SAND2021-15380C

Linear Covariance Analysis Framework for Aerospace Vehicle Trajectory Modeling and Parametric Design

Lucas B. Heflin*, Nicholas J. Zuiker †, Grace E. Calkins ‡ and Zachary R. Putnam §
University of Illinois at Urbana-Champaign, Urbana, IL, 61801

Daniel Whitten ¶
Sandia National Laboratories, Albuquerque, NM, 87123

This paper presents the Symbolic Linear Covariance Analysis Tool (SLiC), a Python framework capable of simplifying the construction, verification, and analysis of aerospace systems using linear covariance analysis techniques. The framework leverages open-source libraries to enable symbolic manipulation and object-oriented abstraction to remove many of the barriers to linear covariance analysis when compared to other methods. The benefits of linear covariance analysis with Monte Carlo verification are addressed and the framework design is described. The framework is validated against existing literature results and demonstrated for a sample aerospace use case of a hypersonic entry system.

I. Nomenclature

$\mathbf{c}$	=	continuous measurement function
C	=	augmented state covariance matrix
C_p	=	augmented state and performance metrics covariance matrix
$\mathbf{f}$	=	true state propagation function
$\hat{\mathbf{f}}$	=	navigation state propagation function
F_k	=	partial derivative of function $\mathbf{f}$ with respect to vector $\mathbf{k}$
$\mathbf{g}$	=	controller input function
$\mathbf{h}$	=	discrete measurement function
n	=	dimension of state vector
n_i	=	dimension of vector $\mathbf{i}$
$\mathbf{p}$	=	performance metrics vector
$\mathbf{s}$	=	performance metrics function
t	=	time, s
$\hat{\mathbf{u}}$	=	commanded control input vector
$\mathbf{w}$	=	process noise vector
$\mathbf{x}$	=	state vector
$\hat{\mathbf{x}}$	=	navigation state vector
$\mathbf{X}$	=	augmented state vector
$\tilde{\mathbf{y}}$	=	measurement vector
η	=	measurement noise vector
Σ_i	=	covariance matrix of vector $\mathbf{i}$
σ	=	standard deviation
ξ	=	discrete measurement noise vector

*Master's Student, Aerospace Engineering, AIAA Student Member
†Doctoral Candidate, Aerospace Engineering, AIAA Student Member
‡Master's Student, Aerospace Engineering
§Assistant Professor, Aerospace Engineering, AIAA Senior Member
¶Senior Member of the Technical Staff

II. Introduction

LINEAR covariance analysis (LCA) is a method of uncertainty analysis in which the covariance matrix of a system is propagated along a nominal trajectory. Uncertainty analysis of nonlinear systems is often performed with Monte Carlo techniques where typically thousands of simulations are run to produce sufficient data to estimate statistical measures of system dispersions. In LCA, pertubations are linearized about a nominal, nonlinear trajectory to produce the same statistical information as a Monte Carlo analysis in a single integration. [1] This is achieved by integrating the state dispersion statistics directly alongside the nominal trajectory.

As a result of the relative computational efficiency of linear covariance techniques, LCA has been applied to preliminary mission design and trade studies where Monte Carlo analysis may be computationally prohibitive. [2] For Mars entry, descent, and landing (EDL) systems, LCA has been used to conduct large trade studies of closed-loop guidance, navigation, and control (GNC) flight systems to test how different vehicle configurations, nominal mission profiles, sensor suites, control schemes and other properties of the vehicle and mission design impact flight performance uncertainty. [3, 4] LCA has been successfully applied to spacecraft rendezvous mission applications, vehicle ascent and descent, as well as atmospheric entry. [5–8] More recently, linear covariance analysis has been applied to real-time systems for obstacle avoidance and guidance. [9, 10]

For students and researchers without access to proprietary or restricted software packages, programming LCA tools can lead to rigid and brittle solutions. Much of this is due to the tight coupling between the equations of motion of the system and the corresponding Jacobians required for the LCA. In implementations where the Jacobians are input by the user, any adjustment of the system dynamics or the addition of a new system state cascades into adjustments elsewhere, such as computation of the gradients, parameter passing, and vector dimensions. As a result, solutions are often limited to a specific system or mission architecture. [11]

The Symbolic Linear Covariance analysis tool (SLiC) was created to reduce the development time required for LCA and allow for tandem verification with Monte Carlo analysis without any modification of user inputs. SLiC has been designed with the open-source Python library SymPy to manage symbolic equation manipulation, partial derivative calculation, and execution of the equations. This paper describes the underlying linear covariance model that makes the theoretical backbone of the framework, provides an overview of the framework itself, and finally reviews sample use cases derived from literature and entry systems. The paper is concluded with a discussion on planned tool improvements and future work.

III. Background

The theoretical formulation of LCA used in SLiC is based on the available literature. This specific derivation follows reference [12] and aspects relevant to SLiC are reproduced below.

Consider a nonlinear system with dynamics described by the nonlinear vector differential equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \hat{\mathbf{u}}, t) + \mathbf{w} \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the true state, $\hat{\mathbf{u}} \in \mathbb{R}^{n_{\hat{u}}}$ is the commanded controller input, and t is time. The vector $\mathbf{w} \in \mathbb{R}^n$ is assumed to be a zero-mean random white process noise vector with covariance matrix Σ_w. In addition to the true state, there is the navigation state $\hat{\mathbf{x}} \in \mathbb{R}^{n_{\hat{x}}}$ with dynamics described by the nonlinear equation

$$\dot{\hat{\mathbf{x}}} = \hat{\mathbf{f}}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \tilde{\mathbf{y}}, t) \tag{2}$$

where $\tilde{\mathbf{y}} \in \mathbb{R}^{n_{\tilde{y}}}$ is the continuous measurement vector. The commanded controller input $\hat{\mathbf{u}}$ is a function of the navigation state following

$$\hat{\mathbf{u}} = \hat{\mathbf{g}}(\hat{\mathbf{x}}, t) \tag{3}$$

and the continuous measurement vector is a function of the true state, commanded input, and measurement noise following

$$\tilde{\mathbf{y}} = \mathbf{c}(\mathbf{x}, \hat{\mathbf{u}}, t) + \eta \tag{4}$$

where $\eta \in \mathbb{R}^{n_{\tilde{y}}}$ is the zero-mean white measurement noise vector with covariance Σ_η. The navigation state is assumed to be updated using an extended Kalman filter and discrete sensor measurements are also modeled. For a specific time-step k, the discrete sensor vector $\mathbf{z}_k$ is modeled as a function of the true state following

$$\tilde{\mathbf{z}}_k = \mathbf{h}(\mathbf{x}_k, t_k) + \xi_k \tag{5}$$

where ξ_k is the zero-mean white measurement noise vector with covariance Σ_ξ. These discrete sensor readings are modeled in the navigation system following

$$\hat{\tilde{\mathbf{z}}}_k = \hat{\mathbf{h}}(\hat{\mathbf{x}}_k, t_k) \tag{6}$$

where $\hat{\tilde{\mathbf{z}}}_k$ is the estimated discrete sensor measurement vector at time-step k. The propagation, guidance, and measurement equations above are linearized about a nominal state $\bar{\mathbf{x}}$ to produce

