point for a de-scoped cross-check because the full-fidelity modeling through parachute deploy was mostly the same functionality required for other roles such as cruise maneuver targeting. As described in Burkhart

2013,[20] the *DSENDS* simulation included the highest fidelity models for all required functionality through parachute deploy and therefore provide equivalent functionality. A parachute deploy comparison of a Monte Carlo solution for both simulations is shown in Figure 10. The result shows good agreement in the size and orientation of the dispersed points, with a slight downrange shift in the *DSENDS* derived ellipse of 400m relative to the POST2 ellipse. One factor contributing to the difference was attitude dispersion, which was done differently by each simulation at run time. Since the differences were acceptable to the project, no further work was done to quantify them. The MSL project concluded that the independent V&V was successful based on these results.

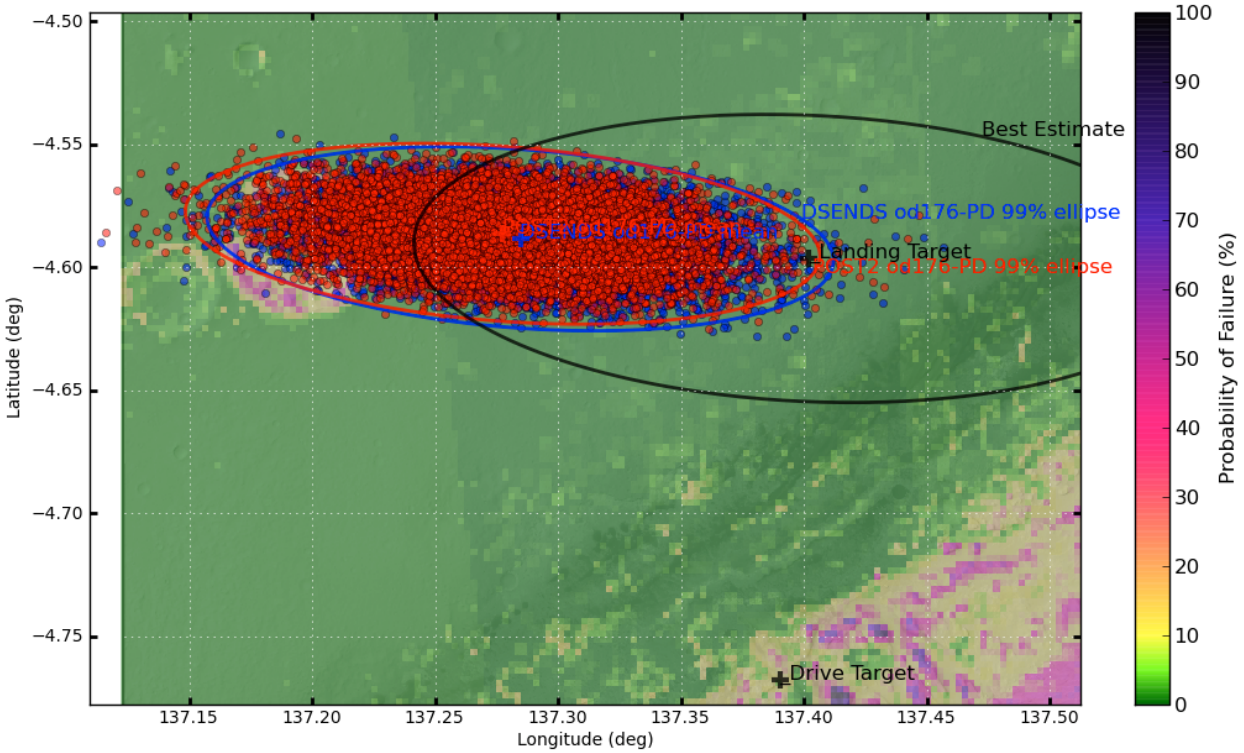

Figure 10. Comparisons of DSENDS (blue) and POST2 (red) Monte Carlo dispersions at parachute deploy for Mars Science Laboratory

IV.B.2. MSL Operations

The V&V analysis described above is a significant part of the operations tasks for the simulation team, specifically for Monte Carlo analysis. During operations, analysis of expected landed performance was performed as needed when the entry conditions changed significantly. Additional tasks were performed using the *DSENDS* simulation for operations, as described in Burkhart 2012.[21] One of these tasks is targeting, which uses an open-loop EDL setup in *DSENDS* in concert with interplanetary trajectory propagation tools to design cruise maneuvers that achieve a desired landing point on Mars. Compared with the full-fidelity simulation above, the open-loop simulation does not include closed-loop GN&C models and defines the capsule attitude during entry via trim aerodynamics. A related task is cruise stage re-contact analysis, which uses the open-loop *DSENDS* setup along with a propagation model for the cruise stage to compute how the cruise stage breaks up in the Martian atmosphere and propagate the components. Trajectories of the capsule and the various cruise stage parts are compared to verify that there is no re-contact. A final related task, used on landing day, was the monitoring of real-time radiometric data to detect EDL events. Events that impart a significant velocity change along the Earth line will be visible as discontinuities or slope changes in the Doppler data. This includes known events, such as separations and ejections of mass, antenna changes, thruster activity, drag due to atmospheric entry and parachute deploy as well as unexpected velocity changes. The open-loop *DSENDS* setup was used with varying degrees of model fidelity to generate a reference Doppler data profile that was used on landing day to compute Doppler residuals used to detect events.

One comparison models the entry phase with atmospheric drag and vertical lift but does not include out-of-plane lift, the parachute, nor any other events after parachute deploy (no heat shield release or powered flight) but instead models flight in the capsule configuration to impact. These residuals are shown in Figure 11.