$$\delta\dot{\mathbf{x}} = F_x\delta\mathbf{x} + F_{\hat{\mathbf{u}}}\hat{G}_{\hat{\mathbf{x}}}\delta\hat{\mathbf{x}} + \mathbf{w} \tag{7}$$

$$\delta\dot{\hat{\mathbf{x}}} = (F_{\hat{\mathbf{x}}} + F_{\hat{\mathbf{u}}}\hat{G}_{\hat{\mathbf{x}}})\delta\hat{\mathbf{x}} + F_{\hat{\mathbf{u}}}C_x\delta\mathbf{x} + \hat{F}_{\tilde{y}}\eta \tag{8}$$

where $\delta\mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}$ and $\delta\hat{\mathbf{x}} = \hat{\mathbf{x}} - \bar{\mathbf{x}}$ are the dispersions of the true state and navigation state from the nominal state, respectively. Uppercase letters denote the partial derivatives of their lowercase functions with respect to the variable indicated in the subscript and evaluated at the nominal state. (e.g. $F_x = \partial\mathbf{f}/\partial\mathbf{x}\|_{\bar{\mathbf{x}}}$) Combining the true and navigation state dispersions, we define the augmented state vector $\mathbf{X}$ as

$$\mathbf{X} = \begin{bmatrix} \delta\mathbf{x} \\ \delta\hat{\mathbf{x}} \end{bmatrix} \tag{9}$$

with dynamics described by

$$\dot{\mathbf{X}} = \mathcal{F}\mathbf{X} + \mathcal{G}\eta + \mathcal{W}\mathbf{w} \tag{10}$$

where

$$\mathcal{F} = \begin{bmatrix} F_x & F_{\hat{\mathbf{u}}}\hat{G}_{\hat{\mathbf{x}}} \\ F_{\hat{\mathbf{u}}}C_x & F_{\hat{\mathbf{x}}} + F_{\hat{\mathbf{u}}}\hat{G}_{\hat{\mathbf{x}}} \end{bmatrix} \qquad \mathcal{G} = \begin{bmatrix} O_{n\times n_{\mathbf{y}}} \\ \hat{F}_{\mathbf{y}} \end{bmatrix} \qquad \mathcal{W} = \begin{bmatrix} I_{n\times n} \\ O_{\hat{n}\times n} \end{bmatrix} \tag{11}$$

Discrete updates are applied to the augmented state following:

$$X_k^+ = \mathcal{A}_k X_k^- + \mathcal{D}_k\xi_k \tag{12}$$

where

$$\mathcal{A}_k = \begin{bmatrix} I_{n\times n} & 0_{n\times\hat{n}} \\ \hat{K}_k H_k & I_{\hat{n}\times\hat{n}} - \hat{K}_k\hat{H}_k \end{bmatrix} \qquad \mathcal{D}_k = \begin{bmatrix} 0_{n\times n_z} \\ \hat{K}_k \end{bmatrix} \tag{13}$$

It can be shown that the covariance matrix C of the augmented state has the propagation equation [1]

$$\dot{C} = \mathcal{F}C + C\mathcal{F}^\intercal + \mathcal{G}\Sigma_\eta\mathcal{G}^\intercal + \mathcal{W}\Sigma_w\mathcal{W}^\intercal \tag{14}$$

with discrete updates in the form of

$$C(t_k^+) = \mathcal{A}_k C(t_k^-)\mathcal{A}_k^T + \mathcal{D}_k\Sigma_\xi\mathcal{D}_k^T \tag{15}$$

It is often useful to extract covariance information for additional "performance metrics," such as extracting the altitude covariance from the position vector. [3] These performance metrics, $\mathbf{p}$, are typically functions of the true and navigation states following

$$\mathbf{p} = \mathbf{s}(\mathbf{x}, \hat{\mathbf{x}}) \tag{16}$$

Upon completion of a LCA, the covariance of these performance metrics with respect to the augmented state can be obtained via

$$C_p = SCS^\intercal \tag{17}$$

where

$$S = \text{diag}(I_n, I_{\hat{n}}, S_{\mathbf{x}}, S_{\hat{\mathbf{x}}}) \tag{18}$$

and S_x and $S_{\hat{x}}$ represent the partial derivatives of $\mathbf{s}$ with respect to the true and navigation states, respectively. These equations, along with initial conditions for $\mathbf{x}$, $\hat{\mathbf{x}}$, and C, define the dynamics necessary for LCA.

IV. Framework Description

SLiC is a Python framework that simplifies the setup of LCA and includes Monte Carlo verification. SLiC takes a single Python object as input, called a `Scenario`, and outputs the estimated system covariance about a computed nominal trajectory. Statistical distributions and trajectory data are also included for the Monte Carlo verification. These data can be saved as binary data files and visualized in illustrative plots.

To achieve the goals of simplicity, accuracy, and flexibility, SLiC leverages the open-source Python library SymPy. SymPy is a Python library for symbolic mathematics that includes support for symbolic expression manipulation and simplification, calculus, and linear algebra among others. [13] SymPy has been used in various open-source projects in the fields of mathematics, chemistry, electrical engineering, and physics. The reasons to use a symbolic implementation with SymPy are threefold: implementation flexibility, system transparency, and performance.

Using a symbolic implementation provides analysis flexibility for LCA. By allowing the symbolic differentiation of the system equations of motion, system uncertainty can be injected ad-hoc without any modification of the underlying dynamics or output partial derivatives. User-defined scenarios can be adjusted quickly as analysis needs change. New states can be added, new dynamics implemented, and new measurements taken without any modification of the partial derivatives by the user.

System transparency is vital to ensure the user can validate the tool and apply proper engineering judgment. While SLiC removes tedious computations of partial derivatives, it preserves system transparency by outputting computed derivatives in a human-readable format. This is especially important for verifying complex dependencies and nested functions that chain together.

Differentiating the system dynamics analytically provides the most accurate partial derivatives possible. When compared to numeric methods, symbolic methods have a higher overhead time cost to compute the partial derivatives, but can save time over numeric methods by removing the additional function calls needed at each integration step to compute derivatives numerically with finite-difference equations. For functions with relatively simple derivatives, this function call time savings can be on the order of 2-4 times faster than second and fourth-order accurate finite difference schemes. Moreover, these analytic functions can be compiled and saved for future use to provide additional time savings.

A. Framework Structure

SLiC is designed using core object oriented programming principles of abstraction and encapsulation. Figure 1 shows a high-level class diagram of the modules of SLiC. Each colored region represents a different module with multiple implemented classes available to the user. These modules include `Surrogates`, `Stochastic Models`, `Device Models`, `Environment Models`, and `Scenario`. Of these, SLiC has been designed such that a user only needs to interact and design their own `Scenario` class which serves as a single object to be input into the `Monte Carlo Analyst` and `Linear Covariance Analyst` objects. The four other modules feed into `Scenario` by offering plug-and-play utilities for aerospace systems.