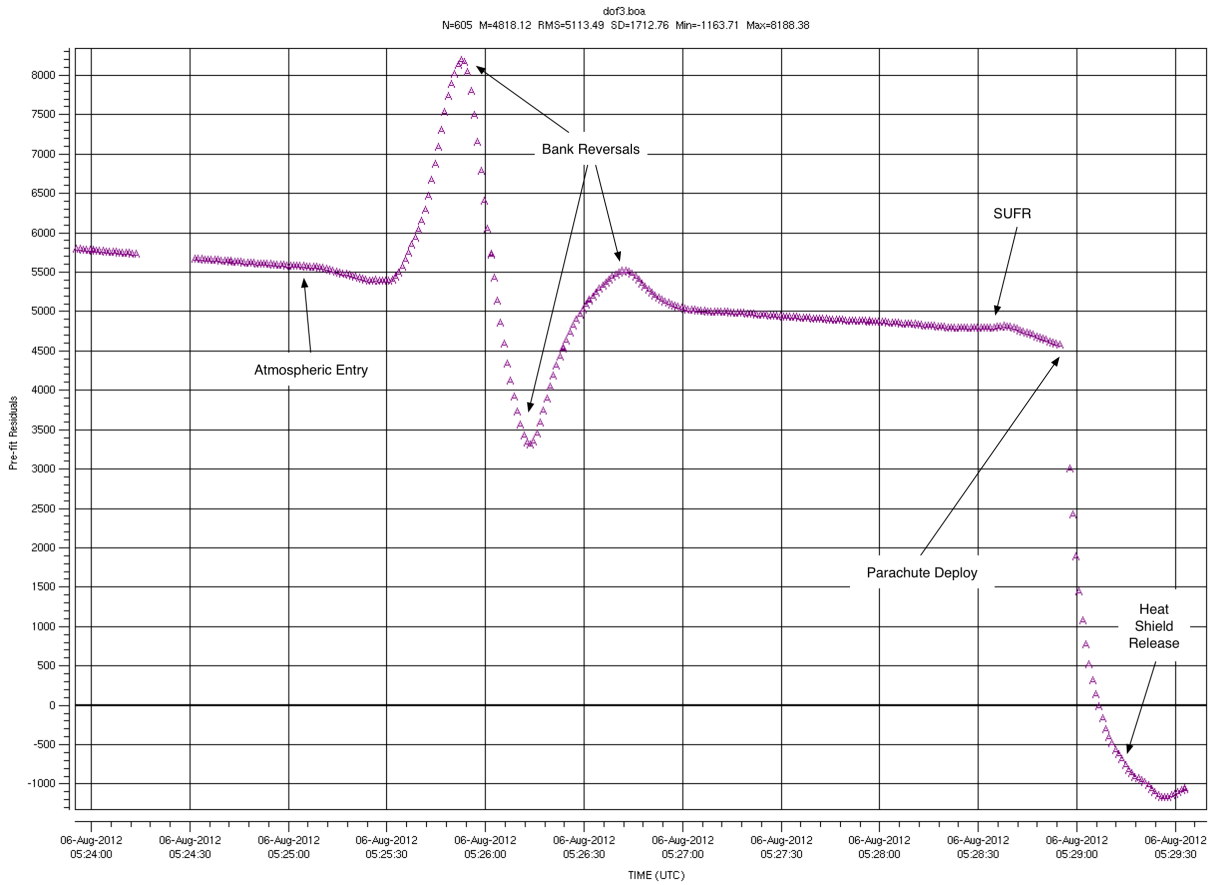

Figure 11. Real-time Doppler from entry to occultation, prediction with drag and vertical lift only. Bank reversals, SUFR (Straighten Up and Fly Right), parachute deploy and heat shield release are noted.

On the day of the flight experiments, a GNC monitoring console was used to monitor telemetry and display current spacecraft configurations using data from live telemetry.[22] This console is shown in Figure 12.

IV.C. Phoenix and Insight Missions

On both Phoenix and InSight Mars missions, *DSENDS* was used for maneuver design, landing dispersion analysis, and trajectory targeting.[23] Both landers are very similar and the application of *DSENDS* on these two projects is equally similar. EDL Systems design was conducted using POST2.[24–27] Phoenix and InSight do not use any entry guidance, the trajectories are largely uncontrolled up until powered descent, and the touchdown ellipses are on the order of 100 km, along-track, for both missions. Because of this and because it was not used for EDL systems design, the *DSENDS* simulation had significantly lower requirements than a mission such as MSL might.

The EDL simulation in *DSENDS* differs from POST2 in that it does not attempt to model the EDL system to the level of fidelity that is modeled in POST2. While the simulation does have an accurate 6-DOF model of the trajectory through parachute deploy that has been verified against POST2, it only has an engineering model of certain parts of the flight software, which were relevant to accurately predicting landing location. (Namely, the parachute-deploy triggers and separation of the lander from the parachute were modeled for Phoenix and Insight.) The parachute deploy trigger in the *DSENDS* simulation is very similar to that used by actual flight software, although the lander separation was commanded at a constant 1000-meter altitude for Phoenix and using a simple gravity-turn model for InSight.