The `Surrogates` module is capable of encapsulating data-driven or highly nonlinear functions not suitable for symbolic differentiation. A `Surrogate` object wraps user-supplied functions or data in a symbolic function definition that integrates seamlessly in the SymPy-based framework. Examples include aerodynamic data, device saturation behaviors, and gain scheduling. Surrogates are divided into three types: functional, data-driven, and numeric. Functional surrogates are the most generic objects and take any non-symbolic, user-defined function with an associated non-symbolic, user-defined derivative function and wraps them to interface with the SymPy API. Data-driven surrogates linearly interpolate an input data array of arbitrary dimension, returning the value of the slope along each dimension as the partial derivative. Numeric surrogates take an input user-defined function and perform a fourth-order accurate numeric differentiation for the Jacobian. With the `Surrogates` module, SLiC extends the SymPy architecture through the chain and product differentiation rules to ensure mathematical accuracy for all possible user inputs.

The `Stochastic Models` module encapsulates stochastic processes. Such processes must be handled separately from continuous dynamics as a function of the numeric integration time step used. SLiC includes Gaussian random

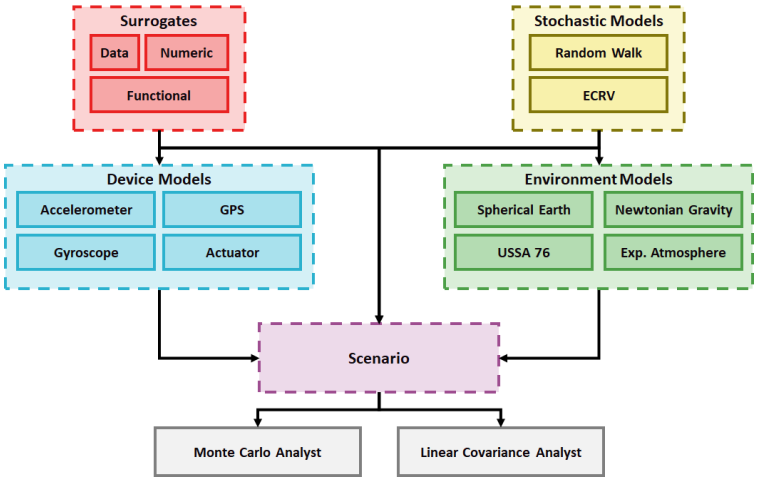


Fig. 1 Simplified, high-level structure of SLiC.

walk processes and exponentially correlated random variables (ECRVs), two of the most common uncertainty models used in aerospace. The ECRV uncertainty can be treated as either constant, time-dependent, or state-dependent. The state-dependency is particularly useful in atmospheric models, for example, where atmospheric uncertainty may vary with altitude. [14] The user provides a piecewise-continuous scaling function with inputs of augmented state variables that outputs a state-scaled standard deviation value to be used in the ECRV propagation. The elements of the process and sensor noise matrices, $\Sigma_{\mathcal{W}}$ and Σ_{η}, are updated according to the provided scaling function prior to integration of the covariance matrix each time step. As a benefit of the object-oriented design, these classes adhere to the same interface and can be quickly exchanged by a user.

The `Device Models` module encapsulates symbolic models of common devices such as actuators and sensors. This includes common devices like accelerometers, gyroscopes, and GPS-like systems. Many device models share common uncertainty models like scaling and biasing uncertainties. These can be combined using the SLiC object-oriented design into larger models, such as a three-axis inertial measurement unit being composed of component accelerometers and gyroscopes.

The `Environment Models` module encapsulates various environmental models such as atmospheric, gravity, and surface models relevant for simulations. These include analytic models, such as the hydrostatic equation, and numeric or data-driven models like atmospheric temperature profiles.

Finally, the `Scenario` module is the class in which the user will define their aerospace system to analyze. Leveraging the other modules for relevant aerospace models and numeric wrappers, users define their state vectors, measurement and control vectors, and corresponding propagation equations symbolically. SymPy, then, is used to generate the system Jacobians necessary for LCA and compiles the code to be integrated numerically. This removes the onus on the user to simultaneously define the system and its Jacobians that made previous programs rigid and prone to error. As a result, SLiC is simple, flexible, and accurate.

V. Illustrative Examples

A. Verification Against Literature-Derived Rocket Sled Scenario

To verify the outputs generated in the LCA, results from SLiC were compared against the literature for a simple 1-dimensional rocket sled example. [1]

1. Scenario Description

The rocket sled is a rigid body that travels perpedicularly to gravity, powered by an "acceleration actuator" that supplies Δv. The sensor suite includes an accelerometer and Doppler velocimeter. The state dynamics can be divided

into the true state, sensor instrument stochastic processes, and actuator stochastic processes. The true state is:

$$\mathbf{x} = \begin{bmatrix} v \\ d \\ b \\ s \\ e \\ z \end{bmatrix} \tag{19}$$

where v is the vehicle velocity, d is the disturbance acceleration on the vehicle, b is the accelerometer bias, s is the accelerometer scale factor, e is the actuator bias, and z is the Δv used by the system. All biases and scale factors are modeled as ECRVs with time constants τ_i and uncertainty standard deviations σ_i, where i is the respective variable. The state dynamics are as follows:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \hat{\mathbf{u}}, t) + \mathbf{w} = \begin{bmatrix} d + \hat{a}_{com} + e - c_d v^2 \\ -\frac{d}{\tau_d} + w_d \\ -\frac{b}{\tau_b} + w_b \\ -\frac{s}{\tau_s} + w_s \\ -\frac{e}{\tau_e} + w_e \\ |a_{act}| \end{bmatrix} \tag{20}$$

where $\hat{a}_{com}$ is the commanded thrust acceleration, c_d is the drag coefficient, and a_{act} is the net actuator input on the vehicle, $\hat{a}_{com} + e$. The Doppler velocimeter is modeled as a discrete sensor with noise that updates every 10 seconds. The Doppler velocity measurement at time-step k is given by:

$$\tilde{v}_k = v_k + \xi_k \tag{21}$$

where ξ_k is the Doppler measurement noise, modeled as a zero-mean Gaussian random process. The accelerometer is modeled as a continuous sensor with the measurement equation:

$$\tilde{a} = (1 + s)(d + \hat{a}_{com} + e - c_d v^2 + b + \eta_a) \tag{22}$$

where η_a is the accelerometer measurement noise modeled as a zero-mean Gaussian random process. The navigation state vector is:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{v} \\ \hat{b} \\ \hat{s} \end{bmatrix} \tag{23}$$

From the design model derived in [1], the navigation state dynamics are modeled as:

$$\dot{\hat{\mathbf{x}}} = \begin{bmatrix} \frac{\tilde{a}}{1-s} - \hat{b} \\ -\frac{\hat{b}}{\tau_{\hat{b}}} \\ -\frac{\hat{s}}{\tau_{\hat{s}}} \end{bmatrix} \tag{24}$$

The goal of the closed-loop control system on board the vehicle is to achieve a desired final velocity at the specified final time. Three guidance and control laws are considered. The first is a time-to-go guidance law in Eq. (25) and the second is identical to the first with the addition of a drag compensation term in Eq. (26). For both of these options, the state is estimated along the reference trajectory using an extended Kalman filter (EKF).

$$\hat{a}_{com} = \frac{v_t - \hat{v}}{k + (t_f - t)} \tag{25}$$

$$\hat{a}_{com} = \frac{v_t - \hat{v}}{k + (t_f - t)} + c_d \hat{v}^2 \tag{26}$$

The third guidance and control law utilizes the same control law as Eq. (25), however, utilizes a dead-reckoning scheme for navigation.