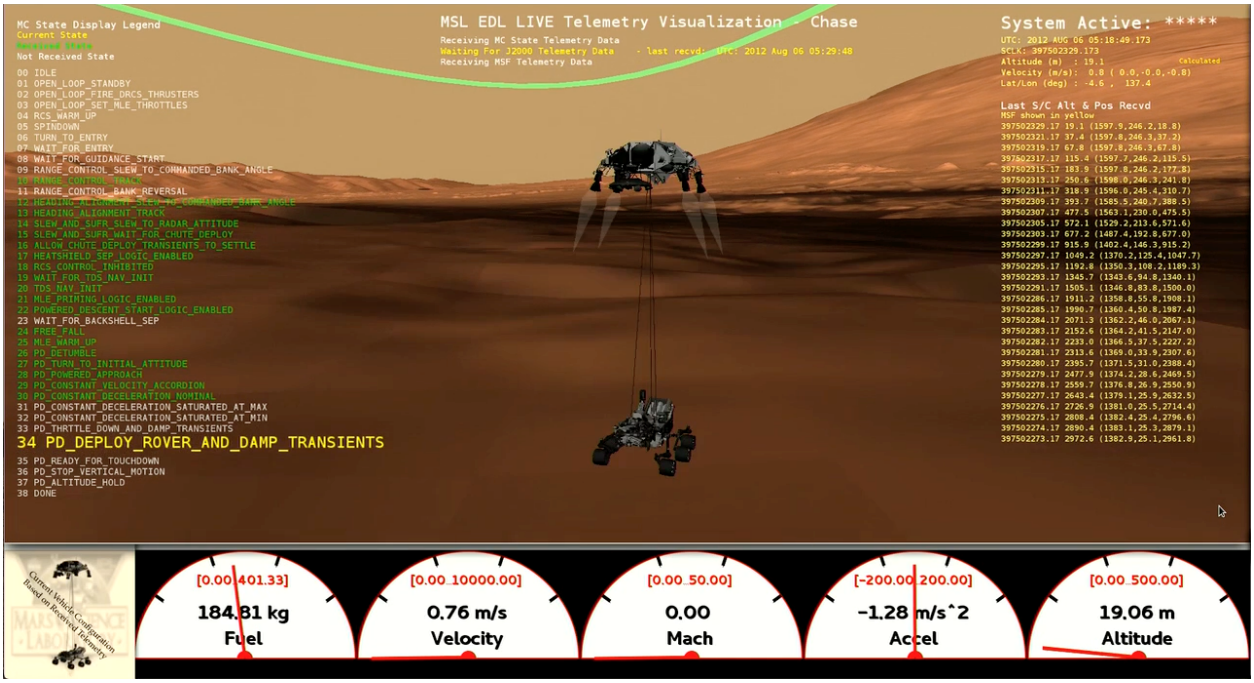

Figure 12. Mission Operations Telemetry Monitoring Console, Created Using DSENDS. Events are highlighted on the left as they occur. Spacecraft configurations are updated based on live telemetry.

DSENDS does not model the reaction control system (RCS) on the Phoenix spacecraft. This is not necessary because the attitude and attitude rate error dead bands in the flight software are increased to the point that there is almost no chance of the RCS thrusters firing during the hyper-sonic phase. The simulation has no IMU or radar modeling in *DSENDS* and, therefore, assumes perfect knowledge for all of the major events during the EDL trajectory. Finally, the descent thrusters were not modeled for Phoenix because the distance traveled over the ground during the near-vertical propulsive phase is insignificant. Table 3 is a comparison of POST2 and *DSENDS* that was presented at a certification review from Phoenix. This

Comparison Against EDL Anchor (TCM-6 Delivery ESF with Expected Dispersions)		
	DSENDS	**POST (Anchor)**
Mean Chute Time	**819.15 s**	**820.09 s**
3-Sig Chute Time	**+/-16.32 s**	**+/-16.41 s**
Mean TD Lat/Lon	**68.19/233.29**	**68.20/233.30**
Mean TD Miss Dist	**307 meters**	
Nominal Lat/Lon	**68.17/233.36**	**68.18/233.35**
Nom. TD Miss Dist	**304 meters**	
Ellipse Size (km)	**103.1 X 19.6**	**103.1 X 19.9**

Table 2. Examples of supported generic assemblies for DSENDS simulations

comparison is for a Monte Carlo run performed in both *DSENDS* and POST2 during the development phase. The comparison clearly shows the differences between the two tools (-300 meters) are minor compared to the size of the landing ellipse (-103 km). Figure 13 shows an overview of the Phoenix Entry Decent and Landing profile.

The inputs in the Monte Carlo runs capture all of the dispersions known to have a significant impact on the landing location and ellipse size. The Monte Carlo runs consists of a nominal run and 2000 dispersed runs, each with an independent set of dispersed inputs. For both Phoenix and InSight missions, *DSENDS* became an official Mission Design and Navigation (MDNAV)/EDL targeting tool. For the InSight mission in par-

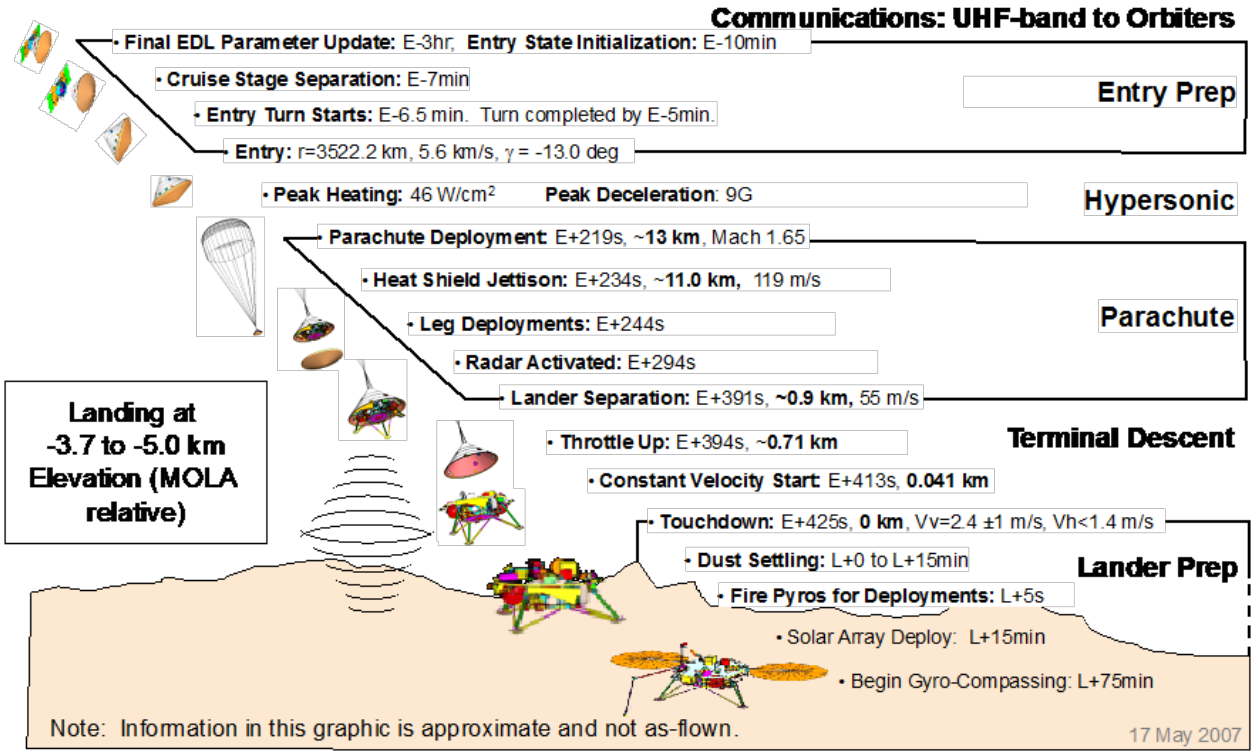

Figure 13. Phoenix EDL Time Line

ticular, the way in which DSENDS was used was substantially enhanced with the direct MONTE/DSENDS software integration. MONTE,[28–30] is a certified navigation and mission analysis trajectory program directly calls *DSENDS* for an integrated interplanetary and EDL targeting at the fixed landing location on Mars. Figure 14 illustrates the entry targeting algorithm which takes an initial guess for the B-plane angle and entry epoch, then iterates with *DSENDS* to solve for landing latitude and longitude. The targeting is completed when the change in the entry time and B-plane angle are within the design tolerance. In addition to the MDNAV support for InSight, *DSENDS* became the primary software for the landing site hazard assessment. Its role became crucial during the TCM design and the final decisions on the maneuver execution. The output of the *DSENDS* EDL Monte Carlo analysis produced a set of dispersed landing points on the ground which via LSAT (Landing Site Assessment Tool, a MONTE-based package) allowed the tool to compute the probability of the successful landing and monitor the shift in the mean landing target before and after the trajectory maneuver. Figure 15 illustrates a typical InSight landing hazard assessment analysis graph, which was frequently used in the critical TCM decision-making process.

IV.D. LDSD/SIAD

In 2012, NASA initiated the Low-Density Supersonic Decelerators (LDSD) project to develop a new generation of supersonic aerodynamic decelerators: 6-m and 8-m Supersonic Inflatable Aerodynamic Decelerators (SIADs), and a large, 30.5-m, supersonic parachute.[31] As part of the LDSD project, several new ground-based test architectures were developed for performing structural testing of the decelerators. However, to fully evaluate deployment, inflation, and supersonic and subsonic aerodynamic behaviors, a full-scale flight test was required at conditions relevant to how the technologies would be utilized at Mars. The test architecture is outlined in Figure 16.

For a nominal mission, a large helium balloon is used to hoist a 4.7-m diameter blunt body Test Vehicle (TV) to an altitude of over 36 km above Hawaii. The test vehicle is released from the balloon, spun-up for stability, and a STAR 48 solid rocket motor is ignited. The motor accelerates the test vehicle to approximately Mach 4 and an altitude of 52 km. Upon burn-out, the vehicle is de-spun and the primary test phase begins. Shortly thereafter, the first of the technologies, a SIAD is deployed. Later in the flight, the ballute or Parachute Deployment Device (PDD) is mortar-fired, inflated, and subsequently used as a pilot