2. SLiC Implementation

The SLiC implementation for the rocket sled centers on the `Scenario` definition. First, variables and parameters in the scenario are defined using the `SLCAVariable` and `SLCAParameter` objects which wrap symbolic manipulation capabilities and provide additional descriptive information. The ECRVs b, d, s, and e are defined with the built in `ECRV` class. Similarly, the noise processes w, η, and ξ are implemented as `GaussianProcess` objects.

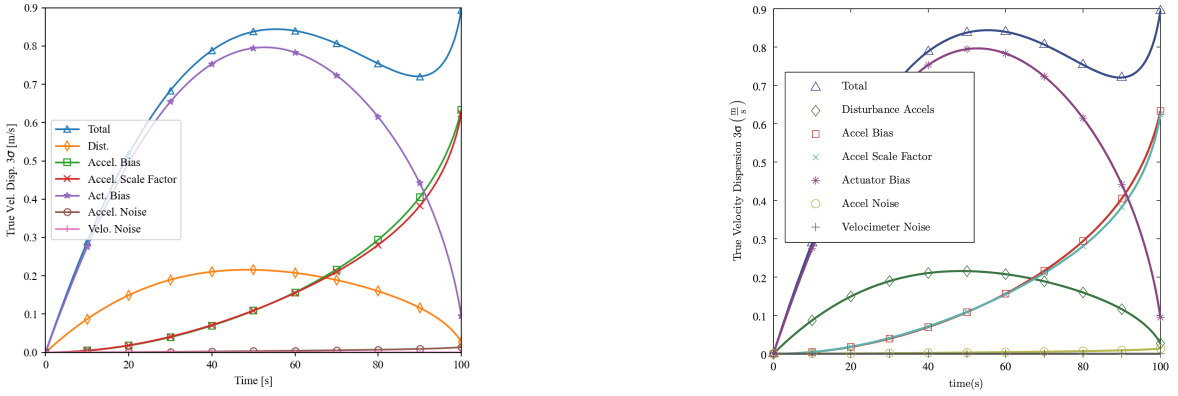
Next, the true state vector, true state dynamics, control command vector, and guidance function are defined within the scenario as symbolic, analytic equations following the equations listed in Section V.A.1 above. The rocket sled EKF navigation is contained within the scenario as a `NavComponent` object. This component defines the navigation state, navigation propagation equations, and handles discrete updates based upon the discrete measurements. The Doppler velocimeter is implemented as a `DiscreteSensor`, which contains the measurement equations along with noise properties, and added to the scenario navigation component. The accelerometer is defined within the scenario's continuous sensing function following Eq. (22).

Finally, scenario parameter configurations are defined in the `param_set` method. Multiple configurations can be input to permit GNC trade studies like the one shown in Section V.A.3. The scenario is now ready to be integrated for a given initial augmented state condition and initial augmented state uncertainty.

3. Analysis

The rocket sled system was analyzed with LCA for three different GNC schemes, three different sensor quality specifications, and eight sensor and process noise specifications for a total of forty-eight linear covariance analyses. Assessing the system via a 500-sample Monte Carlo analysis, as was done in [1], would require twenty-four thousand vehicle simulations to produce the same analysis. These results were qualitatively compared to the literature to verify SLiC's accuracy. [1]

The components of the true velocity dispersion can be found by varying a single true error source per run while setting all other error sources to zero. These results show the user how each noise source contributes to the overall uncertainty of the system over time. The results of performing this analysis in SLiC with the dead reckoning navigation scheme are presented in Fig. 2 and compared to the same analysis from [1]. These results show that the actuator bias is the most significant contribution to overall velocity uncertainty. As expected, without a Kalman filter, the contributions of accelerometer noise uncertainty grow with time. The contribution from the accelerometer noise is very small and the contribution from the velocimeter is zero.



(a) SLiC Results **(b) Literature Results [1]**

Fig. 2 Velocity state dispersion components from each noise source for dead reckoning GNC scheme.

When implementing an EKF navigation filter, the terminal true velocity dispersion decreases by approximately 84%, as seen in Fig. 3. The EKF does not reduce the effects of actuator bias as it does not estimate this in the filter. Comparing the results here to Fig. 2a, we see that in the initial 20 seconds, dead reckoning is comparable to the EKF,

but as the accelerometer uncertainties grow with time, the dead reckoning method becomes less accurate.

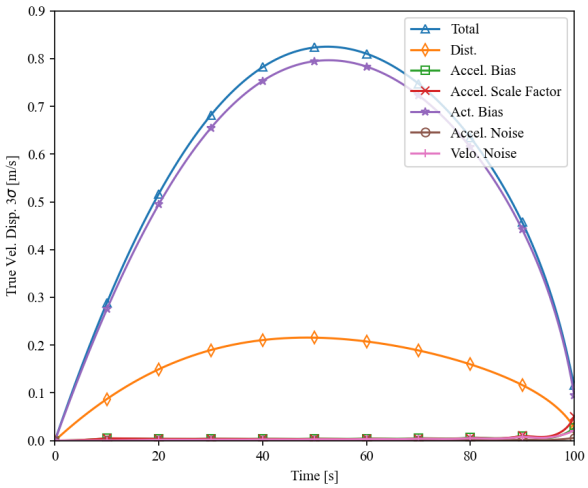


Fig. 3 SLiC velocity state dispersion components from each noise source for EKF navigation filter.

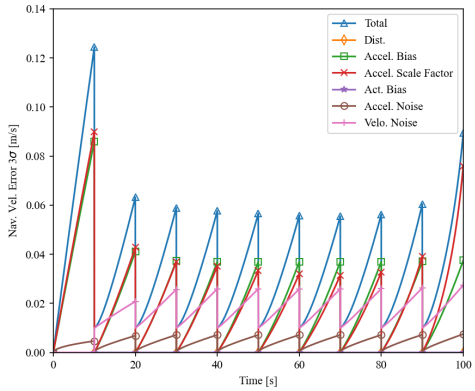
The user can also compare sensor quality by varying the standard deviations of the stochastic processes implemented. High quality, expensive sensors have lower measurement uncertainties while lower quality, less expensive sensors generally have larger measurement uncertainties. Three cases, a nominal case, a high quality case, and a low quality case, were considered in SLiC using EKF for navigation and the time-to-go guidance scheme without drag compensation. Each change in sensor quality is treated as an order of magnitude change in the sensor uncertainty magnitude. The specifications for the nominal and high quality sensor suites, for example, are shown in Table 1. The three sensors suites are compared in Fig. 4a with the value of the navigation velocity error over time. SLiC agrees very well with the nominal quality results shown in Fig. 4b. As expected, every 10 seconds the velocity error decreases as a new velocimeter measurement is received. Fig. 4a shows that the high quality sensor reduced navigation error by approximately 76% from the nominal case.

Table 1 Sensor and actuator specifications for nominal and high quality sensors

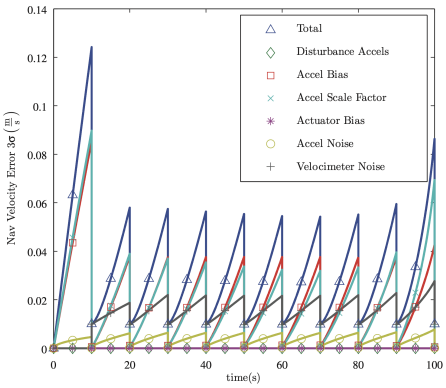
Variable	High Quality Value	Nominal Value	Units
Accelerometer Bias 3-σ, $\sigma_{b,ss}$	90	900	μg
Accelerometer Scale Factor 3-σ, $\sigma_{s,ss}$	90	900	ppm
Actuator Bias 3-σ, $\sigma_{e,ss}$	0.003	0.03	m/s^2
Accelerometer Measurement Noise 3-σ, σ_a	15	150	$\mu g\sqrt{s}$
Velocity Measurement Noise 3-σ, R_v	0.001	0.01	m/s

Additionally, the improvement on the error contribution with each sensor option is presented in Fig. 5. Again, the results from SLiC are congruent with the results presented in literature. Actuator bias is again the most significant error contribution as it is not estimated in the Kalman filter; however, improving sensor quality reduces the actuator bias by approximately 90%. Improving the sensor suite reduces the navigation dispersion for all of the noise contributions except the disturbance acceleration, which remains constant as it is not accounted for in the navigation filter.

As seen here, SLiC allows the user to easily implement different control schemes, stochastic models, and navigation filters. From these results, a designer could determine which GNC scheme to implement to satisfy mission requirements and compare a broad array of sensors to determine the best fit.

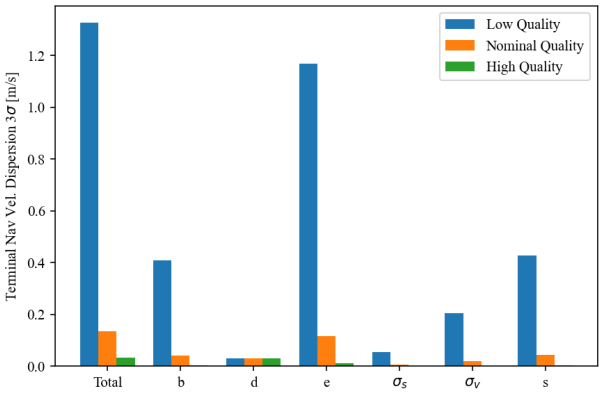


(a) SLiC Results

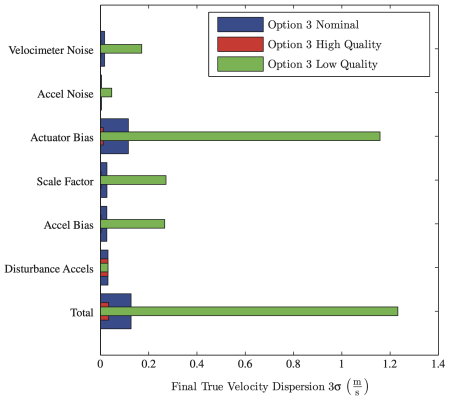


(b) Literature Results [1]

Fig. 4 Navigation velocity error with EKF navigation and time-to-go guidance and control scheme without drag compensation.



(a) SLiC Results



(b) Literature Results [1]

Fig. 5 Sensor quality impact on terminal true velocity dispersion for EKF navigation filter and time-to-go guidance and control scheme with drag compensation.

B. Entry System Uncertainty Analysis

1. Scenario Description

Consider the example of a sample return mission profile. The entry vehicle is modeled after a 60° sphere-cone and enters Earth's atmosphere at hypersonic speeds. The vehicle is modeled with the state

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \rho_e \\ \alpha_e \\ \beta_e \end{bmatrix} \tag{27}$$

where $\mathbf{r}$ is the position vector of the vehicle in the inertial frame, $\mathbf{v}$ is the velocity vector in the inertial frame, ρ_e is the atmospheric density disturbance, α_e is the angle of attack disturbance, and β_e is the bank angle disturbance. The three disturbance variables, ρ_e, α_e, and β_e, are modeled as ECRVs. The vehicle has dynamics described by

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \hat{\mathbf{u}}, t) + \mathbf{w} = \begin{bmatrix} \mathbf{v} \\ \mathbf{a}_L + \mathbf{a}_D + \mathbf{a}_g \\ -\frac{\rho_e}{\tau_\rho} + w_\rho \\ -\frac{\alpha_e}{\tau_\alpha} + w_\alpha \\ -\frac{\beta_e}{\tau_\beta} + w_\beta \end{bmatrix} \tag{28}$$

where $\mathbf{a}_L$ is the lift acceleration vector, $\mathbf{a}_D$ is the drag acceleration vector, and $\mathbf{a}_g$ is the acceleration due to gravity in the inertial frame. The parameters τ_ρ, τ_α, and τ_β are the time constants of their respective ECRVs and w_ρ, w_α, and w_β, are their driving zero-mean Gaussian white noise processes with standard deviations σ_ρ, σ_α, and σ_β, respectively. Attitude of the vehicle is determined based upon the instantaneous velocity vector of the vehicle and transforming into the wind-relative frame. It is assumed that the vehicle velocity does not enter the singularity when $\mathbf{v}$ is parallel to $\mathbf{r}$ where this transformation is undefined. The accelerations are functions of the state $\mathbf{x}$, input command $\hat{\mathbf{u}}$, and time t given by

$$\mathbf{a}_L = \left(qAC_L(M, \hat{\alpha} + \alpha_e)\right) R \begin{bmatrix} 0 \\ \sin(\hat{\beta} + \beta_e) \\ -\cos(\hat{\beta} + \beta_e) \end{bmatrix} \tag{29}$$

$$\mathbf{a}_D = \left(qAC_D(M, \hat{\alpha} + \alpha_e)\right) R \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \tag{30}$$

$$\mathbf{a}_g = -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} \tag{31}$$

where q is the dynamic pressure, A is the vehicle reference area, M is the Mach number, R is the rotation matrix from the wind frame to the inertial frame, and μ is the standard gravitational parameter. Variables $\hat{\alpha}$ and $\hat{\beta}$ are the commanded angle of attack and bank angles, respectively. Of note in Eqs. 29 and 30 is that both the lift and drag coefficients, C_L and C_D, are functions of both the Mach number and angle of attack. Parameters used for this scenario are provided in Table 2. State variables and their associated initial values and uncertainties are provided in Table 3.