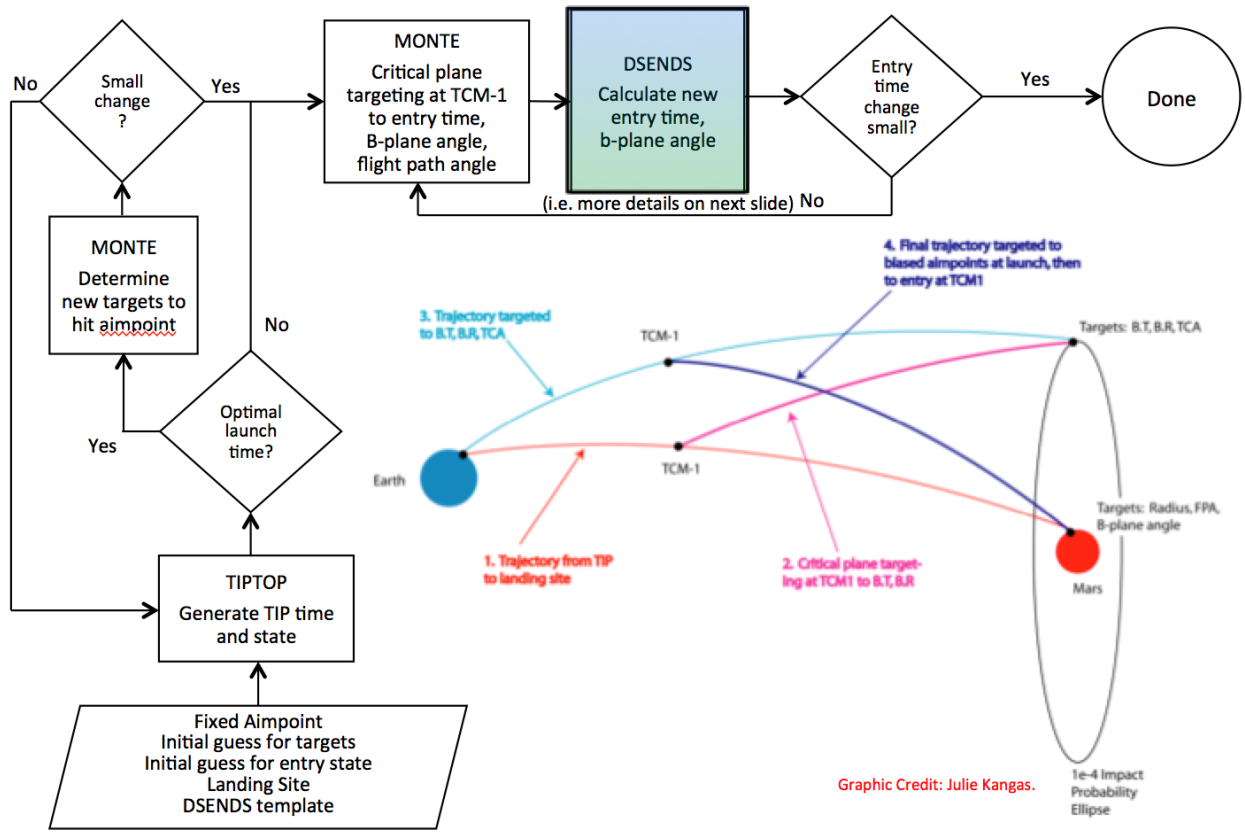

Figure 14. InSight Mission Design / Navigation Targeting Overview

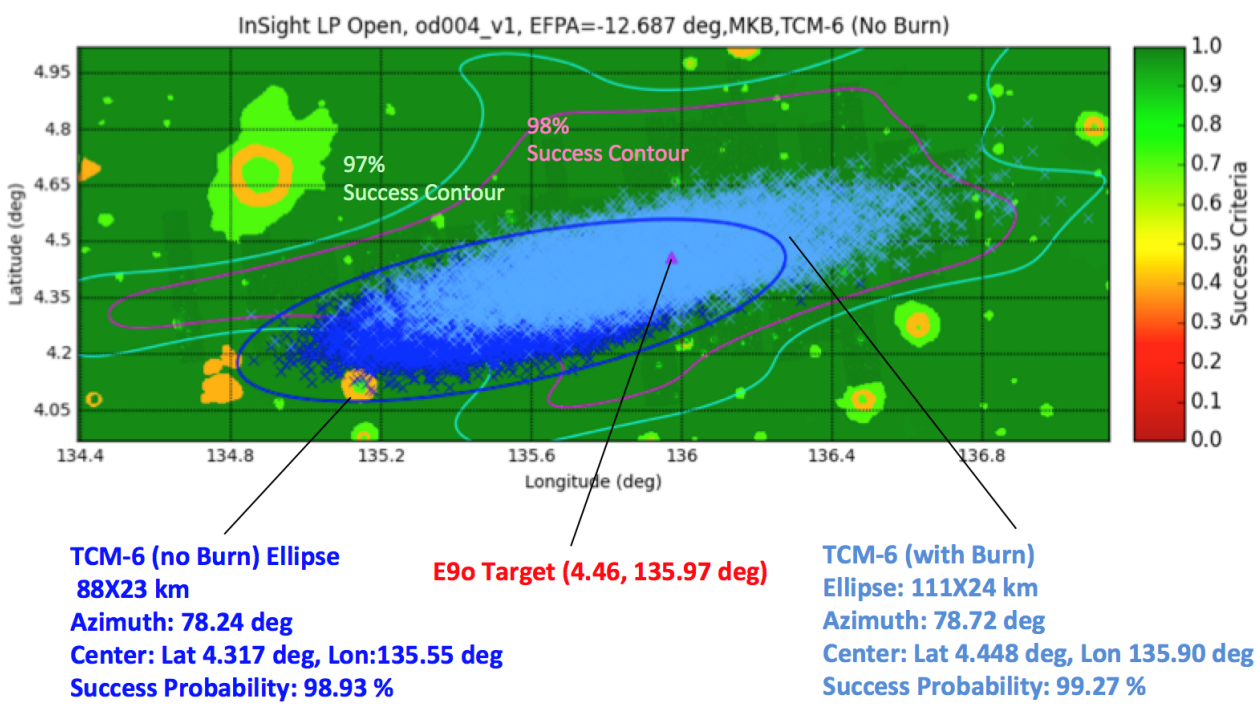

Figure 15. InSight Landing Hazard Assessment Analysis with DSENDS landing points

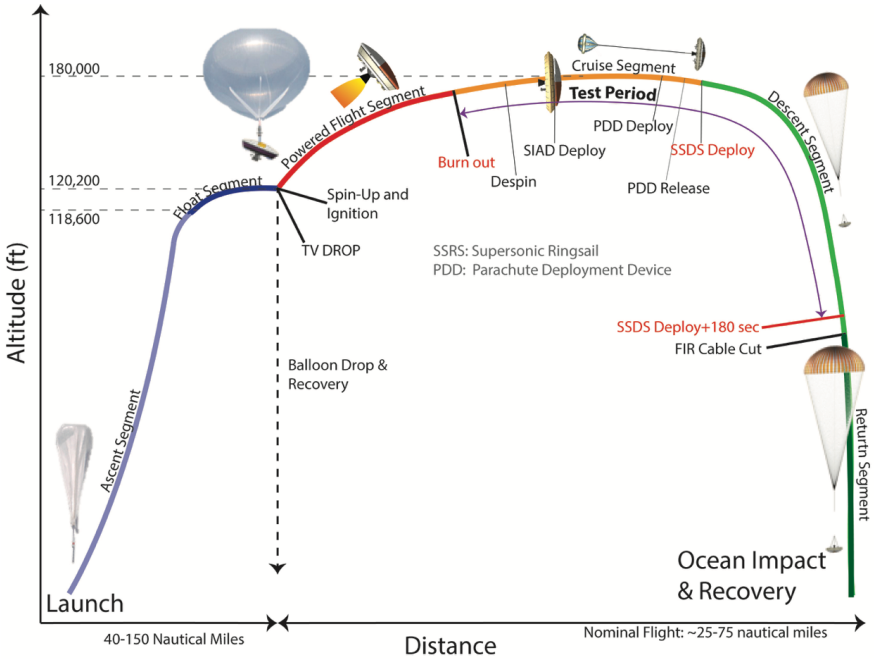

Figure 16. Low Density Supersonic Decelerator Flight Profile. These flight experiments were performed at high altitudes above Hawaii to simulate Mars supersonic entry conditions.

device to extract and deploy a large supersonic parachute from the test vehicle. The parachute decelerates the vehicle to subsonic conditions and the vehicle descends to the ocean for recovery.