Table 2 Parameter values for linear covariance evaluation

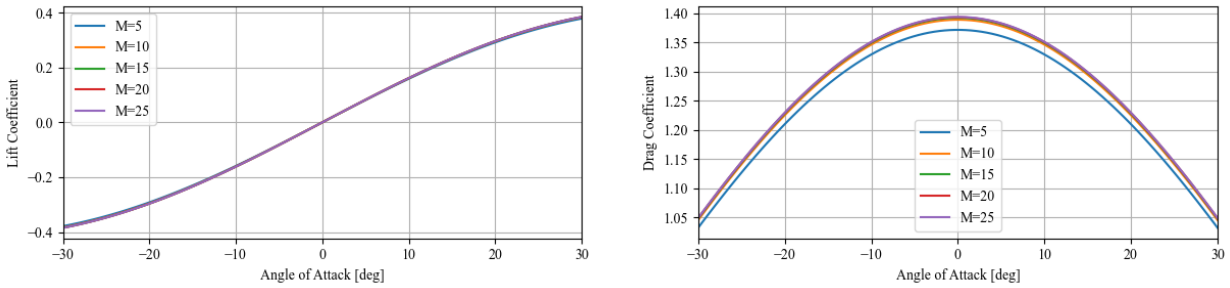
Parameter	Description	Value	Units
S_{ref}	aerodynamic surface area	0.503	m^2
m	vehicle mass	305.397	kg
σ_e	air density disturbance ECRV standard deviation	$0.1225/3$	kg/m^3
σ_α	angle of attack disturbance ECRV standard deviation	2/3	$^\circ$
σ_β	bank angle disturbance ECRV standard deviation	5/3	$^\circ$
τ_e	air density disturbance ECRV time constant	1	s
τ_α	angle of attack disturbance ECRV time constant	10	s
τ_β	bank angle disturbance ECRV time constant	10	s
h_r	reference altitude	0	m
h_s	scale altitude	8000	m
ρ_r	nominal sea-level air density	1.225	kg/m^3
μ	Earth standard gravitational parameter	3.986×10^{14}	m^3/s^2
γ	ratio of specific heat for air	1.4	-
R_{air}	specific gas constant for air	287	$\frac{J}{kg \times m}$
$r_\oplus$	Earth radius	6378×10^3	m

Table 3 Initial entry conditions for analysis

Variable	Description	Initial Value	Initial standard deviation	Unit
r_x	ECI x-position	6458.0	1.0	km
r_y	ECI y-position	0.0	1.0	km
r_z	ECI z-position	0.0	1.0	km
v_x	ECI x-velocity	-2083.778	50.0	m/s
v_y	ECI y-velocity	11817.693	50.0	m/s
v_z	ECI z-velocity	0.0	50.0	m/s
ρ_e	atmospheric density disturbance	0.0	0.1225/3	kg/m^3
α_e	angle of attack disturbance	0.0	1	$^\circ$
β_e	bank angle disturbance	0.0	5/3	$^\circ$

2. SLiC Implementation

As with the rocket sled example, the core of the user experience with SLiC lies in the definition of the user's `Scenario`. The symbolic equivalents for the state variables (r_x, r_y, etc) and system parameters (m, μ, etc) are defined. The ECRVs ρ_e, α_e, and β_e are implemented as built-in `ECRV` objects with included time constant and standard deviation properties. Lift and drag coefficients are wrapped as a data-driven `Surrogate` model using data from the CFD software package, CBAERO. [15] The data spanned Mach numbers between 5 to 25 and angles of attack from -30° to 30°. (Fig. 6) In this scenario, there is no navigation state and therefore the navigation components of the implementation are ignored. While closed loop feedback is possible in SLiC, developing a flight controller is beyond the scope of this work. Control inputs for the commanded angle of attack $\hat{\alpha}$ and bank angle $\hat{\beta}$ were modeled as open-loop periodic functions.



(a) **(b)**

Fig. 6 Aerodynamic (a) lift and (b) drag curves for the 60-deg sphere cone vehicle. Values derived from CBAERO.

3. Analysis

The system is integrated using a fourth-order Runge-Kutta integrator. Each variable in the augmented state vector and the performance state is plotted against both absolute axes and relative to the nominal trajectory to visualize dispersions. As expected, the linear covariance shows excellent agreement with the Monte Carlo results. (Fig. 7) Comparison of flight profile examined in Fig. 7 alongside maximum lift and no lift cases show SLiC capability of modifying control functions and comparing results. (Fig. 8)

As a measure of the system flexibility, different noise characteristics were explored. Specifically, the ECRV β_e was replaced ad-hoc with a random walk behavior by leveraging the modularity built into the framework. Both the ECRV and random walk modes show expected behavior with the ECRV staying bounded and the random walk growing over time. (Fig. 9)

Environment models implemented in SLiC are similarly designed to be simple to update to meet different levels of fidelity. Atmospheric density uncertainty, for example, can be easily modified in the scenario to account for

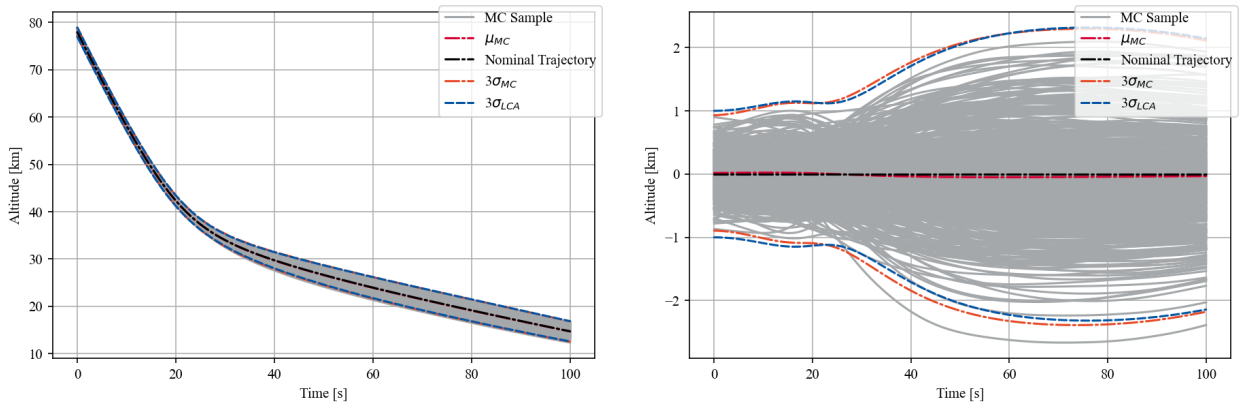


Fig. 7 Comparison between linear covariance and Monte Carlo distributions in altitude in (a) absolute coordinates and (b) relative to the nominal trajectory.

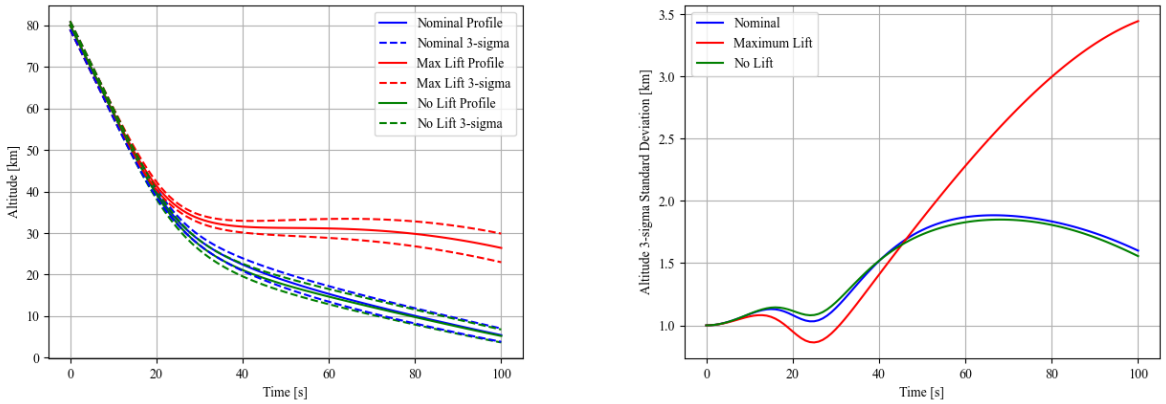


Fig. 8 Comparison of different commanded attitudes in (a) flight profiles with uncertainty in absolute coordinates and (b) relative to nominal trajectories.

state-dependent variations, such as altitude, using the state-dependent ECRV. This enables simple, rapid assessment of the effects of not only atmospheric properties, but also atmospheric models on the estimated system performance.

System outputs beyond the augmented state and performance metrics are also included in the output SLiC data, such as sensor measurements and control inputs. The open-loop angle of attack and drag coefficients of the Monte Carlo analysis are shown in Fig. 10. Measurements such as these can be particularly useful for engineers as they can easily record values before and after perturbations are applied. Such perturbations are clearly visible in the output of the angle of attack, $\alpha = \hat{\alpha} + \alpha_e$. (Fig. 10a) These perturbations are inputs in both the Monte Carlo and linear covariance analyses as part of the nominal case referenced in this section. Figure 10b shows the measured drag coefficient over time subject to the periodic angle of attack input and the variable Mach number throughout the trajectory using the data-driven `Surrogate` object. Note that the drag coefficient, which is symmetric in angle of attack, oscillates at twice the angle of attack frequency.

SLiC includes several visualization capabilities to simplify the visualization of the augmented state uncertainty ellipses. Such ellipses are embedded in the output LCA covariance matrix for both the augmented state and performance metrics. In this scenario, performance metrics for latitude and longitude were created to accurately transform the covariance in the position vector into spherical coordinates on the Earth. These uncertainty ellipses can be combined using other open-source Python modules to enhance graphics further. Figure 11 shows the uncertainty ellipses merged

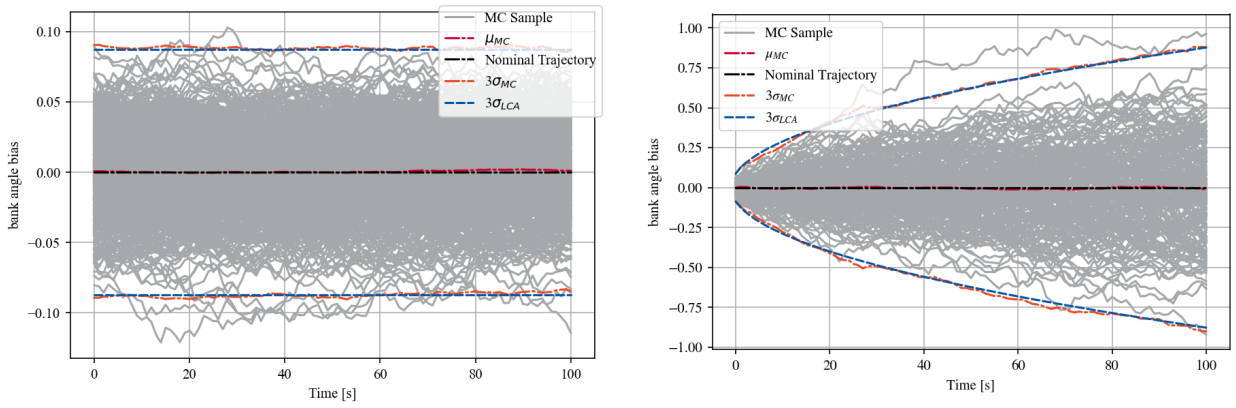


Fig. 9 Comparison between an (a) ECRV uncertainty and (b) random walk uncertainty. These were implemented in the SLiC `ECRV` and `RandomWalk` objects.

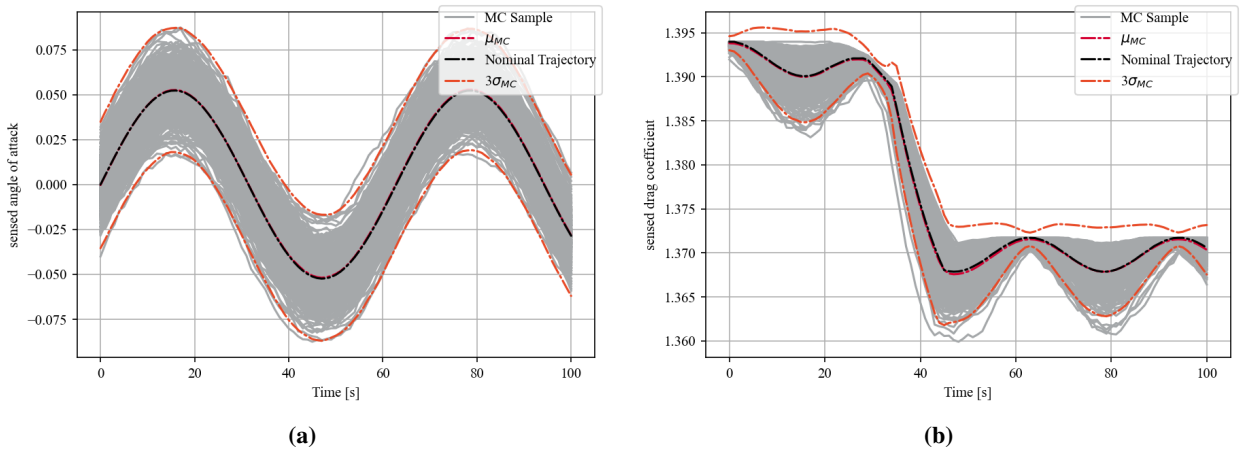


Fig. 10 Time-series plots of (a) vehicle angle of attack and (b) drag coefficient for 200 samples of the Monte Carlo simulation.

together using the Python shapely library to form a more intuitive envelope of the vehicle trajectory. Similarly, the uncertainty ellipses can be extended for use in 3D space by plotting the ellipsoids of the state uncertainty. (Fig. 12)

VI. Conclusion

In this paper, the novel symbolic LCA capability SLiC was described and demonstrated. SLiC overcomes some of the development challenges common in LCA by leveraging the open-source symbolic computer algebra library SymPy. With the symbolic framework, SLiC retains computational accuracy while achieving broad flexibility to meet a variety of system definitions. An engineer using SLiC can supply a single `Scenario` object as an input for both LCA and Monte Carlo which can be modified easily to adjust to mission needs. SLiC produces time series data and figures of the simulated trajectories, including the augmented state, performance metrics, controller inputs, sensor outputs, and the estimated covariance matrix of the system. SLiC was qualitatively validated against existing literature results and the implemented LCA was verified through the tandem Monte Carlo analysis capability. The utility of SLiC was further demonstrated against a nominal entry system application where control inputs, system parameters, and environmental uncertainty models were varied to assess the impact of these models on the system uncertainty and analysis accuracy.