DSENDS was used during all phases of the project to perform Test Vehicle flight simulations in support of systems design, systems analysis, and flight operations. *DSENDS* was utilized in a full 6-DOF mode with the integration of the following models: propulsion, atmosphere, h/w configuration, aerodynamics, flight software, and geophysical. Both nominal and dispersed/uncertainty simulations were performed. Nominal simulations provided critical insight for initial design and trajectory studies. Uncertainty analysis was performed to define TV hardware build tolerances, flight dynamics robustness, and range safety splashdown footprints. *DSENDS* was used to develop targeting data necessary to aim the vehicle to within range safety clearance areas. Finally, *DSENDS* was used to visualize telemetry data during the actual flight experiments[22] as shown in Figure 17.

IV.E. Asteroid Redirect Robotic Mission

The Asteroid Redirect Robotic Mission is a proposed NASA mission aimed at collecting a large boulder from the surface of a near-Earth asteroid and bringing it into lunar orbit. A second, crewed phase of the mission would allow human astronauts to inspect and sample the boulder while in lunar orbit. During early phases of mission planning, the option of capturing an entire asteroid - measuring up to 13 m in diameter and weighing up to 1000 metric tons - was studied in detail. The capture system in this case would have consisted of a large, flexible bagging mechanism attached to the main spacecraft bus, capable of completely encapsulating and securing the target asteroid. *DSENDS* was used extensively to study the dynamics of the capture problem, as part of the design process for the capture mechanism itself and the associated guidance, navigation and control architecture.[3,32] A visualization of the *DSENDS* simulation for this analysis is shown in Figure 18.

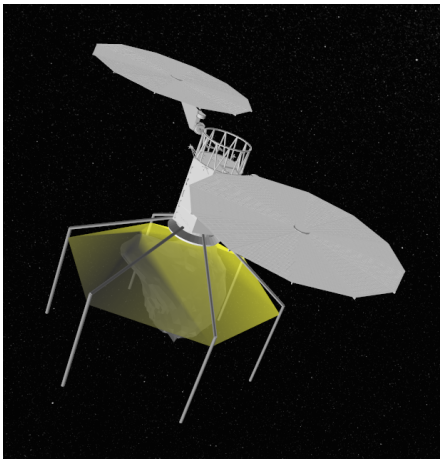

Figure 18. Asteroid Capture. DSENDS was used to study the dynamics of the asteroid capture process using a trampoline capture device.

The capture problem is complicated due to the relatively high expected rotation rate of the target asteroid and the likely tendency

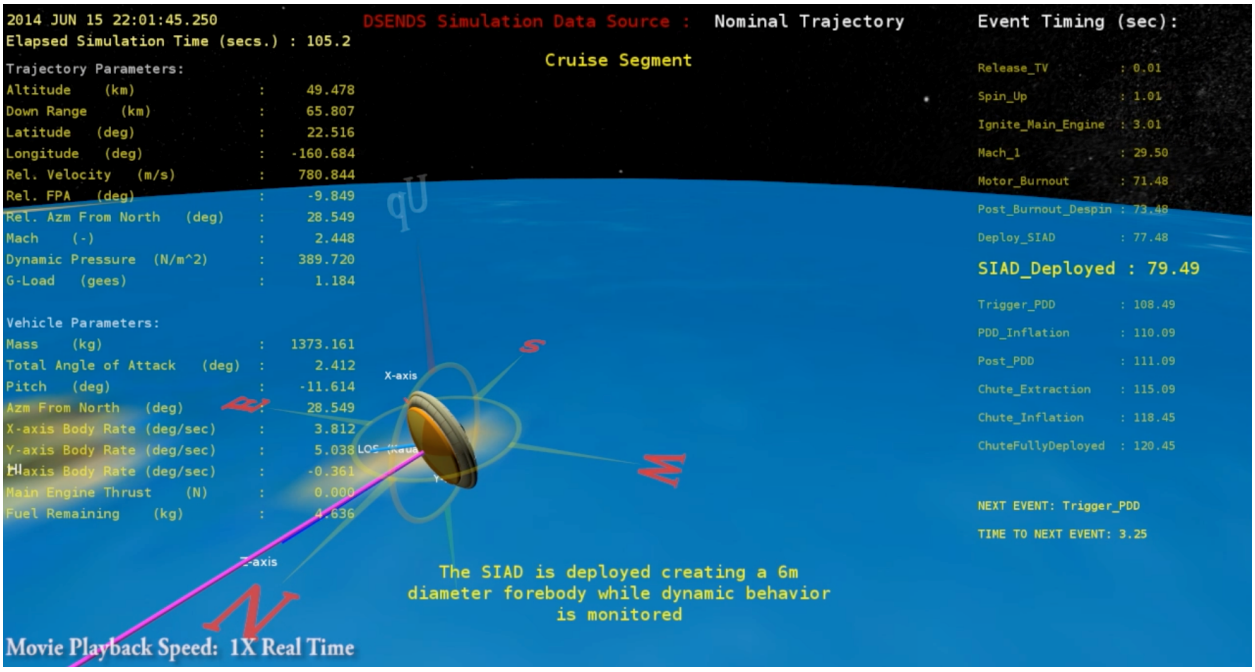

Figure 17. LDSD/SIAD Telemetry Visualization Using DSENDS

to tumble in space. The eventual capture mechanism design consisted of a trampoline-like flexible structure situated within an inflatable capture bag. The trampoline would be pressed up against the target asteroid using the spacecraft thrusters, thus naturally accommodating the shape and rotation of the asteroid itself prior to the closing of the bag. The trampoline was represented in *DSENDS* using a novel, energy-based model.[32] Modeling of earlier iterations of the capture mechanism are described in Grip et al., 2014.[3]

The ARRM *DSENDS* simulation was also used as part of a hardware-in-the-loop zero-gravity testbed designed for validation of the capture concept as shown in Figure 19. The testbed itself consisted of a stationary capture mechanism and an asteroid mounted on a robotic arm. *DSENDS* was used in real-time to simulate and visualize the motion of the the spacecraft and asteroid in zero gravity; the simulated trajectory of the asteroid relative to the spacecraft was used to prescribe the motion of the physical asteroid in the testbed. The reaction forces and torques between the spacecraft and the asteroid were measured using a force-torque sensor at the base of the capture mechanism; these were fed back into the *DSENDS* simulation and used to propagate the state. In this way, the approximate model of the flexible trampoline was replaced by a physical implementation of the trampoline. Details of the testbed are described in an earlier paper.[3]