SLiC continues to be actively developed and improved. Future work focuses on incorporating more aerospace-focused modules to simplify the examination of a broader range of systems and different levels of fidelity. These include support

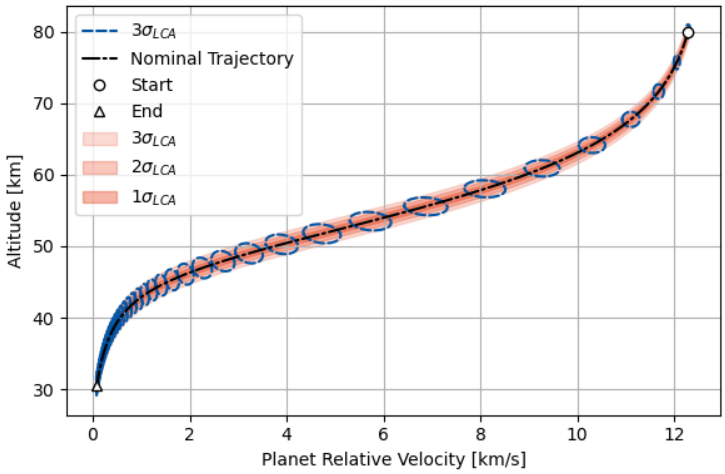


Fig. 11 Hypersonic entry trajectory visualizing the flight envelope of the vehicle based upon linear covariance analysis.

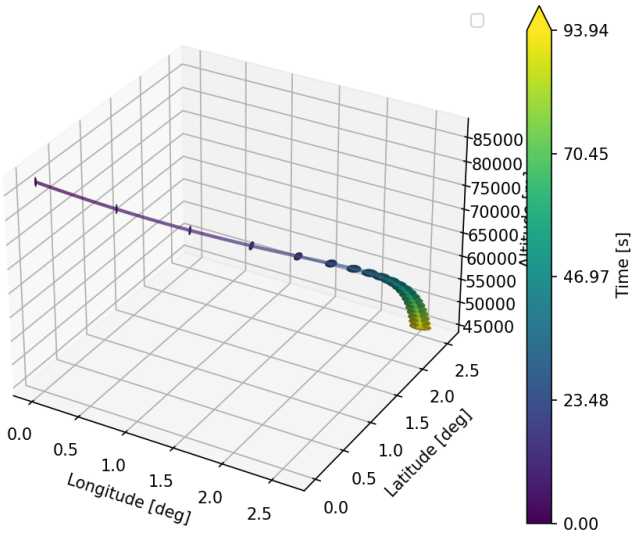


Fig. 12 Latitude/longitude/altitude representation of a hypersonic entry trajectory with 3D covariance ellipsoids derived from linear covariance analysis. Ellipsoids are colorized by time from dark blue (earliest) to bright yellow (latest).

for state-dependent event triggers and discrete true state updates, streamlined implementation and visualization of attitude representations such as quaternions, and more advanced environmental models to include models for other planets such as Mars. Performance improvements, such as paralellization and compiled libraries, are also planned as SLiC matures. Upon reaching a suitable level of development, SLiC will be made available to the community on common open-source platforms.

Acknowledgments

This work was supported by the Department of Education, through the Graduate Assistance in Areas of National Need Fellowship Program award P200A180050-19 at the University of Illinois Urbana-Champaign Department of Aerospace Engineering, and the Laboratory Directed Research and Development program at Sandia National Laboratories, a

multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] Christensen, R. S., and Geller, D., "Linear covariance techniques for closed-loop guidance navigation and control system analysis," *Journal of Aerospace Engineering*, Vol. 221, No. 1, 2012, pp. 44–65.

[2] Yun, S., Tuggle, K., Zanetti, R., and D'Souza, C., "Sensor Configuration Trade Study for Navigation in Near Rectilinear Halo Orbits," 2020. URL `https://doi.org/10.1007/s40295-020-00224-1`.

[3] Woffinden, D. C., Robinson, S. B., Williams, J. W., and Putnam, Z. R., "Linear Covariance Analysis Techniques to Generate Navigation and Sensor Requirements for the Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) Project," *AIAA SciTech Forum*, AIAA, San Diego, CA, 2019.

[4] Williams, J. W., c. Woffinden, D., and Putnam, Z. R., "Mars Entry Guidance and Navigation Analysis Using Linear Covariance Techniques for the Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) Project," *AIAA Scitech 2020 Forum*, 2020.

[5] Geller, D. K., Rose, M. B., and Woffinden, D. C., "Event Triggers in Linear Covariance Analysis with Applications to Orbital Rendezvous," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 1, 2009, pp. 102–111.

[6] Geller, D. K., and Christensen, D. P., "Linear Covariance Analysis for Powered Lunar Descent and Landing," *Journal of Spacecraft and Rockets*, Vol. 46, No. 6, 2009, pp. 1231–1248.

[7] Rose, M. B., and Geller, D. K., "Linear Covariance Techniques for Powered Ascent," *AIAA Guidance, Navigation, and Control Conference*, AIAA, Toronto, Ontario Canada, 2010.

[8] Markley, F. L., and Carpenter, J. R., "Generalized Linear Covariance Analysis," *The Journal of the Astronautical Sciences*, Vol. 57, No. 1-2, 2009, pp. 223–260.

[9] Berning, A. W., Girard, A., Kolmanosvsky, I., and D'Souza, S. N., "Rapid uncertainty and change-constrained path planning for small unmanned aerial vehicles," *Advanced Control Applications*, Vol. 2, No. 23, 2020.

[10] Arneberg, J., Tal, E., , and Karaman, S., "Guidance Laws for Partially-Observable Interception Based on Linear Covariance Analysis," *International Conference on Intelligent Robots and Systems (IROS)*, IEEE/RSJ, Madrid, Spain, 2018.

[11] Woods, J., "lincov," , 2021. URL `https://github.com/openlunar/lincov/tree/master/lincov`.

[12] Geller, D. K., "Linear Covariance Techniques for Orbital Rendezvous Analysis and Autonomous Onboard Mission Planning," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, 2006, pp. 1404–1414.

[13] Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A., "SymPy: symbolic computing in Python," *PeerJ Computer Science*, Vol. 3, 2017, p. e103. https://doi.org/10.7717/peerj-cs.103, URL https://doi.org/10.7717/peerj-cs.103.

[14] "U.S. Standard Atmosphere, 1976," Tech. Rep. NOAA-S/T-76-1562, 1976. URL https://ntrs.nasa.gov/citations/19770009539, NTRS Document ID: 19770009539 NTRS Research Center: Legacy CDMS (CDMS).

[15] Kinney, D. J., "Aerothermal Anchoring of CBAERO Using High Fidelity CFD," *AIAA Reno Conference*, 2007.