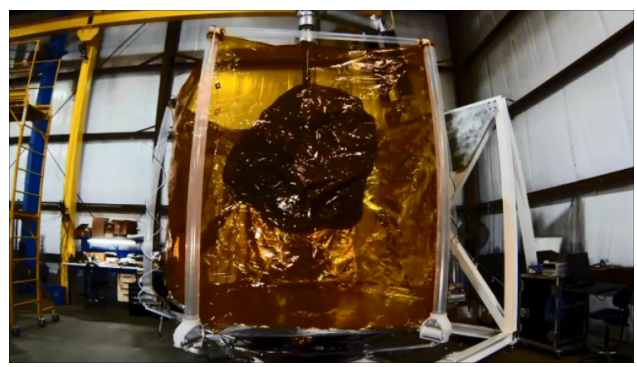

Figure 19. Asteroid Capture Testbed. DSENDS was used in real-time to emulate a zero-G environment.

V. Broader applications to future missions

DSENDS is currently being used at JPL to analyze flight vehicle performance and guidance, navigation, and control performance for various space applications, and in the future we will be deploying *DSENDS* for Uncertainty Quantification (UQ) and Sensitivity Analysis (SA) of vehicle performance in all project phases (proposal through flight operations). We currently are developing Entry, Descent, and Landing models and analysis techniques to include new terrain sensor types (most notably computer vision and LIDAR terrain sensing) and also new trajectory profiles, like supersonic retro propulsion which may or may not require deploy-able aerodynamic decelerator technologies (parachutes, SIADs, *etc.*). *DSENDS* is also deployed in

system-level UQ and SA for Mars Ascent Vehicle system design, using new design exploration tools such as DAKOTA[33] to move beyond Monte Carlo techniques.

We will continue to work with our NASA partners at other centers who develop aerodynamic, aerothermal, and environmental models for various vehicles and applications, and continue to collaborate in independent Verification and Validation of flight missions, such as InSight, Mars 2020, and potential Europa Lander. Some of these missions and mission concepts include the integration of GNC flight software, leveraging the experience of Mars Science Laboratory, on which *DSENDS* played a key EDL targeting and high fidelity V&V simulation role, which continues into the Mars 2020 lander mission. *DSENDS* has been compared with POST2 at LaRC for the MSL, LDSD, and Mars 2020 projects as well as the STAMPS tool at JSC for independent Verification and Validation of system-level flight dynamics performance.

Missions will continue to use the *DSENDS* visualization engine for development and flight operations, and we also have been starting to use *DSENDS* for the simulation engine to generate image inputs from simulated trajectories to hardware-accelerated computer vision systems, like the Lander Vision System on Mars 2020. Both EDL and Small Body Proximity Operations proposals are currently being developed that will leverage this technology as well. Program offices often ask for concept development in the areas of EDL and Proximity Operations, and *DSENDS* provides a capable environment to support technology research and development and mission proposal formulation. *DSENDS* is an indispensable tool that can live at all fidelity levels, making it well suited for use in all phases of the project life cycle.

Looking to the future, we plan to include *DSENDS* engine as a core analytical tool leveraging the Python development environment and the Python scientific computing community to provide new ways to answer questions for our projects and programmatic customers. Incorporating tools developed at our national laboratories like DAKOTA and in the open source community like the SciPy[34] stack, Jupyter[35] notebooks, and HDF5 data storage are all areas of active development where we are using *DSENDS* to streamline analysis and improve analysis products for our current and future customers.

VI. Conclusion

In this paper, we have described key underlying components of the *DSENDS* simulation framework and have shown how each component contributes to making *DSENDS* a powerful and flexible simulation and analysis tool. The modeling functionality that *DSENDS* supports is suitable for designing and implementing space flight involving complex systems at various levels of fidelity as illustrated by the variety of applications of *DSENDS* to NASA missions and flight experiments described in the paper.

Although this paper has focused on *DSENDS* for aerospace applications, *DSENDS* is part of a family of simulation tools based on the *Dshell* simulation framework. Other tools in the family are being used for ground vehicle simulations[36] and protein folding.[37] These tools are all based on the same infrastructure and are interoperable.

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors thank the *COMPASS* team of the Flight Operations Directorate at Johnson Space Center for their support and collaboration.

References

[1]Balaram, J., Austin, R., Banerjee, P., Bentley, T., Henriquez, D., Martin, B., McMahon, E., and Sohl, G., "DSENDS - A High-Fidelity Dynamics and Spacecraft Simulator for Entry, Descent and Surface Landing," *IEEE 2002 Aerospace Conf.*, Big Sky, Montana, mar 2002.

[2]Wilcox, B., San Martin, M., Giersch, L., Howe, S., Grip, H. F., Nayeri, R., Litwin, T., Carlson, J., Shekels, M., Jain, A., Lim, C., Myint, S., Dunkle, J., Sirota, A., and Fuller, C., "Testbed for Studying the Capture of a Small, Free-flying Asteroid in Space," *AIAA Space Conference*, Pasadena, CA, 2015.

[3]Grip, H., Ono, M., Balaram, J., J., C., Jain, A., Kuo, C., Myint, S., and Quadrelli, M., "Modeling and Simulation of Asteroid Retrieval Using a Flexible Capture Mechanism," *Proc. 2014 IEEE Aerospace Conference*, Big Sky, Montanna, 2014.

[4]Lim, C. and Jain, A., "Dshell++: A Component Based, Reusable Space System Simulation Framework," *Third International Conference on Space Mission Challenges for Information Technology (SMC-IT 2009)*, Pasadena, CA, jul 2009.

[5]Jain, A., "Structure Based Modeling and Computational Architecture for Robotic Systems," *2013 IEEE International Conference on Robotics and Automation*, 2013.

[6]Cameron, J. M., Myint, S., Kuo, C., Jain, A., Grip, H. F., Jayakumar, P., and Overholt, J., "Real-Time and High-Fidelity Simulation Environment for Autonomous Ground Vehicle Dynamics," *2013 NDIA Ground Vehicle Systems Engineering and Technology Symposium*, Troy, Michigan, 2013.

[7]"http://dartslab.jpl.nasa.gov," .

[8]Jain, A., *Robot and Multibody Dynamics: Analysis and Algorithms*, Springer, 2011.

[9]Jain, A., "Multibody graph transformations and analysis Part II: Closed-chain constraint embedding," *Nonlinear Dynamics*, Vol. 67, No. 3, aug 2012, pp. 2153–2170.

[10]Jain, A., Cameron, J., Lim, C., and J., G., "SimScape Terrain Modeling Toolkit," *Second International Conference on Space Mission Challenges for Information Technology (SMC-IT 2006)*, July 2006.

[11]"https://www.hdfgroup.org/HDF5," .

[12]"http://www.ogre3d.org," .

[13]"http://bulletphysics.org," .

[14]Myint, S., Jain, A., Cameron, J. M., and Lim, C., "Large Terrain Modeling and Visualization for Planets," *Fourth IEEE International Conference on Space Mission Challenges for Information Technology*, San Francisco, CA, 2011.

[15]"http://www.swig.org," .

[16]Bowes, A. and et al., "LDSD POST2 Simulation and SFDT-1 Pre-Flight Launch Operations Analyses," *25th AAS/AIAA Space Flight Mechanics Meeting (AAS 15-232)*, Williamsburg, VA, January 11-15 2015.

[17]"For more information about the COMPASS program, see https://re.grc.nasa.gov/compass/ and https://www-robotics.jpl.nasa.gov/tasks/showTask.cfm?FuseAction=ShowTask&TaskID=209&tdaID=700018," .

[18]Way, D., "Preliminary Assessment of the Mars Science Laboratory Entry, Descent, and Landing Simulation," *Proceedings 2013 IEEE Aerospace Conference*, March 2-6 2013.

[19]Striepe, S., Way, D., Dwyer, A., and Balaram, J., "Mars Science Laboratory Simulations for Entry, Descent, and Landing," *Journal of Spacecraft and Rockets*, Vol. 43, No. 2, March-April 2006.

[20]Burkhart, P. D., Casoliva, J., and Balaram, B., "Mars Science Laboratory Entry Descent and Landing Simulation Using DSENDS," *Journal of the American Astronomical Society (AAS 13-421)*, 2013.

[21]Burkhart, P. D. and Casoliva, J., "MSL DSENDS EDL Analysis and Operations," *Proceedings 2012 International Symposium for Space Flight Dynamics*, 2012.

[22]Pomerantz, M., Lim, C., Myint, S., Woodward, G., and Kuo, C., "Multi-Mission Simulation and Visualization for Real-time Telemetry Display, Playback and EDL Event Reconstruction," *2012 AIAA Space*, Pasadena, CA, 2012.

[23]Bonfiglio, E., Adams, D., Craig, L., Spencer, D., Arvidson, R., and Heet, T., "Landing-Site Dispersion Analysis and Statistical Assessment for the Mars Phoenix Lander," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2012, pp. 784–797.

[24]Prince, J., Desai, P., Queen, E., and Grover, M., "Mars Phoenix Entry, Descent, and Landing Simulation Design and Modeling Analysis," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2011, pp. 756–764.

[25]Prince, J., Desai, P., Queen, E., and Grover, M., "Entry, Descent, and Landing Operations Analysis for the Mars Phoenix Lander," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2011, pp. 778–783.

[26]Queen, E., Prince, J., and Desai, P., "Mars Phoenix Entry, Descent, and Landing," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2011, pp. 765–771.

[27]Desai, P., Prince, J., Queen, E., Schoenenberger, M., Cruz, J., and Grover, M., "Entry, Descent, and Landing Performance of the Mars Phoenix Lander," *Journal of Spacecraft and Rockets*, Vol. 48, No. 5, 2011, pp. 798–808.

[28]"http://monte.jpl.nasa.gov," .

[29]Evans, S., Taber, W., Drain, W., Smith, J., Wu, H.-C., Guevara, M., Sunseri, R., and Evans, J., "MONTE: The Next Generation of Mission Design & Navigation Software," *Proceedings of the 6th International Conference on Astrodynamics Tools and Techniques (ICATT)*, Proceedings Darmstadt, Germany, 2016.

[30]Sunseri, R., Wu, H.-C., Evans, S., Evans, J., Drain, T., , and Guevara, M., "Mission Analysis, Operations, and Navigation Toolkit Environment (MONTE) Version 040," *NASA Tech Briefs*, Vol. 36, No. 9, 2012.

[31]Ivanov, M., "Low Density Supersonic Decelerator Flight Dynamics Test - 1 Flight Design and Targeting," *Proceedings of the 23rd AIAA Aerodynamic Decelerator Systems Technology Conference*, Daytona Beach, FL, April 2015.

[32]Grip, H. F., San Martin, M., Jain, A., Balaram, J., Cameron, J. M., and Myint, S., "Modeling and Simulation of Asteroid Capture using a Deformable Membrane Capture Device," *Proceedings of the ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Boston, MA, 2015.

[33]"https://dakota.sandia.gov," .

[34]"https://www.scipy.org," .

[35]"http://jupyter.org," .

[36]Cameron, J., Myint, S., Kuo, C., Jain, A., Grip, H., Paramsothy, J., and Overholt, J., "Real-Time and High-Fidelity Simulation Environment for Autonomous Ground Vehicle Dynamics," *2013 NDIA Ground Vehicle Systems Engineering and Technology Symposium*, Troy, Michigan, Aug 21-22 2013.

[37]Larsen, A. B., Wagner, J. R., Kandel, S., Salomon-Ferrer, R., Vaidehi, N., and Jain, A., "GneimoSim: A modular internal coordinates molecular dynamics simulation package," *Journal of Computational Chemistry*, Vol. 35, No. 31, 2014, pp. 2245–2255.